POLITECHNIKA LUBELSKA

Jerzy Montusiewicz
Elżbieta Miłosz
Maria Jarosińska-Caban

Podstawy programowania w języku C

Ćwiczenia laboratoryjne

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
{
  printf("Ile elementow?  ");
  int n,i; unsigned int t; float j=0;
  scanf("%d",&n); t=abs(n);
  float a[t],b[t];
  for(i=0;i<n;i++)
  {
   printf("Podaj %d elementy/ow  ",i+1);
   scanf("%f",&a[i]);
   if(a[i]>0)
   j=j+sqrt(a[i]);
   b[i]=j/(i+1);
  }
```

PODRĘCZNIKI

Lublin 2015

Naszym Rodzinom

Podręczniki – Politechnika Lubelska

Politechnika Lubelska
Wydział Elektrotechniki i Informatyki
ul. Nadbystrzycka 38A
20-618 Lublin

Jerzy Montusiewicz
Elżbieta Miłosz
Maria Jarosińska-Caban

Podstawy programowania w języku C

Ćwiczenia laboratoryjne

Politechnika Lubelska
Lublin 2015

Recenzent:
dr hab. Stanisław Grzegórski, prof. Politechniki Lubelskiej

Publikacja wydana za zgodą Rektora Politechniki Lubelskiej

ISBN: 978-83-7947-151-5

Wydawca: Politechnika Lubelska
ul. Nadbystrzycka 38D, 20-618 Lublin
Realizacja: Biblioteka Politechniki Lubelskiej
Ośrodek ds. Wydawnictw i Biblioteki Cyfrowej
ul. Nadbystrzycka 36C, 20-618 Lublin
tel. (81) 538-46-59, email: wydawca@pollub.pl
www.biblioteka.pollub.pl
Druk: TOP Agencja Reklamowa Agnieszka Łuczak
www.agencjatop.pl

Elektroniczna wersja książki dostępna w Bibliotece Cyfrowej PL www.bc.pollub.pl
Nakład: 100 egz.

Spis treści

Od Autorów

Język C opracowany przez Briana W. Kernighana i Dennisa M. Ritchie'go w 1971 r. jest pierwszym, podstawowym językiem, którego nauczanie prowadzone jest przez pracowników Instytutu Informatyki Politechniki Lubelskiej. Dotyczy to różnych kierunków kształcenia realizowanych na Wydziale Elektrotechniki i Informatyki (WEiI), takich jak Informatyka, Inżynieria biomedyczna oraz Mechatronika. Celem podręcznika przygotowanego dla studentów jest wprowadzenie pewnego standardu nauczania języka C, niezależnego od osoby prowadzącej oraz kierunku studiów. Dzięki temu studenci WEiI mogą opanować jednolite podstawy programowania na bazie języka strukturalnego.

Specyfika planu zajęć na WEiI polega na tym, że nauczanie języków programowania nie jest poprzedzone zajęciami z podstaw algorytmiki. Dlatego przygotowany podręcznik rozpoczynają dwa rozdziały nie związane bezpośrednio z językiem C, lecz dotyczące algorytmów i omówienia środowiska programistycznego Dev-C++. W strukturze podręcznika, bezpośrednio po omówieniu podstaw programowania imperatywnego (rozdział 3) umieszczono dział poświęcony wprowadzeniu funkcji standardowych i własnych (rozdział 4). Ma to na celu uświadomienie studentom, że taka metoda programowana jest działaniem powszechnym. Wieloletnie doświadczenie dydaktyczne, szczególnie w odniesieniu do młodzieży kierunków innych niż Informatyka, pokazało, że studenci mają poważne trudności, by przestawić się na inny tok myślenia, gdy z jednolitego kodu funkcji `main` muszą wydzielić funkcje własne realizujące określone działania.

W końcowej części podręcznika zamieszczono wykaz najważniejszych pojęć związanych z algorytmiką oraz programowaniem w języku C. Wykaz ten opracowano w języku polskim, angielskim oraz ukraińskim. W związku z faktem, że w ostatnim czasie na WEiI podejmuje naukę młodzież narodowości ukraińskiej, która w początkowym okresie studiowania miewa kłopoty z właściwym rozumieniem pojęć używanych na zajęciach, takie uzupełnienie uznano za zasadne.

W podręczniku do minimum ograniczono opis teoretyczny, skupiając się przede wszystkim na praktycznych aspektach programowania i rozwiązywania konkretnych zadań. Rozdziały dotyczące nauczania języka C mają jednolitą strukturę, w której można wydzielić: podstawy teoretyczne uzupełnione stosownymi przykładami, arkusz przykładowych ćwiczeń, w którym przedstawiono i opisano sposób ich rozwiązywania, a także arkusz zadań do samodzielnego wykonania. Zadania te mają zróżnicowany stopień trudności, z uwagi na potrzebę dostosowania ich do poziomu studentów z różnych kierunków studiów.

Autorzy mają nadzieję, że prezentowany podręcznik będzie użyteczny studentom Politechniki Lubelskiej w zdobyciu podstaw z programowania w języku C i ułatwi im przygotowywanie własnych programów.

1. Wprowadzenie do programowania. Algorytmy

Cel rozdziału

Poznanie podstawowych paradygmatów programowania w tym zasad programowania strukturalnego. Nabycie praktycznych umiejętności w tworzeniu algorytmów.

Podstawy teoretyczne

Paradygmat programowania określa sposób postrzegania przez programistę procedur tworzących program komputerowy. Paradygmat programowania to wzorzec postępowania, który w konkretnym etapie rozwoju informatyki lub w poszczególnych obszarach zastosowania, jest częściej używany od innych. W procesie programowania można wyodrębnić wiele paradygmatów, co prowadzi do programowania: proceduralnego, strukturalnego, funkcyjnego, imperatywnego, obiekwego, uogólnionego, zdarzeniowego, deklaratywnego, logicznego, aspektowego, agentowego lub modularnego. Niektóre języki programowania można nazwać językami hybrydowymi, ponieważ łączą w sobie wiele elementów programowania, np. proceduralnego, obiektowego oraz uogólnionego (język C++).

W przypadku języka C mamy do czynienia z programowaniem strukturalnym polegającym na podziale kodu źródłowego programu na funkcje oraz hierarchicznie usytuowane bloki przy wykorzystaniu instrukcji pętli i wyboru (struktury kontrolne). Takie programowanie znacznie zwiększa przejrzystość kodu oraz ułatwia jego czytelność, między innymi przez rezygnację z instrukcji skoków.

Programowanie w języku C zawiera następujące etapy:

1. Określenie celów programu.
2. Projektowanie programu (ALGORYTM).
3. Pisanie kodu.
4. Kompilacja.
5. Uruchomienie programu.
6. Testowanie i usuwanie błędów.
7. Pielęgnowanie i modyfikacja programu.

Algorytm można zdefiniować na wiele sposobów, np. algorytm to skończony, uporządkowany ciąg określonych działań lub czynności koniecznych do wykonania określonego zadania. W języku potocznym algorytm oznacza przepis (zbiór koniecznych czynności uporządkowanych we właściwej kolejności), dzięki któremu możemy sprawnie przygotować posiłek lub wymienić zużyte baterie w urządzeniu technicznym. Algorytmy można przedstawiać w postaci:

- opisowej (lista kroków, lub za pomocą tzw. pseudojęzyka – *ang. pseudocode*),
- graficznej:
  - schematy blokowe (*ang. flowcharts*),
  - schematy Nassi-Schneidermana (schematy N/S lub zwarte).

Przygotowanie algorytmu pozwala na sprawną implementację zaproponowanego rozwiązania na wybrany język programowania.

Pseudojęzyk

W pseudojęzyku wyróżniamy następujące konstrukcje:

- zdanie proste

```
przypisz średniej wartość zero
czytaj x
pisz wynik
```

- zdanie decyzyjne

```
jeżeli warunek to zdanie
lub
jeżeli warunek to zdanie1
w przeciwnym przypadku
zdanie2
```

- zdanie iteracyjne podczas gdy

```
podczas gdy warunek wykonuj zdanie
```

- zdanie iteracyjne powtarzaj

```
powtarzaj zdanie aż warunek
```

- zdanie iteracyjne dla

```
dla lista sytuacji wykonuj zdanie
```

- zdanie wybierz

```
wybierz przełącznik z
wartość1: zdanie1
...
w innym przypadku zdanie domyślne
```

- zdanie grupujące {...} lub begin ... end

```
{ zdanie1        begin zdanie1
  zdanie2              zdanie2
...                    ...
 }               end
```

Przykład 1.1

Obliczanie średniej arytmetycznej dwóch liczb (zdanie proste)

Lista kroków:

1. Wczytaj dwie liczby `a` i `b`
2. Dodaj do siebie te liczby i wynik podziel przez 2 → `(a+b)/2`
3. Wyświetl wynik. Zakończ algorytm.

Pseudokod:

1. Czytaj dwie liczby `a` i `b`
2. Przypisz średniej wartość: `srednia= (a+b)/2`
3. Pisz `srednia`.

Przykład 1.2 a

Obliczanie średniej ocen studenta (zdanie iteracyjne `dla`)
Student w czasie sesji zdaje n egzaminów. Obliczyć średnią ocen w sesji.

Lista kroków:

1. Wczytaj liczbę egzaminów `n`
2. Wyzeruj zmienną `s` → `s=0`
3. Wykonaj `n` razy:
   - wczytaj kolejną ocenę `x`
   - dodaj x do dotychczas obliczonej sumy `s` → `s=s+x`
4. Oblicz wartość średniej → `s=s/n`
5. Wyświetl wynik `s`. Zakończ algorytm.

Pseudokod:

1. Czytaj `n`
2. Przypisz s wartość zero → `s=0`
3. Dla zmiennej `i` zmieniającej się od `1` do `n` wykonuj:
   - {czytaj `x`
   - przypisz s wartość poprzednią +x → `s=s+x` }
4. Przypisz s wartość → `s=s/n`
5. Pisz `s`.

Przykład 1.2 b

Obliczanie średniej ocen studenta (zdanie iteracyjne `powtarzaj`)
Student w czasie sesji zdaje n egzaminów. Obliczyć średnią ocen w sesji.

Lista kroków:

1. Wczytaj liczbę egzaminów `n`
2. Wyzeruj zmienne: `s i licz → s=0; licz=0`

3. Powtarzaj:
   - wczytaj kolejną ocenę `x`
   - dodaj `x` do dotychczas obliczonej sumy `s` → `s=s+x`
   - zwiększ `licz` o 1 → `licz=licz+1`

   aż `licz` osiągnie wartość `n`
4. Oblicz wartość średniej → `s=s/n`
5. Wyświetl wynik `s`. Zakończ algorytm

Pseudokod:

1. Czytaj `n`
2. Przypisz zmiennym `s` i `licz` wartość zero: `s=0; licz=0`
3. Powtarzaj:
   - `{`czytaj `x`
   - przypisz `s` wartość poprzednią `+x` → `s=s+x`
   - przypisz `licz` wartość następną → `licz=licz+1}`

   aż `licz` osiągnie wartość `n`
4. Przypisz `s` wartość `s/n` → `s=s/n`
5. Pisz `s`

Przykład 1.3

Rozwiązywanie równania kwadratowego (zdanie decyzyjne)

Dane są współczynniki równania kwadratowego. Zbadać istnienie pierwiastków, gdy istnieją obliczyć je.

Lista kroków:

1. Wczytaj współczynniki równania `a, b, c, (a≠0)`
2. Oblicz wyróżnik delta → `d=b*b-4*a*c`
3. Jeśli `d<0`
   - wyświetl komunikat o braku pierwiastków
   - zakończ algorytm
4. Jeśli `d>0` to:
   - oblicz pierwiastki $x_1 = (-b + \sqrt{d})/(2*a) \quad x_2 = (-b - \sqrt{d})/(2*a)$
   - wyświetl wyniki
   - zakończ algorytm
5. Jeśli `d==0` to:
   - oblicz pierwiastek → $x = -b/(2*a)$
   - wyświetl wyniki
   - zakończ algorytm

Pseudokod (inna wersja algorytmu):

1. Czytaj `a, b, c, (a≠0)`
2. Przypisz d wartość → `d=b*b-4*a*c`
3. Jeśli `d<0` to: pisz: Brak pierwiastków
   w przeciwnym wypadku:
4. Jeśli `d>0` to:
   - przypisz `x1` i `x2` wartości:
     $x_1 = (-b + \sqrt{d})/(2*a)$ $x_2 = (-b - \sqrt{d})/(2*a)$
   - pisz `x1` i `x2`

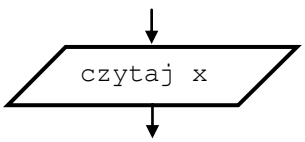
   w przeciwnym wypadku:
   - przypisz `x` wartość → $x = -b/(2*a)$
   - pisz `x`
   - zakończ algorytm.

Schematy blokowe

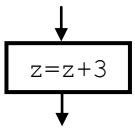
Schemat blokowy jest narzędziem prezentującym w graficznej postaci kolejność działań (czynności) w tworzonym algorytmie. Składa się figur geometrycznych (np. prostokąt, romb, owal) prezentujących różne działania oraz strzałek wskazujące powiązania między elementami schematu oraz kierunek przepływu informacji. Obecnie rzadko używane przez informatyków, ale dość często stosowane przez inżynierów innych branży. Podstawowe elementy:

Blok graniczny — Przedstawia początek, koniec, przerwanie lub wstrzymanie wykonywania działania, np. `blok końca programu`

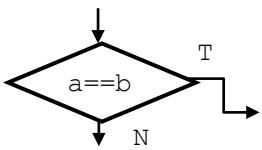
Blok wejścia-wyjścia — Oznacza czynność wprowadzania danych do programu i wyprowadzenia wyników obliczeń, np. `pisz a+7, czytaj x`



Blok operacyjny — Przedstawia wykonanie operacji, w efekcie której zmienią się wartości, postać lub miejsce zapisu danych, np. `z=z+3`

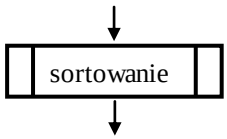


Blok decyzyjny, warunkowy — Pokazuje wybór jednego z dwóch wariantów wykonywania programu na podstawie sprawdzenia wpisanego warunku blok, np. `a==b`



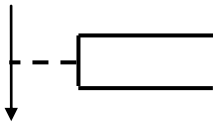
Blok podprogramu

Pokazuje podprogram zdefiniowany odrębnie, np. procedura (funkcja) `sortowanie`



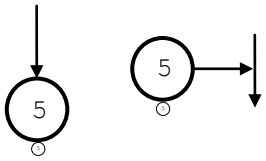
Blok komentarza

Wprowadzanie komentarzy wyjaśniających części schematu, ułatwia zrozumienie, np. wprowadzenie danych.



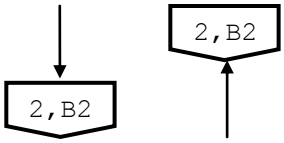
Łącznik wewnątrzstronicowy (wewnętrzny)

Służy do łączenia odrębnych części schematu znajdujących się na tej samej stronie, łączniki powiązane ze sobą mają identyczne oznaczenie, np. `B2,5`



Łącznik międzystronicowy (zewnętrzny)

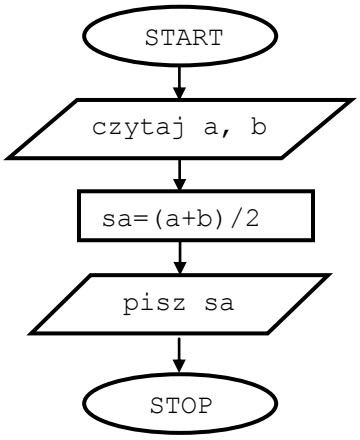
Pozwala łączyć odrębne części schematu znajdujące się na różnych stronach; łączniki powiązane ze sobą mają identyczne oznaczenie, np. `B1,5`, powinien zawierać numer strony, do której się odwołuje, np. `2,B2`



Przykład 1.4

Obliczanie średniej arytmetycznej dwóch liczb (algorytm liniowy)

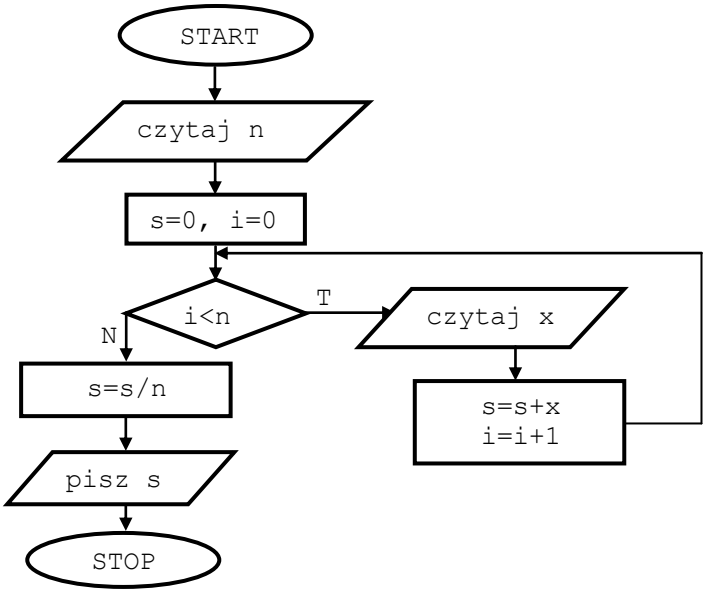
Powtórna realizacja przykładu 1.1



Przykład 1.5

Obliczanie średniej ocen studenta (algorytm iteracyjny)

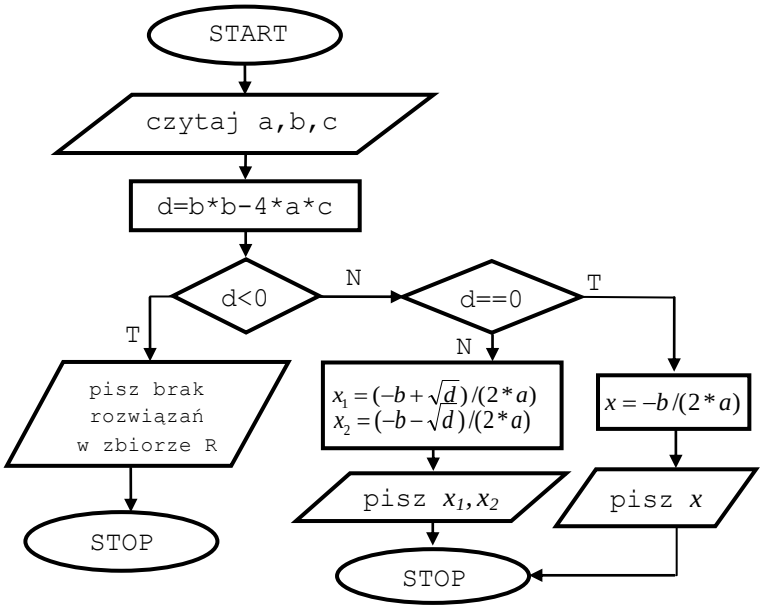
Powtórna realizacja przykładu 1.2



Przykład 1.6

Rozwiązywanie równania kwadratowego (algorytm decyzyjny)

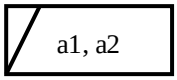
Powtórna realizacja przykładu 1.3



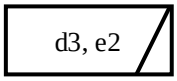
Schematy Nassi-Schneidermana (schematy N/S)

Schemat N/S jest graficznym przekazaniem kolejności działań w przygotowanym algorytmie. Pokazuje wprost główną ideę algorytmu – jego strukturę. W porównaniu do schematu blokowego jego forma jest bardziej zwarta. Nie występują w sposób jawny elementy łączące – strzałki. Domyślny kierunek powiązania elementów przebiega z góry na dół. Podstawowe elementy składowe to:

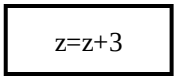
Blok wejścia – Wprowadzenie danych, klawiatura, plik



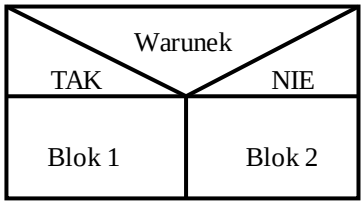
Blok wyjścia – Wyprowadzenie danych, monitor, drukarka, plik



Blok operacyjny – Przedstawia wykonanie operacji, np. `z=z+3`

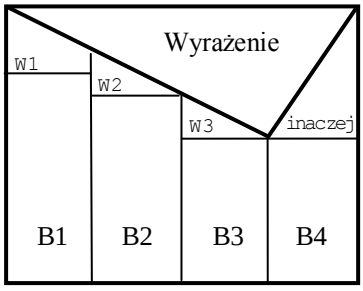


Blok decyzyjny – Pokazuje wybór jednego z dwóch wariantów wykonywania programu na podstawie sprawdzenia wpisanego warunku blok, np. `a==b`

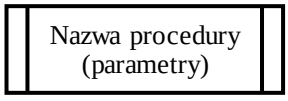


Blok wyboru z szeregu możliwości – Prezentuje wybór jednego wariantu z wielu możliwych

W1, W2 , … – stałe
B1, B2, … – bloki programowe



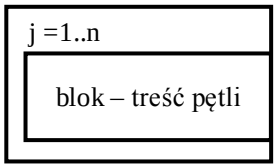
Blok wywołania procedury – Pokazuje odrębne nazwane fragmenty algorytmu



Blok iteracyjny (BI), pętla Dla

Z istniejącą zmienną sterującą

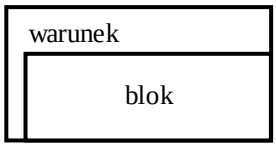
Przedstawia cykliczne wykonywanie bloku dla j=1, 2, …, n



BI, Dopóki

Dopóki <warunek> wykonuj <blok>

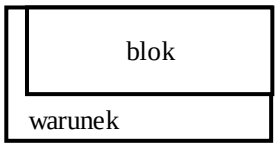
Pokazuje cykliczne wykonywanie bloku, gdy warunek sprawdzany na początku jest prawdziwy



BI, Powtarzaj

Powtarzaj <blok> aż <warunek>

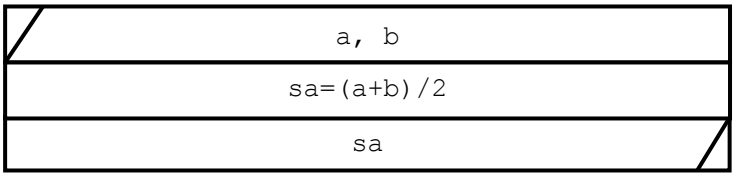
Pokazuje cykliczne wykonywanie bloku, gdy warunek sprawdzany na końcu jest prawdziwy



Przykład 1.7

Obliczanie średniej arytmetycznej dwóch liczb (algorytm liniowy)

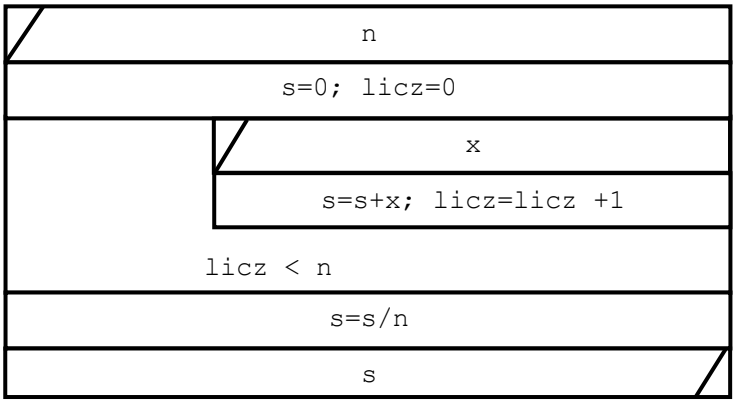
Powtórna realizacja przykładu 1.1



Przykład 1.8

Obliczanie średniej ocen studenta (algorytm iteracyjny)

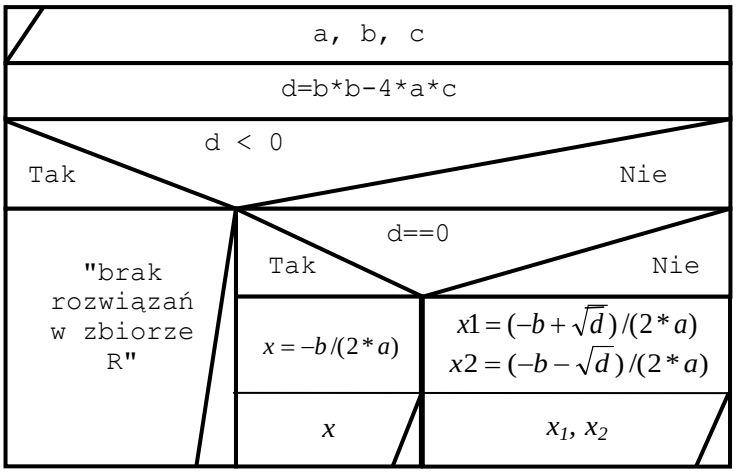
Powtórna realizacja przykładu 1.2



Przykład 1.9

Rozwiązywanie równania kwadratowego (algorytm decyzyjny).

Powtórna realizacja przykładu 1.3



Arkusz przykładowych ćwiczeń

Ćwiczenie 1.1

Ciąg Fibonacciego*

Stosując pseudokod zbuduj algorytm obliczający i wyświetlający elementy ciągu Fibonacciego. Pierwsze dwa elementy są równe 1: `f(1)=1, f(2)=1`. Następne obliczane są z następującej zależności: `f(n)=f(n-2)+f(n-1)`.

Wynik:

Założenia: zmienna `a`- przechowuje wyraz `n-2`, zmienna `b` wyraz `n-1`

```
czytaj n
przypisz a wartość 0
przypisz b wartość 1
dla i zmieniającego się od 1 do n wykonuj:
   {pisz b
    przypisz b wyraz następny: b=b+a
    przypisz a wyraz b: a=b-a
    }
```

* Leonardo z Pizy zwany Fibonaccim (Filius Banacci) autor traktatu „Liber abaci” (Księga liczydeł) z 1202 r., z którego wyrosła cała późniejsza matematyka i ekonomia finansowa (zastosowanie systemu dziesiętnego przejętego od arabskich muzułmanów, ale opracowanego w Indiach).

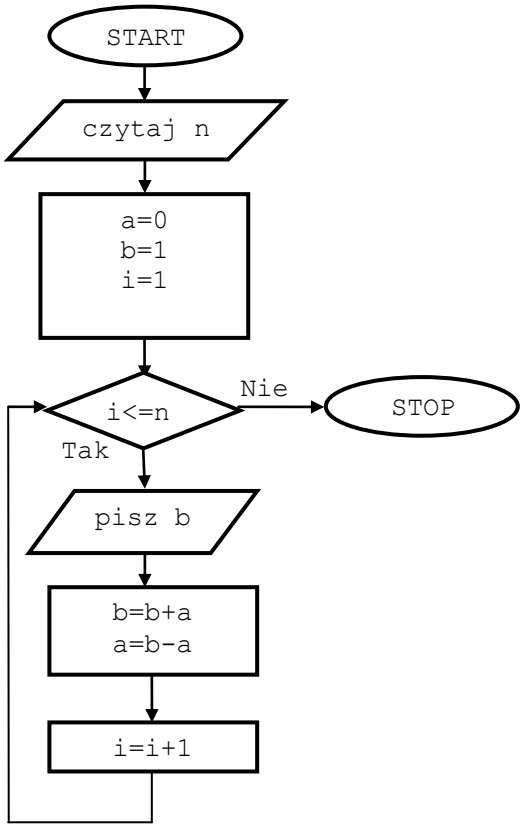
Ćwiczenie 1.2

Ciąg Fibonacciego

Powtórna realizacja ćwiczenia 1.1

Stosując schemat blokowy zbuduj algorytm obliczający elementy ciągu Fibonacciego.

Wynik:



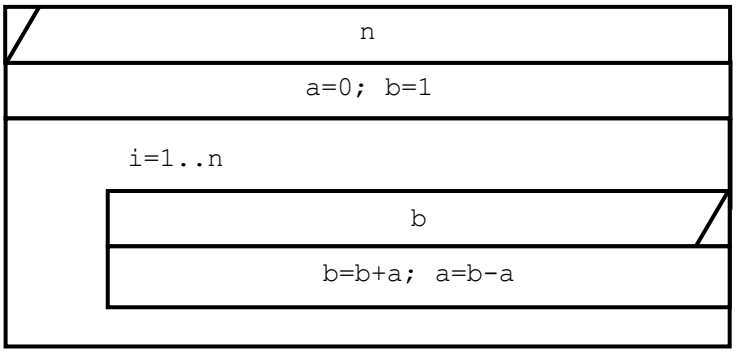
Ćwiczenie 1.3

Ciąg Fibonacciego

Powtórna realizacja ćwiczenia 1.1

Stosując schemat N/S zbuduj algorytm obliczający elementy ciągu Fibonacciego.

Wynik:



Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 1.1

Obliczanie pola powierzchni i objętości brył geometrycznych.

(*) Oblicz pole powierzchni i objętość sześcianu, prostopadłościanu, kuli i walca o podanych wymiarach.
Wykonaj schemat N/S.

Zadanie 1.2

Obliczanie zapotrzebowania na farbę do malowania pokoju

(*) Oblicz liczbę puszek farby potrzebnych do pomalowania ścian pokoju o wymiarach `a x b x c` (w metrach) jeśli 1 puszka farby (1 litr) wystarcza na pomalowanie `Y` m^2. Okno w pokoju ma wymiary `1 x 0,9` (w metrach.), zaś drzwi: `2 x 0,9` m. Pokój ma dwa okna i jedne drzwi.
Wykonaj schemat N/S.

Zadanie 1.3

Klasyfikacja wzrostu osoby (osób)

(*)
1. Określ, czy dana osoba jest niskiego, średniego czy wysokiego wzrostu.
2. Dokonaj takiej klasyfikacji wzrostu dla N studentów.

Wykonaj odpowiedni schemat N/S.

Zadanie 1.4

Wybranie najlepszej oceny w sesji studenta

(**) Student PL w czasie sesji zimowej zdaje N egzaminów.
a) podaj najlepszą ocenę,
b) podaj z którego egzaminu ocena była najlepsza.
Wykonaj schemat N/S.

Zadanie 1.5

Obliczenie wypłat dla grupy pracowników

(**) Wypłata dla pracownika składa się ze stawki bazowej, dodatku stażowego i premii. Dodatek stażowy przysługuje po 5 latach i wynosi:

$$dodatek = \begin{cases} 20\% & gdy\ lata > 20 \\ tyle\ \%\ ile\ lat\ \ pracy & gdy\ lata \in <5,20> \end{cases}$$

Sporządzić listę wypłat dla N pracowników i sumaryczną wartość wypłat.
Wykonaj schemat N/S.

Zadanie 1.6

Fundusz emerytalny

(**) Pracownik pewnej firmy wpłaca do funduszu emerytalnego X zł składki. Fundusz potrąca z niej Y%, a resztę inwestuje osiągając stały dochód Z% w skali roku. Jaką kwotą będzie dysponował pracownik po N latach pracy?
Wykonaj schemat N/S.

Zadanie 1.7

Logowanie do systemu

(**) Dostęp do systemu realizowany jest po podaniu odpowiedniego hasła. Narysuj schemat NS programu umożliwiającego logowanie użytkownika (max 5 prób) i przekazanie tajnej wiadomości. Po 5 nieudanych próbach następuje wyjście z programu. Po poprawnym zalogowaniu się użytkownik podaje wiadomość, która jest szyfrowana (do każdego znaku dodawana jest pewna liczba). Wiadomość może być wyświetlona w zaszyfrowanej formie, lub wyświetlona w oryginale (po zastosowaniu odpowiedniego dekodera).

Zadanie 1.8

Zawody

(**) W zawodach sportowych w każdej konkurencji bierze udział N zawodników. Wczytaj wyniki kilku konkurencji (np. biegu na 100 m, rzutu oszczepem) i wyświetl je w kolejności od najlepszego do najgorszego wyniku. Podaj 3 pierwsze miejsca.

Wykonaj schemat N/S.

2. Praca w wybranym środowisku programistycznym

Cel rozdziału

Poznanie pracy w zintegrowanym środowisku programistycznym Dev-C++, które umożliwia tworzenie kodu programu w języku C, jego kompilację, linkowanie, uruchomienie, a także identyfikację powstających błędów. Nabycie praktycznych umiejętności posługiwania się tym środowiskiem programistycznym.

Podstawy teoretyczne

Kod źródłowy programu w języku C można napisać w dowolnym edytorze tekstu, który zapisuje znaki w kodzie ASCII, np. notatniku. Efektywniej jest to jednak wykonywać w specjalizowanych środowiskach. Obecnie dostępnych jest wiele bezpłatnych środowisk programistycznych, w których można programować w języku C, a także w C++. Istniejące programy różnią się zasadami licencji. Program Dev-C++ jest środowiskiem programistycznym dostępnym bezpłatnie na licencji GPL wykorzystującym kompilator GCC (GNU Compiler Collection), który może pracować na różnych platformach: Linux, DOS, WIN32, OS/2. Pakiety instalacyjne można pobrać ze strony:

`http://www.bloodshed.net/devcpp.html`.

Dev-C++ znamionuje niewielki rozmiar, stabilne działanie, wzajemna przenośność kodów utworzonych na platformach Windows oraz Linux oraz istnienie polskiej wersji językowej interfejsu.

Kompilator

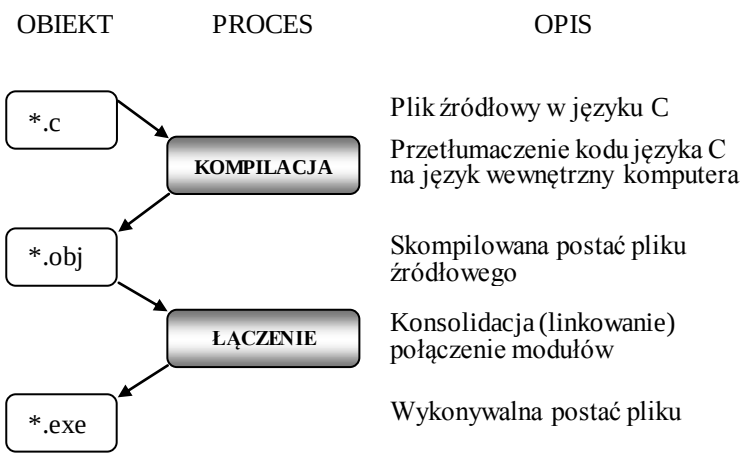
Kompilator pozwala na przekształcenie kodu źródłowego do kodu wynikowego pozwalający na bezpośrednie uruchomienie go na komputerze. Proces kompilacji pozwala na sprawdzenie poprawności składni napisanego kodu (np. brak średnika czy niewłaściwa nazwa). Wykrycie błędu nie pozwala na wygenerowanie kodu wynikowego. Uzyskana lista błędów wraz z opisem pozwala na łatwiejszą korektę

kodu źródłowego. Kompilator może wyświetlić również ostrzeżenia (ang. warning), które mogą dotyczyć błędów logicznych. W takim przypadku kod wynikowy zostanie wykonany. Proces kompilacji składa się z następujących etapów: prekompilacji, właściwej kompilacji, optymalizacji kodu asemblera oraz asemblacji.

Linker

Linker (konsolidator) odpowiada za operacje zmierzające do utworzenia pliku binarnego z kodem wykonywalnym. Dzięki niemu następuje połączenie skompilowanych wersji kodów źródłowych ze wskazanymi w nich funkcjami niezdefiniowanymi, które znajdują się w przyłączonych bibliotekach. Ponadto następuje przypisanie kodu maszynowego do ustalonych adresów.

Schemat procesu otrzymywania programu do uruchomienia.

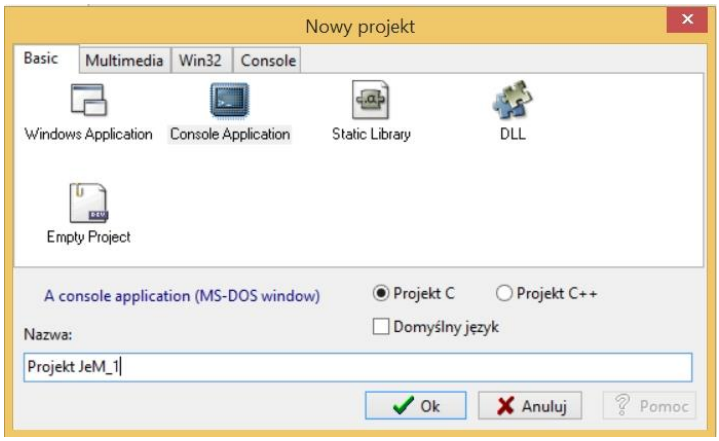


Sposób pracy z programem Dev-C++ może nieznacznie różnić się w zależności od zainstalowanej wersji. Więcej praktycznych informacji można znaleźć w licznych *tutorialach* dostępnych na stronach internetowych.

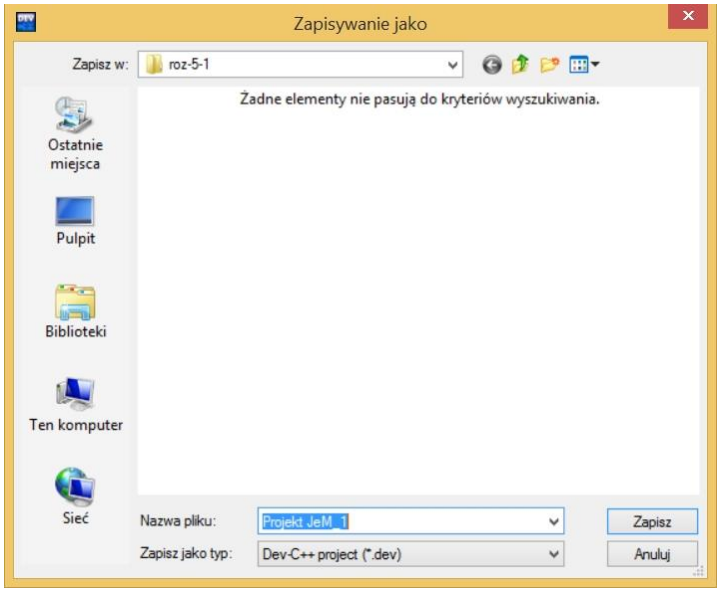
Skrócony opis najważniejszych funkcji

Tworzenie nowego projektu

1. Z menu *Plik* wybrać opcję ***Nowy***, a następnie ***Projekt…***
2. Wybrać ***Console Aplication*** oraz ***Projekt C***.
3. W polu ***Nazwa*** wpisać własną lub pozostawić nazwę systemową. Zatwierdzić *OK*.

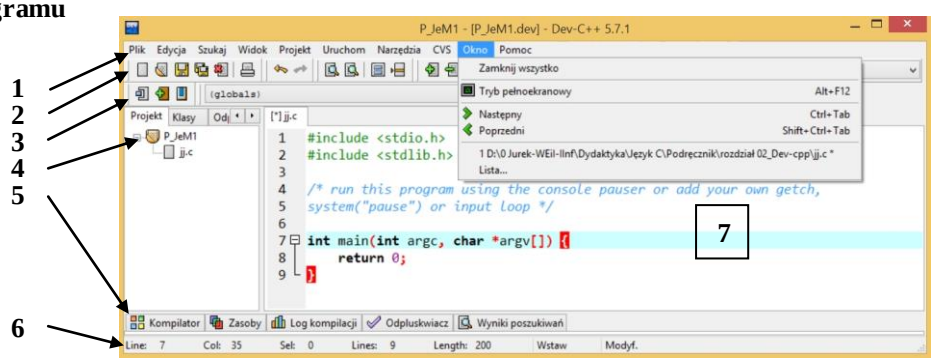


4. Wybrać miejsce (katalog) zapisu, zatwierdzić przyciskiem *Zapisz*.



Powyższe działania prowadzą do uzyskania okna edycyjnego do wpisywania kodu programu.

Okno główne programu



1 – Menu główne programu, rozwijalne
2 – Włączone paski narzędzi: od lewej: *Główny, Edycja, Przeszukiwanie; Projekt, Kompilowanie i uruchamianie* (zasłonięty przez rozwinięte okno)
3 – Włączone paski narzędzi: *Dodatki, Klasy*
4 – Przeglądarka projektu / klas
5 – Pasek zakładek (*Kompilator, Zasoby, Log kompilacji, Odpluskwiacz, Wynik poszukiwań*)
6 – Pasek stanu
7 – Edytor

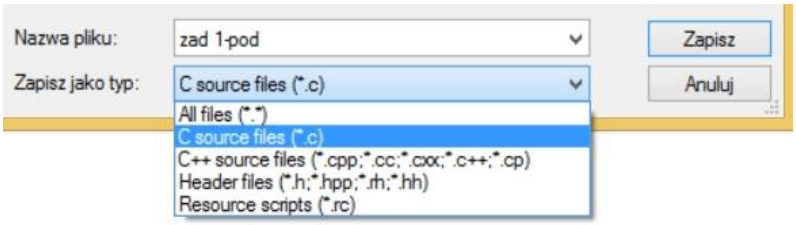
Tworzenie kodu źródłowego programu

Edytor środowiska Dev-C++ jest edytorem specjalizowanym, który ułatwia programiście pracę nad tworzeniem kodu źródłowego programu. Między innymi edytor ten barwi lub pogrubia rozpoznane typy tekstów, automatycznie generuje pewne wybrane znaki.

1. Rozpoznane dyrektywy preprocesora (znak #) wyświetlane są na zielono.
2. Teksty komentarza (znajdujące się pomiędzy znakami /* … */ lub poprzedzone znakiem //) przyjmują kolor niebieski.
3. Rozpoznane słowa kluczowe języka C są pogrubiane.
4. Nawiasy (), [], {} oraz znaki typu kropka, przecinek, dwukropek, średnik, gwiazdka, operatory wyświetlane są na kolor czerwony.
5. Otwarcie dowolnego nawiasu powoduje automatycznie dostawienie nawiasu zamykającego.
6. Wpisanie znaku cudzysłowu tworzy automatycznie dopisanie drugiego znaku, który zamyka napisaną sekwencję – kolor granatowy.
7. Po lewej stronie pojawia się numeracja linii programu – ułatwia to identyfikację ewentualnych błędów przygotowanego kodu programu.
8. Obok pojawia się linia pozwalająca na kontrolę zamknięcia obszaru przez nawiasy klamrowe (ciała funkcji lub bloku instrukcji).

Zapisywanie zmian kodu programu

Przy zapisywaniu nowej wersji programu należy pamiętać aby wybrać właściwy typ pliku – rozszerzenie **.c** (domyślnie środowisko Dev-C++ ustawiono na rozszerzenie .cpp).



Kompilacja, uruchamianie programu

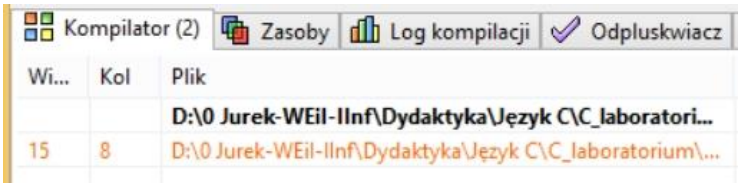
Kompilację programu wykonujemy przez kombinację klawiszy ***Ctrl+F9*** lub z menu wybrać opcję ***Uruchom*** -> ***Kompiluj***  . Kompilację z jednoczesnym uruchomieniem programu wykonujemy przez kombinację klawiszy ***Ctrl+F11*** lub z menu wybrać opcję ***Uruchom*** -> ***Kompiluj i uruchom***  .

Fragment programu

Program oblicza pewne wyrażenie arytmetyczne

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[])
5  {
6    printf("Ile elementow?  ");
7    int n,i; unsigned int t; float j=0;
8    scanf("%d",&n); t=abs(n);
9    float a[t],b[t];
10   for(i=0;i<n;i++)
11   {
12    printf("Podaj %d elementy/ow  ",i+1);
13    scanf("%f",&a[i]);
14    if(a[i]>0)
15    j=j+sqrt(a[i]);
16    b[i]=j/(i+1);
17   }
```

Wynik kompilacji



dalszy komunikat …

WARNING !

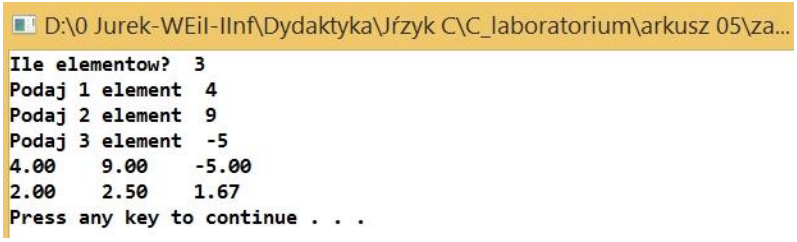
Kompilator wypisał ostrzeżenie dotyczące wiersza 15. w 8. kolumnie. W tym wypadku chodzi o fakt, że wbudowana funkcja `sqrt` nie została jawnie zadeklarowana, a jej włączenie zostało dokonane domyślnie. Kod wynikowy programu został wykonany i program może być uruchomiony. Dopisanie linii

```
#include <math.h>
```

likwiduje to ostrzeżenie. Uzyskany komunikat jest następujący.

```
- Output Filename: D:\0 Jurek-WEiI-IInf\Dydaktyka\Język C\
- Output Size: 130,0048828125 KiB
- Compilation Time: 0,17s
```

Wykonanie programu



Do zatrzymania działania programu zastosowano następującą linię kodu.

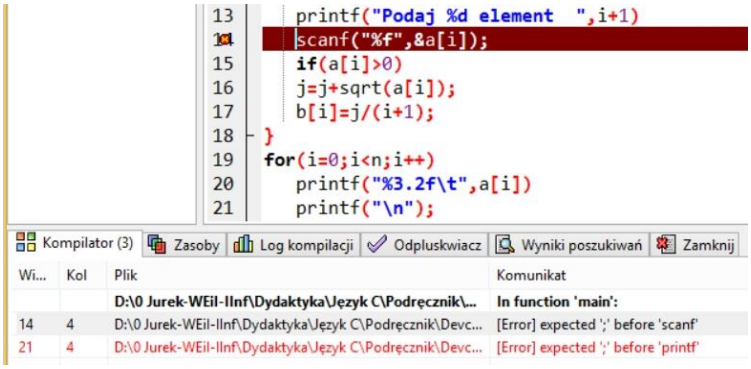
```
system("PAUSE");
```

Po wciśnięciu dowolnego klawisza uzyskano komunikat

```
--------------------------------
Process exited after 34.12 seconds with return value 0
Press any key to continue . . .
```

Program zakończył działanie.

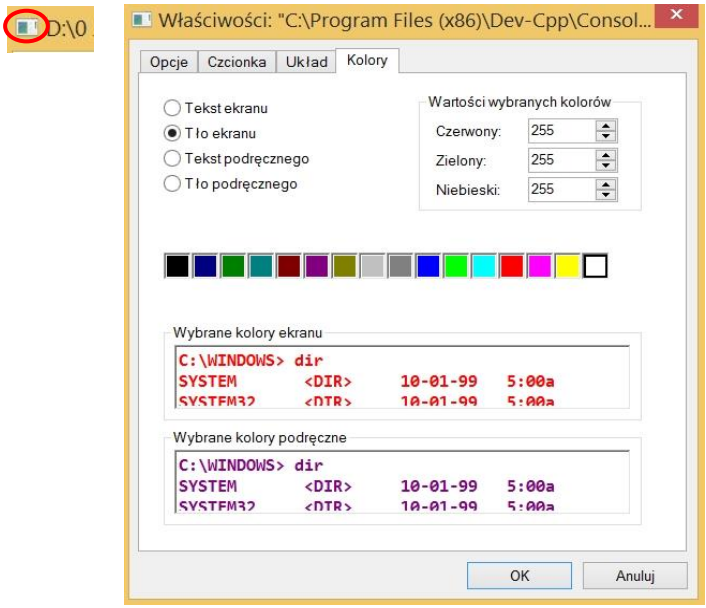
Kompilacja programu z błędnym kodem



Kompilator wskazuje na dwa błędy [Error]. W obu przypadkach wskazuje na brak znaku średnika (expected ';' – oczekuje ';'). Należy zwrócić uwagę, że wyświetlany jest wiersz następny w stosunku do linii, w której znajduje się błąd – Wi… 14 oraz komentarz before 'scanf', a w drugim przypadku Wi… 21 oraz komentarz before 'printf'. Kod wynikowy programu nie został wykonany.

Wciśnięcie lewego klawisza myszy na małym okienku w lewym górnym rogu włącza menu pozwalające zmienić ustawienia okna wyświetlającego wykonywanie programu. Wybierając zakładkę ***Właściwości*** można na nowo zdefiniować między innymi kolor ekranu, czcionki lub jej rozmiar.

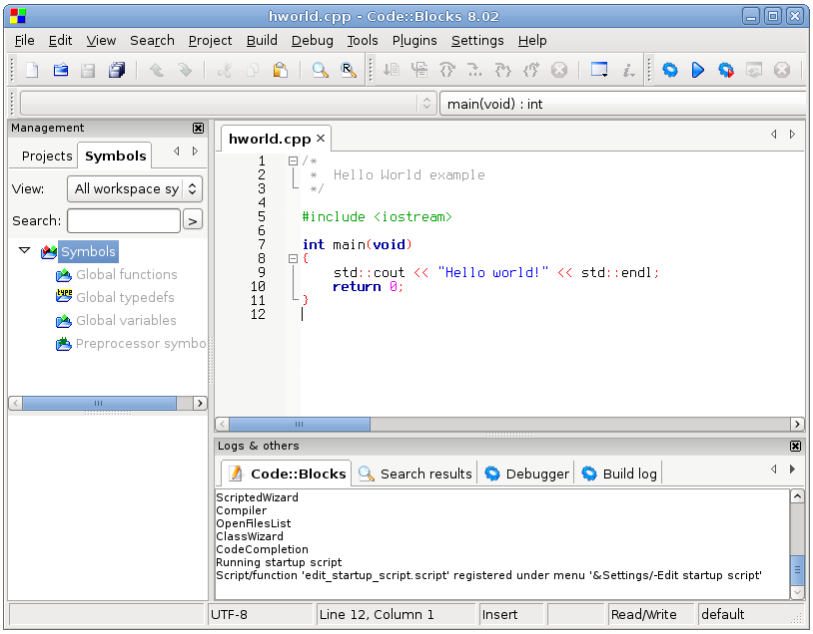
Edycja okna wykonania programu



Code:Blocks

Innym popularnym i darmowym środowiskiem programistycznym do programowania w języku C dla Windows, jak i Linux jest **Code::Blocks** dostępny na stronie `www.codeblocks.org`.

Code::Blocks to wieloplatformowe, zintegrowane środowisko programistyczne (IDE) na licencji GNU, oparte na projekcie Scintilla do programowania w języku C i C++.



3. Proste programy imperatywne. Wprowadzanie i wyprowadzanie danych. Instrukcja przypisania. Wyrażenia

Cel rozdziału

Zaznajomienie z realizacją prostych algorytmów realizowanych sekwencyjnie i programowaniem imperatywnym. Nabycie praktycznych umiejętności pisania prostych programów imperatywnych z wykorzystaniem wyrażeń różnych typów.

Podstawy teoretyczne

Program w języku C jest zbiorem modułów. Każdy moduł składa się z globalnych deklaracji typów, zmiennych, funkcji. Dokładnie jeden moduł zawiera definicję funkcji głównej (`main`). Wykonanie programu polega na opracowaniu jego globalnych deklaracji, a potem na wykonaniu instrukcji funkcji main. Zakończenie programu następuje po wykonaniu instrukcji powrotu (`return`) w funkcji main lub po powrocie z funkcji `exit`. Instrukcje przeznaczone dla kompilatora (dyrektywy preprocesora) rozpoczynają się znakiem # i nie kończą średnikiem. Wielkość liter w identyfikatorach (nazwach własnych) ma znaczenie. Dobrą praktyką jest stosowanie małych liter do identyfikatorów zmiennych, a dużych do identyfikatorów stałych.

Komentarz jednowierszowy oznaczany jest po znakach `//komentarz`. Komentarz wielowierszowy zawarty jest między znakami: `/* komentarz */`. Prosty program imperatywny w języku C posiada następującą strukturę:

```
//dyrektywy preprocesora
int main() //naglowek funkcji głownej
{
//deklaracje
//instrukcje
return 0;  //instrukcja powrotu
}
```

Przykład 3.1

```
#include <stdio.h> /* dolaczenie pliku
    nagłówkowego, szczegoly w rozdziale 15 */
int main(void)   /*definicja funkcji main*/
{
 int liczba, wynik;   /*deklaracja zmiennych
                        liczba, wynik*/
 printf("Prosty program ");/*wywołanie funkcji
                             wyjscia*/
 printf("w C\n");
 printf("Podaj liczbe ");
 scanf("%d", &liczba);    /*wywołanie funkcji
                            wejscia*/
 wynik=2*liczba;        /*instrukcja przypisania*/
 printf("Wynik to %d",wynik);
 return 0;      /*instrukcja powrotu*/
}
```

Elementy programu:

- ***typy danych*** podstawowe (*int, float, double, char*) i ich modyfikatory (*long, short, signed, unsigned*)
- ***zmienne***

```
typ zmienna;
typ zmienna=wartość;
```

- ***stałe***

```
#define nazwa wartość //dyrektywa preprocesora
const typ nazwa=wartość;
```

Przykład 3.2

```
#define pi 3.14
...
char odp, znak; //zmienne znakowe
float wynik;    //zmienna rzeczywiste
int x=1, y=2; char znak='k'; double z=1.5;
const float PODATEK=0.22, PREMIA=0.50;
```

Wyprowadzanie danych na ekran – funkcje: `printf()`, `putchar()` z pliku nagłówkowego `<stdio.h>` lub `putch(znak)` z pliku nagłówkowego `<conio.h>`

```
printf("łańcuch sterujący", argumenty);
```

```
putchar(znak);
putch(znak);
```

Wczytywanie danych z klawiatury – funkcje: `scanf()`, `getchar()` z pliku nagłówkowego `<stdio.h>` lub `getch(znak)`, `getche(znak)` z pliku nagłówkowego `<conio.h>`

```
scanf("łańcuch sterujący", argumenty);
getchar(znak);
getch(znak);
getche(znak);
```

Przykład 3.3

```
char znak;
int liczba_c;
float liczba_rz;
char imie[20];
printf("Podaj znak, liczbę całkowita i liczbe
  rzeczywista\n");
scanf("%c %d %f", &znak, &liczba_c, &liczba_rz);
printf("Podales znak: %c, liczbe calkowita: %d,
  liczbe rzeczywista:%f\n",znak, liczba_c,
  liczba_rz );
printf("Podaj imie\n");
scanf("%s", imie);
printf("Witaj %s\n", imie);
```

PAMIĘTAJ !

& ampersant *– ten znak musi poprzedzać nazwę zmiennej prostego typu (elem tablicy także), której wartość wprowadzamy z klawiatury.*

Polskie litery w Dev C++

Aby wyprowadzić na ekran teksty zawierające polskie znaki narodowe można dołączyć plik nagłówkowy **`locale.h`** i wywołać funkcję **`setlocale`** z parametrami **`LC_ALL`** i **`""`**.

Przykład 3.4

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
int main(int argc, char *argv[]) {
    setlocale(LC_ALL,"");
     printf("ąęółżź");
     return 0;
}
```

Wyrażenia i operatory

Wyrażenie to kombinacja operatorów i operandów np. w wyrażeniu

```
suma=x+y
```

symbole `'='` oraz `'+'` to operatory, a `suma, x, y` to operandy.
Podstawowe typy operatorów:

- arytmetyczne: +, -, *, /, % (modulo – reszta z dzielenia), ++, -- (inkrementacja, dekrementacja),
- przypisania: =, +=, -=, *=, /= ,
- relacyjne: < <= > >= == (równe) != (różne),
- logiczne: && (koniunkcja I/AND), || (alternatywa LUB/OR), ! (negacja NIE/NOT).

Instrukcje

Instrukcje to główne elementy programu – polecenia wydawane komputerowi – wyrażenie zakończone średnikiem.
Podstawowe typy instrukcji:

- przypisania,
- warunkowa,
- wyboru,
- iteracyjne,
- wywołania funkcji.

Instrukcja przypisania

```
zmienna=wartość;
zmienna+=wartość; //zmienna=zmienna+wartosc;
zmienna-=wartość; //zmienna=zmienna-wartosc;
zmienna*=wartość; //zmienna=zmienna*wartosc;
zmienna/=wartość; //zmienna=zmienna/wartosc;
```

Przykład 3.5

```
char znak;
int liczba_c;
float liczba_rz;
liczba_rz=liczba_c=znak='A';
printf("%c %d %0.2f\n", znak, liczba_c,
liczba_rz);
znak++;            //znak='B'
liczba_c+=znak*2; //liczba_c=65+66*2=197
liczba_rz-=znak;  //liczba_c=65.0+66.0=-1.00
printf("%c %d %0.2f\n", znak, liczba_c,
liczba_rz);
```

Arkusz przykładowych ćwiczeń

Ćwiczenie 3.1

Struktura prostego programu imperatywnego

Napisz program wyświetlający informacje o autorze programu, obliczający pole koła i wypłatę dla 2 pracowników.

Wynik:

Program może wyglądać w postaci przedstawionej na listingu 3.1:

Listing 3.1. Prosty program liniowy

wersja I

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5
6 int main(int argc, char *argv[])
7 { //deklaracje
8  int ROK=1;
9  float PREMIA=0.20, PI=3.14159;
10  float r, pole, stawka, wyplata;
11  int lg;
12  //instrukcje
13  printf("========================\n");
14  printf("************************\n");
15  printf("========================\n");
16  printf("Programowanie liniowe\n");
17  printf("Autor programu: %s, kierunek: %s, rok:
18   %d\n", "Jan Kowalski", "Informatyka", ROK);
19  printf("========================\n");
20  printf("************************\n");
21  printf("========================\n");
22  printf("Podaj promien kola\n"); scanf("%f",&r);
23  pole=PI*r*r;
```

```
24  printf("Pole kola o promieniu %0.2f =
25   %0.2f\n",r,pole);
26  printf("========================\n");
27  printf("************************\n");
28  printf("========================\n");
29  printf("Pracownik1\n");
30  printf("Podaj liczbe godzin\n");
31  scanf("%d",&lg);
32  printf("Podaj stawke\n"); scanf("%f",&stawka);
33  wyplata= lg*stawka+ lg*stawka*PREMIA;
34  printf("Wyplata = %0.2f\n",wyplata);
35  printf("Pracownik2\n");
36  printf("Podaj liczbe godzin\n");
37  scanf("%d",&lg);
38  printf("Podaj stawke\n"); scanf("%f",&stawka);
39  wyplata= lg*stawka+ lg*stawka*PREMIA;
40  printf("Wyplata = %0.2f\n",wyplata);
41  printf("========================\n");
42  printf("************************\n");
43  printf("========================\n");
44  system("PAUSE");
45  return 0;
46 }
```

WYKONANIE PROGRAMU

```
=========================
*************************
=========================
Programowanie liniowe
Autor programu: Jan Kowalski, kierunek: Informatyka, rok: 1
=========================
*************************
=========================
Podaj promien kola
7.5
Pole kola o promieniu 7.50 = 176.71
=========================
*************************
=========================
Pracownik1
Podaj liczbe godzin
30
Podaj stawke
8.5
Wyplata = 306.00

Pracownik2
Podaj liczbe godzin
24
Podaj stawke
11.3
Wyplata = 325.44
=========================
*************************
=========================
Press any key to continue . . .
```

Komentarz do programu:

Wiersze:

- 1,2: dyrektywa kompilatora dołączająca pliki nagłówkowe `stdio.h, stdlib.h`
- 6: nagłówek funkcji głównej `main`
- 8,9: definicje zmiennych z jednoczesną inicjalizacją
- 10,11: deklaracje zmiennych
- 13-16: wyświetlenie tekstu z przejściem do nowego wiersza (\n)

17,18: wyświetlenie tekstu i wartości stałych
22: wyświetlenie tekstu i wczytanie z klawiatury wartości zmiennej (&r)
23: instrukcja przypisania
24-25: wyświetlenie tekstu i wartości zmiennych
44: wywołanie funkcji zatrzymującej ekran konsoli
45: zwrócenie wartości 0 przez funkcję główną `main`
46: koniec funkcji głównej `main` i koniec programu

Listing 3.1. Prosty program imperatywny

wersja II po modyfikacji

```
1-2 bez zmian
3 #define PI 3.14159  //stale
4 #define AUTOR "Jan Kowalski"
5 #define  KIERUNEK "Informatyka"
6 int main(int argc, char *argv[])
7 { //deklaracje
8  const int ROK=1; //stala
9  const float PREMIA=0.20;
12-18 bez zmian
19  printf("Autor programu: %s, kierunek: %s, rok:
20   %d\n", AUTOR, KIERUNEK, ROK);
21-46 bez zmian
```

Komentarz do programu:

Wiersze:

3-5: definicje stałych, z wykorzystaniem preprocesora
8,9: definicje zmiennych modyfikator const – tworzy z nich wartości stałe
19,20: wyświetlenie tekstu i wartości stałych: `AUTOR`, `KIERUNEK` zadeklarowanych w wierszach 4 i 5

Ćwiczenie 3.2

Struktura programu imperatywnego

Napisz program, który dla 2. liczb całkowitych obliczy i wyświetli ich sumę, różnicę, iloczyn, iloraz i modulo.

Wynik:

Program może wyglądać w postaci przedstawionej na listingu 3.2:

Listing 3.2. Prosty program imperatywny

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(int argc, char *argv[])
5 {int x,y,z;
6  float f;
7  printf("Podaj 2 liczby calkowite\n");
8  scanf("%d %d", &x,&y);
9  z=x+y;
10 printf("suma: %d \n",z);
11 z=x-y;
12 printf("roznica: %d \n",z);
13 z=x*y;
14 printf("iloczyn: %d \n",z);
15 z=x/y;
16 printf("iloraz - wynik dzielenia calkowitego:
17         %d\n",z);
18 z=x%y;
19 printf("iloraz - reszta z dzielenia calkowitego:
20 %d\n",z);
21 f=(float)x/y;  // f=1.0*x/y;
22 printf("iloraz - wynik dzielenia rzeczywistego:
23         %f\n",f);
24 /*printf("iloraz - wynik dzielenia
rzeczywistego:
25          %0.2f\n",f);*/
26
27  system("PAUSE");
28  return 0;
29 }
```

Komentarz do programu:

Wiersze:

1,2:	dyrektywa kompilatora dołączająca pliki nagłówkowe `stdio.h, stdlib.h`
4:	nagłówek funkcji głównej `main`
5,6:	deklaracje zmiennych
7:	wyświetlenie tekstu z przejściem do nowego wiersza (\n)
8:	Wczytanie wartości dwóch zmiennych z klawiatury (&)
9,11,13,15,18:	instrukcje przypisania
10,12,14:	wyświetlenie tekstu i wartości zmiennej
21:	instrukcja przypisania z rzutowaniem jawnym typu, alternatywna wersja po komentarzu //
24,25:	W komentarzu /* */ alternatywna wersja wyświetlenia wartości rzeczywistej z formatem (2 miejsca po przecinku)
27:	wywołanie funkcji zatrzymującej ekran konsoli
28:	zwrócenie wartości 0 przez funkcję główną main
29:	koniec funkcji głównej main i koniec programu

WYKONANIE PROGRAMU

```
Podaj 2 liczby calkowite
3 -4
suma: -1
roznica: 7
iloczyn: -12
iloraz - wynik dzielenia calkowitego: 0
iloraz - reszta z dzielenia calkowitego: 3
iloraz - wynik dzielenia rzeczywistego: -0.750000
Press any key to continue . . .
```

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 3.1

Dane osobowe i adresowe

(*) Napisz program, który na podstawie wprowadzonych z klawiatury danych i zdefiniowanych stałych (*status* przyjmujący wartość *student* i *średnia ocen* przyjmująca wartość twojej porządanej średniej – liczba rzeczywista) wyświetli w jednym wierszu imię, nazwisko, pesel i płeć, a w drugim dane adresowe: kod, miasto, ulica, nr domu, nr mieszkania, a w trzecim twój status.

Zadanie 3.2

Obliczanie pola figur geometrycznych

(*) Napisz program obliczający pole powierzchni i obwód figur: kwadratu, prostokąta, trapezu, trójkąta.

Zadanie 3.3

Obliczanie objętości i pola powierzchni brył

(*) Napisz program obliczający objętość i pole powierzchni sześcianu, prostopadłościanu o podstawie kwadratowej i prostokątnej oraz walca.

Zadanie 3.4

Obliczanie średniej arytmetycznej i średniej geometrycznej dwóch liczb całkowitych

(*) Napisz program obliczający średnią arytmetyczną i średnią geometryczną dwóch liczb całkowitych.

Zadanie 3.5

Przeliczanie wielkości fizycznych

(*) Napisz program przeliczający wielkości fizyczne:

a) z koni mechanicznych [KM] na waty [W] (1KM = 735W]);
b) z kilometrów na godzinę [km/h] na metry na sekundę [m/s];
c) ze stopni Fahrenheita [°F] na stopnie Celsjusza [°C] (t°C=5/9(t°F-32).

Zadanie 3.6

Wiek w przyszłości

(*) Napisz program, który na podstawie twojego obecnego wieku (w latach i miesiącach) i podanego okresu czasu (w miesiącach) obliczy twój wiek w przyszłości (w latach i miesiącach).

Zadanie 3.7

Wyrażenia

(*) Napisz program obliczający wartość wyrażeń (wykorzystaj funkcje standardowe):

$$\left(1+x^2\right)+\sqrt{1+x^2}$$

$$\left|a+bx\right|+x^{100}+\sqrt[3]{1+x}$$

$$\sin 8+\cos^2 x^3+tgx$$

$$\frac{x-y}{2a}+\frac{2a}{x+y}+3\frac{1}{5}x$$

Zadanie 3.8

Obliczanie wartości funkcji

(**) Napisz program, który obliczy wartość wyrażeń (wykorzystaj funkcje standardowe):

a) $10\cos x - 0{,}1x^2 + \sin x + \sqrt{4x^2 + 7}$

b) $\lg(x+5) + e^{x+1} - |tgx + 1|$

c) $\dfrac{\sin^2 \alpha + 0{,}5}{\cos\alpha^4 + tg^4\alpha^2}$

d) $\sqrt{\dfrac{|5\sin\beta^5 + 1|}{3{,}5(\sin\beta + \cos\beta)^2}}$

Zadanie 3.9

Wyrażenia – funkcje trygonometryczne

(**) Napisz program obliczający wartość funkcji trygonometrycznych sinα, cosα i tg α dla α podanego w stopniach np. 90^0, 120^0, 180^0.

Zadanie 3.10

Zamiana zmiennych i liczba odwrotna

(**) Zmienne a i b to dwie liczby całkowite trzycyfrowe. Napisz program zamieniający miejscami wartości tych zmiennych. Po zamianie wyświetl te liczby, a następnie zmodyfikuj je zamieniając miejscami cyfrę jedności i cyfrę setek – wyświetl liczby w odwrotnej kolejności cyfr.

Zadanie 3.11

Obwód okręgu

(*) Napisz program, który obliczy obwód okręgu, który przechodzi przez punkt A(x1, y1) i którego środek znajduje się w punkcie B(x2, y2).

Zadanie 3.12

Pole trójkąta

(**) Napisz program obliczający pole trójkąta, mając podane współrzędne 3 wierzchołków trójkąta w przestrzeni 2D.

Zadanie 3.13

Znaki

(**) Napisz program, który:

a) po podaniu dowolnego znaku wyświetli go wraz z kodem ASCII, a następnie wyświetli znak o kodzie następnym,
b) po podaniu małej litery zamieni ją na dużą.

4. Proste programy strukturalne. Funkcje standardowe i funkcje własne

Cel rozdziału

Zaznajomienie z ideą powtórnego wykorzystania kodu poprzez zastosowanie podprogramów (funkcji) realizujących proste algorytmy sekwencyjne. Zapoznanie z techniką programowania strukturalnego, w tym proceduralnego. Nabycie praktycznych umiejętności pisania prostych programów strukturalnych wykorzystujących funkcje własne i standardowe.

Podstawy teoretyczne

Programowanie strukturalne wykorzystuje ideę powtórnego wykorzystania kodu, czyli dzielenia programu na mniejsze części – bloki. Najmniejszym blokiem jest funkcja, umieszczona w pliku z funkcją główną main, która może być wielokrotnie wykorzystana w tym samym programie. Większym blokiem do wielokrotnego wykorzystania w wielu programach jest moduł – plik źródłowy lub nagłówkowy dołączony do programu. Programowanie strukturalne wykorzystuje hierarchicznie ułożone bloki wykorzystujące instrukcje warunkowe, wyboru i iteracyjne, nie korzysta z instrukcji skoku.

Programowanie proceduralne zakłada dzielenie kodu na procedury (w języku C to funkcje), czyli fragmenty kodu wykonujące określone operacje i opisane nazwą z ewentualnymi parametrami do pobierania i przekazywania danych.

Funkcja – wydzielony fragment kodu spełniający określone czynności. Może zwracać wartość określonego typu (instrukcja **`return`**) lub nie (typ **`void`**).

Prototyp funkcji – zapowiedź funkcji

```
typ_wyniku nazwa_funkcji();
typ_wyniku nazwa_funkcji(parametry_formalne);
```

Przykład 4.1

Prototypy funkcji

```
void witaj();//prototyp funkcji bez parametrów
             // nie zwracającej wartości
int suma(int a, int b); //prototyp funkcji
         //z 2 parametrami typu całkowitego
         //zwracającej wartość typu całkowitego
```

Definicja funkcji – opis algorytmu funkcji

```
typ_wyniku nazwa_funkcji()
{//deklaracje i instrukcje;
   return wyrażenie; //opcjonalnie
}
typ_wyniku nazwa_funkcji(parametry_formalne)
{//deklaracje i instrukcje;
   return wyrażenie; //opcjonalnie
}
```

Przykład 4.2

Definicje funkcji

```
void witaj()
{
  printf("Witaj w swiecie programowania\n");
}

int suma(int a, int b)
{
  return a+b;
}
```

Wywołanie funkcji

```
nazwa_funkcji();
nazwa_funkcji(parametry_aktualne);
```

Przykład 4.3

Wywołania funkcji wewnątrz funkcji main( )

```
witaj();
int liczba1=10; int liczba2=20;
printf("suma=%d \n",suma(liczba1, liczba2));
//wywołanie funkcji z 2 parametrami aktualnymi
//- wyświetlenie wartości funkcji
```

W programie można wykorzystywać funkcje własne i funkcje standardowe, zgrupowane w odpowiednich modułach – dyrektywą **#include** należy dołączyć wybrany plik nagłówkowy.

Wykorzystanie funkcji standardowych:

```
#include <plik_naglowkowy_z_funkcjami>

nazwa_funkcji();
nazwa_funkcji(parametry_aktualne);
```

Grupy wybranych funkcji standardowych:

- Funkcje matematyczne ***<math.h>***
  `asin(x), acos(x), atan(x), sin(x), cos(x), tan(x), exp(x), log(x), pow(x,y), sqrt(x), ceil(x), floor(x), fabs(x), fmod(x,y), ...`
- Funkcje łańcuchowe ***<string.h>***
  `strlen(), strcat(), strcmp(), strcpy(), ...`
- Funkcje znakowe ***<ctype.h>***
  `tolower(), toupper(), isalpha(), isdigit(), isalnum(), ...`
- Funkcje ogólnego użytku ***<stdlib.h>***
  `abs(), rand(), qsort(), ...`

Przykład 4.4

```
#include <math.h>

printf("wynik=%0.2f\n",
sqrt(2)*(sin(0.5)+cos(0.25)));
```

Struktura programu w C – programowanie proceduralne

Program w języku C, wykorzystujący funkcje własne, posiada następującą strukturę:

```
//dyrektywy preprocesora
//protytypy funkcji własnych
//============================
int main() //naglowek funkcji głównej
{
//deklaracje
//instrukcje korzystające z ww. funkcji
return 0;  //instrukcja powrotu
}
//============================
//definicje funkcji własnych
```

Przykład 4.5

```
#include <stdio.h>

void napis();
int wynik(int licz);
//======================
int main(void)
{ int liczba;
  napis();
  printf("Podaj liczbe ");
  scanf("%d", &liczba);
 printf("Wynik to %d\n",wynik(liczba));
  system("PAUSE");
  return 0;
}
//======================
void napis()
{
  printf("Prosty program ");
  printf("w C\n");
 }
int wynik(int licz)
{
    return 2*licz;
}
```

prototypy funkcji

wywołanie funkcji

wywołanie funkcji z parametrem

definicja funkcji

definicja funkcji

WYKONANIE PROGRAMU

```
Prosty program w C
Podaj liczbe 5
Wynik to 10
Press any key to continue . . .
```

Arkusz przykładowych ćwiczeń

Ćwiczenie 4.1

Struktura programu wykorzystującego funkcje – programowanie strukturalne (proceduralne)

Napisz program wyświetlający informacje o autorze programu, obliczający pole koła i wypłatę dla 2. pracowników z wykorzystaniem funkcji – realizacja zadania 3.1, wersja II po modyfikacji, techniką strukturalną.

Wynik:

Program może wyglądać w postaci przedstawionej na listingu 4.1:

Listing 4.1. Prosty program strukturalny

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h> //M_PI
4
5  #define AUTOR "Jan Kowalski"
6  #define KIERUNEK "Informatyka"
7
8  //prototypy funkcji
9  void szlaczek();
10 void info();
11 float pole(float promien);
12 float wyplata(int godziny, float stawka);
13 int main(int argc, char *argv[])
14 { //deklaracje
15  float r,st;
16  int lg;
17  //instrukcje
18  szlaczek(); //wywołanie funkcji
19  info();
20  szlaczek();
21  printf("Podaj promien kola "); scanf("%f",&r);
22  printf("Pole kola o promieniu %0.2f =
23         %0.2f\n",r, pole(r));//wywolanie funkcji
24  szlaczek();
25  printf("Pracownik1\n");
26  printf("Podaj liczbe godzin ");
scanf("%d",&lg);
27  printf("Podaj stawke "); scanf("%f",&st);
28  printf("Wyplata = %0.2f\n", wyplata(lg,st));
29
30  printf("Pracownik2\n");
31  printf("Podaj liczbe godzin ");
scanf("%d",&lg);
32  printf("Podaj stawke "); scanf("%f",&st);
33  printf("Wyplata = %0.2f\n",wyplata(lg,st));
```

prototypy funkcji

wywołanie funkcji

wywołanie funkcji

wywołanie funkcji

```
34
35  szlaczek();
36  system("PAUSE");
37  return 0;
38 }
```

definicje funkcji

```
39
40 void szlaczek()
41 {
42  printf("========================\n");
43  printf("************************\n");
44  printf("========================\n");
45 }

46 void info()
47 {  const int ROK=1;
48    printf("Programowanie liniowe\n");
49    printf("Autor programu: %s kierunek: %s rok:
50           %d\n", AUTOR, KIERUNEK, ROK);
51 }

52 float pole(float promien)
53 {
54  return M_PI*pow(promien,2);
55 }

56 float wyplata(int godziny, float stawka)
57 {  const float PREMIA=0.20;
58   return godziny*stawka+ godziny*stawka*PREMIA;
59 }
```

Komentarz do programu:

Wiersze:

- 1-3: dyrektywa kompilatora dołączająca pliki nagłówkowe `stdio.h, stdlib.h, math.h` (ze stałą `M_PI` -> π)
- 5,6: definicje stałych
- 9-12: prototypy funkcji
- 13: nagłówek funkcji głównej `main`
- 15,16: deklaracje zmiennych
- 18,20,24,35: wywołanie bezparametrowej funkcji `szlaczek` – napisy
- 19: wywołanie bezparametrowej funkcji `info` – napisy

21: wczytanie wartości promienia koła
23: wyświetlenie wartości funkcji `pole` z parametrem
28,33: wyświetlenie wartości funkcji `wyplata` z dwoma parametrami
36: wywołanie funkcji zatrzymującej ekran konsoli
37: zwrócenie wartości 0 przez funkcję główną main
38: koniec funkcji głównej `main`
40-51: definicje funkcji bez parametrów nie zwracających wartości
52-59: definicje funkcji z parametrami zwracających wartości
54: wywołanie funkcji standardowej `pow`() i stałej M_PI z modułu `math.h`

Ćwiczenie 4.2

Struktura programu wykorzystującego funkcje – programowanie strukturalne

Napisz program, który dla 2. liczb całkowitych obliczy i wyświetli ich sumę, różnicę, iloczyn i iloraz z wykorzystaniem funkcji – realizacja zadania 3.2 techniką strukturalną.

Wynik:

Program może wyglądać w postaci przedstawionej na listingu 4.2:

Listing 4.2 **Prosty program strukturalny**

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  //prototypy funkcji
4  int suma(int a, int b);
5  int roznica(int a, int b);
6  int iloczyn(int a, int b);
7  float iloraz(int a, int b);
8  void iloraz2(int a, int b);
9
10 int main(int argc, char *argv[])
11 {int x,y,reszta;
12 printf("Podaj 2 liczby calkowite\n");
13 scanf("%d %d", &x, &y);
14 printf("suma: %d \n",suma(x,y));
15 printf("roznica: %d \n",roznica(x,y));
16 printf("iloczyn: %d \n",iloczyn(x,y));
17 printf("iloraz - wynik dzielenia rzeczywistego:
18         %0.2f\n",iloraz(x,y));
19 iloraz2(x,y);
20 system("PAUSE");
```

prototypy funkcji

wywołanie funkcji

wywołanie funkcji

```
21 return 0;
22 }

23 //definicje funkcji
24 int suma(int a, int b)
25 { return a+b; }
26 int roznica(int a, int b)
27 {  return  a-b;}
28 int iloczyn(int a, int b)
29 {  return  a*b;}
30 float iloraz(int a, int b)
31 {  return  (float)a/b;  }
32 void iloraz2(int a, int b)
33 {   printf("iloraz - wynik dzielenia
                 calkowitego:
34               %d reszta: %d\n",a/b, a%b);
35 }
```

definicja funkcji (wiersz 24)

definicja funkcji (wiersz 26)

definicja funkcji (wiersz 30)

definicja funkcji (wiersz 32)

Komentarz do programu:

Wiersze:

- 1,2: dyrektywa kompilatora dołączająca pliki nagłówkowe `stdio.h, stdlib.h`
- 4-8: prototypy funkcji z parametrami formalnymi `a,b`
- 10: nagłówek funkcji głównej `main`
- 11: deklaracje zmiennych
- 12: wyświetlenie tekstu z przejściem do nowego wiersza
- 13: wczytanie wartości dwóch zmiennych z klawiatury
- 14-18: wyświetlenie wartości funkcji z parametrami aktualnymi `x,y`
- 19: wywołanie funkcji nie zwracającej wartości
- 20: wywołanie funkcji zatrzymującej ekran konsoli
- 21: zwrócenie wartości 0 przez funkcję główną `main`
- 22: koniec funkcji głównej `main`
- 24-35: definicje funkcji

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 4.1

Dane osobowe i adresowe

(*) Napisz i wywołaj funkcję, która na podstawie wprowadzonych z klawiatury danych i zdefiniowanych stałych (*status* przyjmujący wartość *student* i *średnia ocen* przyjmująca wartość twojej pożądanej średniej – liczba rzeczywista) wyświetli w jednym wierszu imię, nazwisko, pesel i płeć, a w drugim dane adresowe: kod, miasto, ulica, nr domu, nr mieszkania, a w trzecim twój status – realizacja zadania 3.1 techniką strukturalną.

Zadanie 4.2

Obliczanie pola figur geometrycznych

(*) Napisz program obliczający pole powierzchni i obwód figur: kwadratu, prostokąta, trapezu, trójkąta z wykorzystaniem funkcji – realizacja zadania 3.2 techniką strukturalną.

Zadanie 4.3

Obliczanie objętości i pola powierzchni brył

(*) Napisz program obliczający objętość i pole powierzchni sześcianu, prostopadłościanu o podstawie kwadratowej i prostokątnej oraz walca z wykorzystaniem funkcji – realizacja zadania 3.3 techniką strukturalną.

Zadanie 4.4

Obliczanie średniej arytmetycznej i średniej geometrycznej dwóch liczb całkowitych

(*) Napisz program obliczający średnią arytmetyczną i średnią geometryczną dwóch liczb całkowitych z wykorzystaniem funkcji – realizacja zadania 3.4 techniką strukturalną.

Zadanie 4.5

Przeliczanie wielkości fizycznych

(*) Napisz program przeliczający wielkości fizyczne z wykorzystaniem funkcji – realizacja zadania 3.5 techniką strukturalną:

a) z koni mechanicznych [KM] na waty [W];
(1KM = 735W]);
b) z kilometrów na godzinę [km/h] na metry na sekundę [m/s];
c) ze stopni Fahrenheita [°F] na stopnie Celsjusza [°C];
(t°C=5/9(t°F-32).

Zadanie 4.6

Wiek w przyszłości

(*) Napisz i wywołaj funkcję, która na podstawie twojego obecnego wieku (w latach i miesiącach) i podanego okresu czasu (w miesiącach) obliczy twój wiek w przyszłości (w latach i miesiącach) – realizacja zadania 3.6 techniką strukturalną.

Zadanie 4.7

Wyrażenia

(**) Napisz i wywołaj funkcje obliczające wartość wyrażeń (wykorzystaj funkcje standardowe) – realizacja zadania 3.7 techniką strukturalną:

$$\left(1+x^2\right)+\sqrt{1+x^2}$$

$$\left|a+bx\right|+x^{100}+\sqrt[3]{1+x}$$

$$\sin 8+\cos^2 x^3+tgx$$

$$\frac{x-y}{2a}+\frac{2a}{x+y}+3\frac{1}{5}x$$

Zadanie 4.8

Obliczanie wartości funkcji

(**) Napisz i wywołaj funkcje, które obliczą wartość wyrażeń (wykorzystaj funkcje standardowe) – realizacja zadania 3.8 techniką strukturalną:

e) $10\cos x - 0{,}1x^2 + \sin x + \sqrt{4x^2 + 7}$

f) $\lg(x+5) + e^{x+1} - |tgx + 1|$

g) $\dfrac{\sin^2 \alpha + 0{,}5}{\cos\alpha^4 + tg^4\alpha^2}$

h) $\sqrt{\dfrac{|5\sin\beta^5 + 1|}{3{,}5(\sin\beta + \cos\beta)^2}}$

Zadanie 4.9

Wyrażenia – funkcje trygonometryczne

(**) Napisz i wywołaj funkcję obliczającą wartość funkcji trygonometrycznych $\sin\alpha$, $\cos\alpha$ i $\mathrm{tg}\,\alpha$ dla α podanego w stopniach np.: 90^0, 120^0, 180^0 – realizacja zadania 3.9 techniką strukturalną.

Zadanie 4.10

Liczba odwrotna

(**) Napisz i wywołaj funkcję, która dla podanej liczby całkowitej trzycyfrowej zwróci liczbę odwrotną – zamieni miejscami cyfrę jedności i cyfrę setek.

Zadanie 4.11.

Obwód okręgu

(*) Napisz i wywołaj funkcję, która obliczy obwód okręgu, który przechodzi przez punkt A(x1, y1) i którego środek znajduje się w punkcie B(x2, y2) – realizacja zadania 3.11 techniką strukturalną.

Zadanie 4.12

Pole trójkąta

(**) Napisz i wywołaj funkcję obliczającą pole trójkąta, mając podane współrzędne 3 wierzchołków trójkąta w przestrzeni 2D – realizacja zadania 3.12 techniką strukturalną.

Zadanie 4.13

Znaki

(**) Napisz i wywołaj funkcje (realizacja zadania 3.13 techniką strukturalną), które:

a) po podaniu dowolnego znaku wyświetlą go wraz z kodem ASCII, a następnie wyświetli znak o kodzie następnym,
b) po podaniu małej litery zamieni ją na dużą.

Zadanie 4.14

Informacja o autorze programu – programowanie strukturalne

(*) Napisz i wywołaj funkcję, która wyświetli dane o autorze programu.

a) wersja 1: dwa pliki źródłowe:
   Prototyp funkcji umieść w pliku źródłowym z funkcją `main()`, definicję funkcji w pliku źródłowym `funkcje.c`,
b) wersja 2: plik źródłowy i plik nagłówkowy:
   Prototyp funkcji umieść w pliku źródłowym z funkcją `main()`, definicję w pliku nagłówkowym `funkcje.h`.

5. Instrukcje warunkowe IF, IF…ELSE. Operator warunkowy

Cel rozdziału

Zaznajomienie z realizacją algorytmów z rozgałęzieniami z wykorzystaniem instrukcji warunkowej oraz z wykorzystaniem operatora ternarnego (warunkowego). Nabycie praktycznych umiejętności programowania algorytmów z rozgałęzieniami.

Podstawy teoretyczne

Instrukcje warunkowe służą do sterowania przebiegiem programu w zależności od spełnienia lub nie spełnienia określonego warunku. Umożliwiają programową realizację algorytmów z rozgałęzieniami.

Instrukcje warunkowe mogą występować w następujących postaciach:

```
if (wyrażenie) instrukcja;
```

Jeżeli wyrażenie jest prawdziwe (≠0) wykonuj ***instrukcję.***

Przykład 5.1

```
if(a>b) max=a;
```

```
if (wyrażenie) instrukcja1;
          else instrukcja2;
```

Jeżeli wyrażenie jest prawdziwe (wynik≠0) wykonuj ***instrukcję1****,*
w przeciwnym wypadku wykonuj ***instrukcję2.***

Przykład 5.2

```
if(a>b) max=a;
   else max=b;
```

Jeżeli w zależności od wartości wyrażenia ma być wykonywanych wiele instrukcji należy użyć instrukcji złożonej (grupującej) {}:

```
if (wyrażenie) {instrukcja1;
                instrukcja2;
               }      /*instrukcja złożona
                              (grupująca)*/
else {instrukcja3;
                instrukcja4;
               }      /*instrukcja złożona
                              (grupująca)*/
```

*Jeżeli wyrażenie jest prawdziwe (wynik≠0) wykonuj **blok instrukcji** (np. instrukcja1 instrukcja2), w przeciwnym wypadku wykonuj **blok instrukcji** (np. instrukcja3, instrukcja 4).*

Przykład 5.3

```
if(a>b){max=a; printf("Max z 2 liczb
=%d\n",max);}
   else{max=b; printf("Max z 2 liczb
=%d\n",max);}
```

Instrukcje warunkowe mogą być zagnieżdżone (jedna w drugiej), przy czym obowiązuje zasada, że `else` związane jest z najbliższym `if`:

Przykład 5.4

```
if(a>b) printf("Liczba a jest większa\n");
   else if(a<b) printf("Liczba b jest
większa\n");
           else printf("Liczby a i b są równe\n);
```

Wyrażenie po słowie `if` może zawierać następujące operatory porównania:

==	równe,
!=	różne,
<	mniejsze,
>	większe,
<=	mniejsze lub równe,
>=	większe lub równe.

Wyrażenie może również zawierać operatory logiczne:

`&&` logiczne AND – iloczyn,
`||` logiczne OR – alternatywa,
`!` logiczne NOT – negacja.

Przykład 5.5

```
if(a==b) printf("Liczby sa rowne\n");
if(a>=2 && a<=5) printf("Liczba jest z
przedziału od 2 do 5\n");
```

Alternatywą dla instrukcji warunkowej jest wykorzystanie operatora warunkowego. Operator ternarny (trójoperandowy operator warunkowy) wykorzystuje trzy operandy, z których każdy jest wyrażeniem:

```
wyrażenie1 ? wyrażenie2 : wyrażenie3
```

Jeżeli wyrażenie1 jest prawdziwe *(≠0) to całe wyrażenie przyjmuje wartość wyrażenia2, w przeciwnym wypadku wyrażenia3.*

Przykład 5.6

```
max=(a>b)? a: b;
```

Arkusz przykładowych ćwiczeń

Ćwiczenie 5.1.

Obliczenie wartości funkcji

Dla danych wartości zmiennych x, y napisz funkcję obliczającą wartość:

$$f(x,y)=\begin{cases} x^2+y^2 & x<0 \; i \; y<0 \\ 0 & xy \le 0 \\ \sqrt{x+y} & x>0 \; i \; y>0 \end{cases}$$

Wynik:

- funkcja może być zdefiniowana w następujący sposób:

definicja funkcji

```
float f(float x,float y)
{ if(x<0 && y<0) return (x*x + y*y);
        else  if(x*y <= 0) return 0;
              else return sqrt(x+y);
}
```

Użycie funkcji przedstawiono na listingu 5.1.

Ćwiczenie 5.2

Sprawdzanie położenia punktu w układzie współrzędnych

Napisz funkcję sprawdzającą położenie punktu A(x,y) w układzie współrzędnych (I, II, III czy IV ćwiartka).

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

definicja funkcji

```
void punkt(float x, float y)
{
 if (x*y ==0)  printf("punkt na osi\n");
   else  if(x>0)
      f (y>0) printf("punkt w I cw.\n");
              else printf("punkt w IV cw.\n");
   else  if(y>0) printf("punkt w II cw.\n");
              else printf("punkt w III cw.\n");
 }
```

Użycie funkcji przedstawiono na listingu 5.1.

Ćwiczenie 5.3

Parzystość liczby

Napisz funkcję sprawdzającą, czy podana liczba całkowita jest parzysta czy nieparzysta.

Wynik:

Funkcja może być zdefiniowana w następujący sposób z wykorzystaniem operatora warunkowego lub funkcji warunkowej:

definicja funkcji

```
char parzysta(int a)
{ return (a%2==0)?'T':'N';
// if(a%2==0) return 'T'; else return 'N';
}
```

Użycie funkcji przedstawiono na listingu 5.1.

Listing 5.1 **Funkcje warunkowe i operator warunkowy**

prototypy funkcji

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  float f(float x,float y);
5  void punkt(float x, float y);
6  char parzysta(int a);
7
```

wywołanie funkcji

wywołanie funkcji

definicja funkcji f( )

definicja funkcji

definicja funkcji

```
 8  int main(int argc, char *argv[])
 9  {float xx,yy; int liczba;
10   printf("Podaj x=");
11   scanf("%f",&xx);
12   printf("Podaj y=");
13   scanf("%f",&yy);
14   printf("wynik funkcji=%0.2f\n", f(xx,yy));
15   punkt(xx,yy);
16   printf("Podaj liczbe calkowita");
17   scanf("%d",&liczba);
18   printf("Liczba %d parzysta: %c\n",liczba,
                 parzysta(liczba));
19   //wywołanie funkcji parzysta
20   system("PAUSE");
21   return 0;
22  }
23
24  float f(float x,float y)      //definicja funkcji f
25  {if(x<0 && y<0) return (x*x + y*y);
26        else  if(x*y <= 0) return 0;
27                else return sqrt(x+y);
28  }
29
30  void punkt(float x, float y) //definicja funkcji punkt
31  {if (x*y ==0)  printf("punkt na osi\n");
32    else  if(x>0)
33                  if (y>0) printf("punkt w I cw.\n");
34                  else printf("punkt w IV cw.\n");
35     else  if(y>0) printf("punkt w II cw.\n");
36                  else printf("punkt w III cw.\n");
37  }
38
39  char parzysta(int a)      //definicja funkcji parzysta
40  { return (a%2==0)?'T':'N';
41  // if(a%2==0) return 'T'; else return 'N';
42  }
```

Komentarz do programu:

Wiersze:

- 1,2: dyrektywa kompilatora dołączająca pliki nagłówkowe `stdio.h, stdlib.h`
- 4-6: deklaracje – prototypów funkcji: `f, punkt, parzysta`
- 8: nagłówek funkcji głównej `main`
- 9: deklaracje zmiennych: xx, yy, liczba
- 10-13, 16-17: wczytywanie z klawiatury wartości zmiennych
- 14: wyświetlenie wartości funkcji `f` zwracającej wartość

15: wywołanie funkcji `punkt`, nie zwracającej wartości
18: wyświetlenie wartości funkcji `parzysta` zwracającej wartość
20: wywołanie funkcji `system` – zatrzymanie ekranu
24-42: definicje zapowiedzianych wcześniej funkcji:
`f, punkt, parzysta`

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 5.1

Obliczanie wartości funkcji

(*) Napisz funkcję obliczającą wartość z. Wywołaj tą funkcję.

$$z = \begin{cases} 1-\sin\alpha & t = 8 \\ \frac{1}{2}(1+\cos\alpha) & t = 0,1,2,3 \\ \sqrt{\alpha^2+1} & t = 4,6,7 \end{cases}$$

Wskazówka:

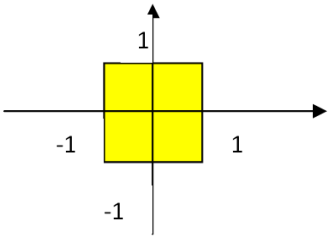
z** jest funkcją zmiennej α obliczanej wg różnych wzorów w zależności od parametru **t.

Zadanie 5.2

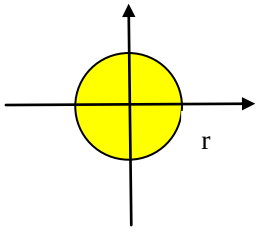
Przynależność punktu do wskazanego obszaru

(*) Napisz funkcję sprawdzającą, czy punkt o współrzędnych x,y należy do zamalowanego obszaru. Wywołaj tę funkcję.

a)



b)



Zadanie 5.3

Równanie kwadratowe

(**) Napisz funkcję obliczającą pierwiastki równania kwadratowego $ax^2+bx+c=0$ uwzględniającą wszelkie możliwe warianty danych a, b, c. Wywołaj tę funkcję.

Wskazówki:

skorzystaj ze schematu ze strony 15 lub 18,
równanie kwadratowe istnieje gdy a jest różne od zera,
równanie kwadratowe posiada pierwiastki gdy delta jest nieujemna,
równanie kwadratowe posiada 2 pierwiastki gdy delta jest większa od 0,
równanie kwadratowe posiada 1 pierwiastek, gdy delta = 0.

Zadanie 5.4

Szczęśliwy bilet

(*) Bilet tramwajowy posiada sześciocyfrowy numer. Napisz funkcję sprawdzającą, czy jest to bilet „szczęśliwy". Bilet uznawany jest za „szczęśliwy", jeżeli suma 3 pierwszych i 3 ostatnich cyfr jest taka sama. Wywołaj tę funkcję.

Zadanie 5.5

Wypłata pracownika

(*) Napisz funkcję, która na podstawie pensji i stażu pracownika obliczy jego wypłatę w następujący sposób: jeżeli staż pracownika jest mniejszy niż 5 lat, dodatek stażowy się nie należy, jeżeli pracownik przepracował w firmie od 5 do 10 lat, dodatek stażowy wynosi tyle procent ile lat ma staż pracownika, jeżeli staż pracownika jest powyżej 10 lat, dodatek stażowy wynosi 15%. Wywołaj tę funkcję.

Zadanie 5.6

Stypendium studenta

(*) Napisz funkcję, która na podstawie 3 ocen z egzaminów wyświetli informację o przyznanym (lub nie) stypendium. Student otrzyma stypendium, jeśli zdał wszystkie egzaminy. Jeśli średnia ocen jest większa od 4 otrzyma stypendium 500 zł; jeśli 3 < średnia <= 4 to stypendium wynosi 300 zł. Wywołaj tą funkcję.

Zadanie 5.7

Pole trójkąta

(*) Napisz i wywołaj funkcję, która na podstawie 3 liczb – boków trójkąta obliczy jego pole. Zweryfikuj czy podane liczby utworzą trójkąt (wersja zadania 4.12 z weryfikacją poprawności danych).

Zadanie 5.8

Liczby

(*) Dane są trzy dodatnie liczby całkowite a, b, c. Jeżeli wszystkie są parzyste oblicz ich sumę, jeżeli dowolna z nich to 1 oblicz ich iloczyn, w pozostałych przypadkach zwróć -1. Zdefiniuj i wywołaj odpowiednią funkcję.

Zadanie 5.9

Znaki

(*) Napisz i wywołaj funkcję, która sprawdzi czy:

a) podany z klawiatury znak jest literą angielskiego alfabetu,
b) kod ASCII podanego znaku jest dwucyfrowy,
c) podany znak jest znakiem działania arytmetycznego.

Zadanie 5.10

Podatek

(**) Napisz i wywołaj funkcję obliczającą podatek wg zasad skali podatkowej podanych w tabeli:

Skala podatkowa od roku 2009 do 2015 r.

Podstawa obliczenia podatku w zł		***Podatek wynosi***
ponad	***do***	
	85528	*18% minus kwota zmniejszająca podatek: 556,02 zł*
85528		*14839,02 zł + 32% nadwyżki ponad 85528 zł*

6. Instrukcja wyboru SWITCH. Instrukcja BREAK

Cel rozdziału

Zaznajomienie z realizacją algorytmów z rozgałęzieniami z wykorzystaniem instrukcji wyboru SWITCH oraz instrukcji BREAK. Nabycie praktycznych umiejętności stosowania instrukcji SWITCH, BREAK i programowania algorytmów z rozgałęzieniami.

Podstawy teoretyczne

Instrukcja wyboru może występować w następujących postaciach:

wybór jednokrotny

a) warianty wyboru zakończone instrukcją BREAK

```
switch (wyrażenie typu int, char lub enum)
{
      case stała1: ciąg_instrukcji_1; break;
      case stała2: ciąg_instrukcji_2; break;
      ...
      default://opcjonalnie
            ciąg_instrukcji_D;
}
```

Jeżeli wyrażenie przyjmuje wartość stałej 1 wykonuj ***ciąg_instrukcji_1*** *i wyjdź z instrukcji switch*
Jeżeli wyrażenie przyjmuje wartość stałej 2 wykonuj ***ciąg_instrukcji_2*** *i wyjdź z instrukcji switch*
...
w przeciwnym wypadku wykonuj ***ciąg_instrukcji_D****.*

Przykład 6.1

```
switch (ocena) {
   case 2:  printf("ndst");break;
   case 3:  printf("dst"); break;
   case 4:  printf("db"); break;
```

```
   case 5:  printf("bdb");break;
 default:  printf("zla ocena");
 }
```

wybór wielokrotny

b) instrukcja BREAK kończy kilka wariantów wyboru

```
switch (wyrażenie typu int, char lub enum)
{
      case stała1:
      case stała2: ciąg_instrukcji_2; break;
      ...
      default://opcjonalnie
            ciąg_instrukcji_D;
}
```

Jeżeli wyrażenie przyjmuje wartość stałej 1 lub stałej 2 wykonuj ***ciąg_instrukcji_2*** *i wyjdź z instrukcji switch*
...
w przeciwnym wypadku wykonuj ***ciąg_instrukcji_D***.

Przykład 6.2

```
switch (ocena) {
   case 2:  printf("negatywna ocena");break;
   case 3:
   case 4:
   case 5:  printf("pozytywna ocena");break;
 default: printf("zla ocena");
 }
```

Zwiększenie czytelności programu można uzyskać stosując typy wyliczeniowe w instrukcjach sterujących – zbiór czytelnych nazw o wartościach stałych całkowitych

Przykład 6.3

definiowanie typu enum – nazwa kolor

```
enum kolory {czerwony, pomaranczowy, zolty,
zielony, niebieski, blekitny, fioletowy};
//typ wyliczeniowy o wartościach: 0, 1, 2, 3,
4, //5, 6
```

definiowanie egzemplarza 'kol'

```
enum kolory kol; //zmienna wyliczeniowego typu
```

Arkusz przykładowych ćwiczeń

Ćwiczenie 6.1

Prosty kalkulator

Dla danych wartości zmiennych rzeczywistych x, y i znaku działania (+, -, *, /) napisz funkcję obliczającą wartość wybranej operacji arytmetycznej:

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
void kalkulator(double x, double y, char znak)
{
 switch (znak) {
   case '+': printf("suma= %0.2lf\n",x+y);
                    break;
   case '-': printf("roznica= %0.2lf\n",
                    x-y);break;
   case '*': printf("iloczyn= %0.2lf\n",
                    x*y);break;
   case '/': if (y) {printf("iloraz= %0.2lf\n",
                    x/y);}
   else printf("y=0\n");break;
 default: printf("zły znak\n\n");
 } //koniec switch
}
```

Użycie funkcji przedstawiono na listingu 6.1.

Ćwiczenie 6.2

Sprawdzanie czy litera jest samogłoską

Napisz funkcję sprawdzającą czy podana litera jest samogłoską.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
void samogloska(char litera)
{ //nie uwzgledniono polskich liter ą, ę, ó
switch(litera)
 { case 'a':
   case 'A':
```

```
    case 'e':
    case 'E':
    case 'i':
    case 'I':
    case 'o':
    case 'O':
    case 'u':
    case 'U':
    case 'y':
    case 'Y': printf("samogloska\n"); break;
    default: printf("nie samogloska\n");
  }
}
```

Użycie funkcji przedstawiono na listingu 6.1.

Ćwiczenie 6.3

Kolory tęczy

Napisz funkcję sprawdzającą, nazwę i odcień podanego koloru w tęczy z wykorzystaniem typu wyliczeniowego `kolory`:

```
enum kolory {czerwony, pomaranczowy, zolty,
zielony, niebieski, blekitny, fioletowy};
```

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

argumentem funkcji jest egzeplarz 'kol' typu enum 'kolory'

```
void tecza(enum kolory kol)
{
switch (kol)
{
case czerwony: printf("czerwony ");break;
case pomaranczowy:  printf("pomaranczowy
");break;
case zolty: printf("zolty "); break;
case zielony:  printf("zielony ");break;
case niebieski:  printf("niebieski ");break;
case blekitny:  printf("blekitny ");break;
case fioletowy: printf("fioletowy"); break;
  default: printf("brak takiego koloru\n");
}
switch (kol)
{
case czerwony:
case pomaranczowy:
```

```
case zolty: printf(" - wybrano cieply kolor\n");
break;
case zielony:
case niebieski:
case blekitny:
case fioletowy: printf(" - wybrano chlodny
kolor\n"); break;
 }
```

Użycie funkcji przedstawiono na listingu 6.1.

Ćwiczenie 6.4

Menu programu

Program główny skonstruuj na zasadzie menu, z którego użytkownik wybiera pozycję do wykonania.

Wynik:

Rezultat przedstawiono na listingu 6.1.

Listing 6.1 **Instrukcja wyboru**

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  enum kolory{czerwony, pomaranczowy, zolty,
5  zielony, niebieski, blekitny, fioletowy};
6  void kalkulator(double x, double y,char
znak);
7  void samogloska(char litera);
8  void tecza(enum kolory kol);
9  //=====================================
10  int main(int argc, char *argv[])
11  { double a,b;
12  char zn,l;
13  int k, wybor;
14  printf("MENU PROGRAMU:\n");
15  printf("1. Kalkulator\n");
16  printf("2. Samogloska\n");
17  printf("3. Kolory teczy\n");
18  printf("Wybierz pozycje\n");
19  scanf("%d",&wybor);
20  switch(wybor)
21  { case 1:
22     printf("podaj 2 liczby i znak dzialania\n
                                              ");
```

definiowanie enum

prototypy funkcji

funkcja main

instrukcja wyboru

```
23      scanf("%lf %lf %c", &a,&b, &zn);
24      kalkulator(a,b,zn); break;
25    case 2:
26      printf("podaj litere\n");
27      getche(); //scanf("%c", &l);
28      printf("\n");
29      samogloska(l); break;
30    case 3:
31      printf("Wybierz pozycje koloru w teczy od
                0 do 6\n");
32      scanf("%d", &k);
33      tecza(k); break;
34    default:
35      printf("zly wybor\n");
36   }
37   system("PAUSE");
38   return 0;
39 }
40//=====================================
41 void kalkulator(double x, double y,char znak)
42 {
43   switch (znak) {
44     case '+': printf("suma= %0.2lf\n",x+y);
                                          break;
45     case '-': printf("roznica= %0.2lf\n",
                                      x-y);break;
46     case '*': printf("iloczyn=
                             %0.2lf\n",x*y);break;
47     case '/': if (y) {printf("iloraz=
                                    %0.2lf\n",x/y);}
48                    else printf("y=0\n");break;
49  default: printf("zły znak\n\n");
50  } //koniec switch
51 }
52//=====================================
53 void samogloska(char litera)
54 { //nie uwzgledniono polskich liter ą, ę ó
55 switch(litera)
56 { case 'a':
57   case 'A':
58   case 'e':
59   case 'E':
60   case 'i':
61   case 'I':
62   case 'o':
```

definicja funkcji kalkulator

definicja funkcji samogloska

```
63   case 'O':
64   case 'u':
65   case 'U':
66   case 'y':
67   case 'Y': printf("samogloska\n"); break;
68   default: printf("nie samogoska\n");
69 }}
70//=====================================
71 void tecza(enum kolory kol)
72 {
73 switch (kol){
74  case czerwony: printf("czerwony ");break;
75  case pomaranczowy:  printf("pomaranczowy
                                ");break;
76  case zolty: printf("zolty "); break;
77  case zielony:  printf("zielony ");break;
78  case niebieski:  printf("niebieski ");break;
79  case blekitny:  printf("blekitny ");break;
80  case fioletowy: printf("fioletowy"); break;
81  default: printf("brak takiego koloru\n");
82 }
83 switch (kol){
84  case czerwony:
85  case pomaranczowy:
86  case zolty: printf(" - wybrano cieply
                        kolor\n"); break;
87  case zielony:
88  case niebieski:
89  case blekitny:
90  case fioletowy: printf(" - wybrano chlodny
                             kolor\n");
91 break;
92 }
93 }//============================
```

definicja funkcji tecza

Komentarz do programu:

Wiersze:

- 1,2: dyrektywa kompilatora dołączająca pliki nagłówkowe `stdio.h, stdlib.h`
- 4,5: Definicja typu wyliczeniowego `kolory`
- 6-8: deklaracje – prototypów funkcji: `kalkulator, samogloska, tecza`
- 10: nagłówek funkcji głównej `main`

11-13: deklaracje zmiennych: `a, b, zn, l, k, wybor`
18,19: Wybór pozycji menu
20: Instrukcja wyboru realizująca menu
24: wywołanie funkcji `kalkulator`
29: wywołanie funkcji `samogloska`
33: wywołanie funkcji `tecza`
37: wywołanie funkcji `system` – zatrzymanie ekranu
41-93: definicje zapowiedzianych wcześniej funkcji: `kalkulator, samogloska, tecza`

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 6.1

Obliczanie wartości funkcji

(*) Napisz funkcję obliczającą wartość z wykorzystując instrukcję `switch`. Wywołaj tę funkcję.

$$z=\begin{cases}1-\sin\alpha & t=8\\ \frac{1}{2}(1+\cos\alpha) & t=0,1,2,3\\ \sqrt{\alpha^2+1} & t=4,6,7\end{cases}$$

Wskazówka:

***z** jest funkcją zmiennej α obliczanej wg różnych wzorów w zależności od parametru **t**.*

Zadanie 6.2

Obliczanie wartości funkcji

(*) Napisz i wywołaj funkcję obliczającą wartość *y* dla podanych wartości zmiennych:

$$y=\begin{cases}a+2/b+4w & w=4 \quad a=1.5\\ a-xw & w=8 \quad b=2\\ (a+b)^2 & w=2 \quad x=1.5\\ brak\ \ rozwiazan & w\neq 4,8,2\end{cases}$$

Zadanie 6.3

Figury

(*) Napisz funkcje z parametrami obliczające pola figur: kwadratu, prostokąta, koła i trójkąta.
Napisz funkcję, która za pomocą parametru wyświetla informację o kolorze figury.
Z wykorzystaniem ww. funkcji napisz program, który w zależności od wyboru użytkownika (rodzaj figury i jej kolor) i odpowiednich danych wyświetla informację *Wybrałeś: <tu nazwa figury>, kolor: <tu kolor figury>, pole: <tu pole figury>*

Zadanie 6.4

Miesiące

(*) Napisz funkcję, która na podstawie numeru miesiąca określi do jakiego kwartału roku on należy i ile dni zawiera. Wywołaj tą funkcję.

Zadanie 6.5

Plan zadań na bieżący tydzień

(*) Napisz funkcję wyświetlającą plan działań na wybrany dzień tygodnia np. *Wtorek – mycie okien*. Wykorzystaj typ wyliczeniowy `tydzien`. Wywołaj tę funkcję.

Zadanie 6.6

Kalkulacja ceny wyjazdu na wypoczynek

(*) Napisz funkcję obliczającą cenę wyjazdu na wypoczynek uzależnioną od wyboru kilku wariantów: liczba dni pobytu, miejsce pobytu (wybór hotelu), rodzaj pokoju, rodzaj wyżywienia. Wywołaj tę funkcję.

Zadanie 6.7

Ryczałt za używanie samochodu osobowego

(**) Napisz funkcję obliczającą wysokość ryczałtu za używanie samochodu osobowego wg obowiązujących aktualnie zasad. Wywołaj tę funkcję.

Wskazówka:

Maksymalne miesięczne kwoty ryczałtów za używanie samochodów osobowych – okres obowiązywania od 14.11.2015:

Okres / Pojemność silnika	***od 14.11.2015 (zł)***	
	do 900 cm³	***powyżej 900 cm³***
limit 300 km	*156,42*	*250,74*
limit 500 km	*260,70*	*417,90*
limit 700 km	*364,98*	*585,06*

UWAGA!

Limit ustalony w zależności od liczby mieszkańców w danej gminie lub mieście, w których pracownik jest zatrudniony, nie może przekroczyć:

- *300 km – do 100 tys. mieszkańców,*
- *500 km – ponad 100 tys. do 500 tys. mieszkańców,*
- *700 km – ponad 500 tys. mieszkańców.*

Zadanie 6.8

Serwis telefoniczny

(**) Napisz funkcję symulującą system ekspertowy pomocy serwisowej urządzania typu:

Witamy w pomocy serwisowej firmy X.
Wybierz tonowo usługę:

1. *zakup towaru*
   - *1.1. problemy z dostawą =>porada 1*
   - *1.2. problemy niezgodności z zamówieniem =>porada 2*
2. *zwrot towaru =>porada 3*
3. *serwis techniczny zakupionego urządzenia*
   - *3.1. urządzanie nie działa*
     - *3.1.1. sprawdź czy włączone => włącz*
     - *3.1.2. sprawdź bezpiecznik => wymień bezpiecznik na nowy*
   - *3.2. urządzenie działa wadliwie*
     - *3.2.1. usterka A =>porada A*
     - *3.2.2. usterka B =>porada B*
     - *3.2.3. usterka C =>porada C*
4. *zgłoszenie operatora =>odpowiedź operatora*

7. Instrukcje iteracyjne WHILE, DO...WHILE. Instrukcja CONTINUE

Cel rozdziału

Zaznajomienie z realizacją algorytmów wymagających wielokrotnego powtarzania sekwencji instrukcji z wykorzystaniem pętli logicznych. Nabycie praktycznych umiejętności programowania algorytmów iteracyjnych z wykorzystaniem instrukcji `WHILE`, `DO...WHILE`.

Podstawy teoretyczne

UWAGA!

Instrukcje iteracyjne (pętli): **`while`** i **`do while`** służą do powtarzania fragmentu programu dopóki wartość wyrażenia wpisanego w instrukcję pętli jest różna od zera, czyli warość logiczna tego wyrażenia jest prawdą.

W pętli **`do...while`** sprawdzenie wyrażenia następuje po pierwszym wykonaniu instrukcji , a więc taka pętla wykona się co najmniej raz.

```
while (wyrażenie) instrukcja;
do {instrukcja} while (wyrażenie);
```

Jeżeli ma być powtórzonych wiele instrukcji należy je ująć w nawiasy { }, czyli zapisać w postaci instrukcji złożonej:

```
while (wyrażenie)
{instrukcja1;
 instrukcja2;
 ...
}
```

Instrukcja **`continue`** powoduje wykonanie kolejnej iteracji (jeśli wyrażenie sterujące powtórzeniem jest prawdziwe) w bloku **`while`** (lub **`do...while`**) z pominięciem instrukcji następujących po **`continue`**.

Przykład 7.1a

```
I=2;
while (I<10) I=I*I;
```

Przykład 7.1b

```
I=2;
do {I=I*I;} while (I<10);
```

*W obu wersjach przykładu **7.1** wartość I będzie kolejno wynosiła: 2, 4, 16. Następne mnożenie nie będzie wykonane, ponieważ wyrażenie I < 10 dla I = 16 nie jest prawdziwe.*

Przykład 7.2a

```
I=1;
while (I<6)
{ printf("*\n");
  I++;
}
```

Przykład 7.2b

```
I=1;
do
{ printf("*\n");
  I++;
}
while (I<6);
```

W obu wersjach przykładu 7.2 nastąpi wyświetlenie pionowo pięciu gwiazdek.

Przykład 7.3

```
int I=0;
while (I<6)
{ I++;
  if (I==3) continue;
  printf("%d\n",I);
 }
```

*Zostaną wyświetlone następujące liczby: 1 2 4 5 6, ponieważ dla I = 3 nie będzie wykonana funkcja **printf(…)**.*

Arkusz przykładowych ćwiczeń

Ćwiczenie 7.1

Obliczenie sumy

Dany jest zbiór N liczb rzeczywistych. Napisz funkcję wczytującą dane i obliczającą sumę liczb > 100.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 float suma()
2 {   int i, n;
3     float s, x;
4     s=0;
5     printf("Podaj ilosc liczb:");
6     scanf("%d", &n);
7     i=1;
8     while(i<=n)
9     {     printf("Podaj liczbe:");
10          scanf("%f", &x);
11          if(x > 100) s+=x;
12          i++;
13    }
14    return s;
15 }
```

Komentarz do programu:

Wiersze:

- 2,3: deklaracje zmiennych: n – ilość liczb, i – numer liczby, x – liczba podana przez użytkownika
- 4: wartość początkowa sumy
- 6,10: wczytywanie wartości zmiennych
- 7: wartość początkowa zmiennej sterującej pętlą, jednocześnie nr wczytywanej liczby
- 9-13: blok pętli while
- 14: przekazanie wyniku z funkcji

Użycie funkcji przedstawiono na listingu 7.1.

Ćwiczenie 7.2

Obliczenie wartości wyrażenia w zadanym przedziale

Dla danych wartości A, B, K napisz funkcję obliczającą i wyświetlającą wartości wyrażenia $x + \sin x$ dla $x \in$ <A,B>, z krokiem K, to znaczy w punktach: x = A, x = A+K, x = A+2K, …

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 void tablica_wartosci(float A, float B, float K)
2 {     float x;
3       x=A;
4       while(x<=B)
5       {     printf("x=%f, x+sin(x)=%f\n", x, x+sin(x));
6             x+=K;
7       }
8 }
```

Użycie funkcji przedstawiono na listingu 7.1.

Ćwiczenie 7.3

Obliczenie iloczynu

Dane są liczby A, B. Napisz funkcję obliczającą iloczyn liczb z przedziału <A,B> podzielnych przez 3. Jeśli A > B zamień wartości zmiennych.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 int iloczyn(int A, int B)
2 {     int w, il;
3       if (A > B)
4       {     w=A;
5             A=B;
6             B=w;
7       }
8       if ((A+1) % 3 ==0) A++;
9       else if ( (A+2) % 3 ==0) A+=2;
10      w=A;
11      il=1;
12      while( w <= B)
```

```
13      {     il*=w;
14            w+=3;
15      }
16      return il;
17 }
```

Komentarz do programu:

Wiersze:

2: deklaracje zmiennych: w – zmienna pomocnicza wykorzystana do zamiany A, B, oraz do sterowania pętlą, il – iloczyn
3-7: ewentualna zamiana wartości A i B
8-10: wyznaczenie pierwszej liczby z przedziału podzielnej przez 3
11: wartość początkowa iloczynu
12-15: blok pętli while obliczającej iloczyn
14: wyjście z funkcji z wartością iloczynu

Użycie funkcji przedstawiono na listingu 7.1.

Ćwiczenie 7.4

Zgadywanie liczby wylosowanej w programie

W `main()` wylosowana jest liczba całkowita a∈ <0 , 9>. Napisz funkcję realizującą najwyżej pięciokrotne zgadywanie przez użytkownika wylosowanej liczby. Funkcja zwraca liczbę prób do uzyskania sukcesu lub liczbę -1, jeśli użytkownik nie zgadł.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 int zagadka (int a)
2 {  int w, il;
3    il=0;
4    do
5    {    printf(" wpisz liczbę ");
6         scanf("%d", &w);
7         il++;
8    } while (a!= w && il < 6);
9    if (a != w) il=-1;
10   return il;
11 }
```

Użycie funkcji przedstawiono na listingu 7.1.

Listing 7.1

Instrukcje iteracyjne WHILE, DO…WHILE

prototypy funkcji

```
1  float suma();
2  void tablica_wartosci(float A, float B, float K);
3  int iloczyn(int A, int B);
4  int zagadka (int a);
5  int main ( )
6  {  int nr=1;
7     while (nr)
8     {printf ("Wpisz numer funkcji (1, 2, 3 lub 4)\n,
       Koniec programu - wpisz 0  ");
9          scanf( "%d", &nr);
```

wywołanie funkcji

```
10         if (nr==1) printf( "suma liczb = %f", suma());
11         else
12         if(nr==2)
13         {     float a,b,k;
14               printf(" wpisz konce przedzialu i
krok\n");
```

wywołanie funkcji

```
15               scanf("%f%f%f", &a, &b, &k);
16               tablica_wartosci(a, b, k);
17         }
18         else
19         if(nr==3)
20         {     int a,b;
21               printf(" wpisz konce przedzialu \n");
22               scanf("%d%d", &a, &b);
```

wywołanie funkcji

```
23               printf(" iloczyn liczb podzielnych przez 3
                  =%d\n", iloczyn(a, b));
24         }
25         else
26         if(nr==4)
27         {     int a,b;
```

wywołanie funkcji

```
28               a=rand()%10;
29               b=zagadka(a);
30               if (b==-1) printf(" nie zgadles\n");
31               else  printf("zgadles za %d razem\n",b);
32         }
33    }
34    printf("\nKONIEC\n");
35    return 0;
36    }
```

definicja funkcji

```
37    // definicje wszystkich funkcji ================
38    float suma()
39    {     int i, n;
40    float s, x;
41    s=0;
42    printf("Podaj ilosc liczb:");
43    scanf("%d", &n);
```

```
44    i=1;
45    while(i<=n)
46    {     printf("Podaj liczbe:");
47          scanf("%f", &x);
48          if(x > 100)  s+=x;
49          i++;
50    }
51    return s;
52    }
```

definicja funkcji

```
53    void tablica_wartosci(float A, float B, float K)
54    {     float x;
55    x=A;
56    while(x<=B)
57    {     printf("x=%f    x+sin(x)=%f\n", x, x+sin(x));
58          x+=K;
59    }
60    }
```

definicja funkcji

```
61    int iloczyn(int A, int B)
62  { int w, il;
63    if (A > B)
64    {     w=A;
65          A=B;
66          B=w;
67    }
68    if ((A+1) % 3 ==0) A++;
69    else if ( (A+2) % 3 ==0) A+=2;
70    w=A;
71    il=1;
72    while( w <= B)
73    {     il*=w;
74          w+=3;
75    }
76    return il;
77  }
```

definicja funkcji

```
78    int zagadka (int a)
79    {     int w, il;
80    il=0;
81    do
82    {     printf(" wpisz liczbę ");
83          scanf("%d", &w);
84          il++;
85    }while (a!= w && il < 6);
86    if (a != w) il=-1;
87    return il;
88  }
```

Komentarz do programu:

Wiersze:

1-4:	prototypy funkcji
6:	przypisanie wartości 1 zmiennej sterującej pętlą
7-33:	czynności wykonane w pętli: odczytanie opcji wybranej przez użytkownika, realizacja wywołanej funkcji. Pętla jest powtarzana dopóki użytkownik wpisuje numer opcji różny od 0.
37-88:	definicje funkcji

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 7.1

Średnia liczb parzystych i nieparzystych

(*) Napisz i wywołaj funkcję, która pobiera od użytkownika liczby całkowite do momentu wpisania cyfry 0 i wyświetla średnie liczb parzystych i nieparzystych.

Zadanie 7.2

Zliczanie wystąpień litery i liczby cyfr

(*) Napisz i wywołaj funkcję, która dla wprowadzonego przez użytkownika zdania zakończonego kropką obliczy ile razy wystąpiła w nim podana litera (np. a) oraz ile było cyfr.

Zadanie 7.3

Obliczenie stanu konta bankowego

(*) Napisz i wywołaj funkcję, która dla podanych w parametrach: wpłaty oraz oprocentowania oblicza i wyświetla po jakim czasie kwota wpłaty będzie co najmniej podwojona i ile dokładnie wynosi.

Zadanie 7.4

Suma szeregu

(*) Napisz i wywołaj funkcję, która dla podanej jako parametr zmiennej EPS oblicza sumę ułamków $\frac{i}{(i+1)^2}$ począwszy od i = 0, 1, … dopóki wartość ułamka jest większa od EPS.

Zadanie 7.5

Znajdowanie największej liczby

(*) Napisz i wywołaj funkcję, która pobiera od użytkownika liczby, ostania z nich ma być równa -1 oraz zwraca największą z tych liczb.

Zadanie 7.6

Ciąg geometryczny

(*) Napisz i wywołaj funkcję, która pobiera od użytkownika liczby dopóki tworzą ciąg geometryczny lub użytkownik wpisał zero. Funkcja zwraca ilość liczb tworzących ciąg geometryczny.

Zadanie 7.7

Ciąg rosnący

(*) Napisz i wywołaj funkcję, która pobiera od użytkownika `n` liczb, sprawdza czy tworzą ciąg rosnący. Wynik funkcji: 1 jeśli tak, 0 jeśli nie.

Zadanie 7.8

Obliczanie sum cząstkowych

(*) Napisz funkcję, która pobiera od użytkownika n liczb, po wpisaniu każdej liczby oblicza i wyświetla dotychczasową sumę.

Zadanie 7.9

Sprawdzanie

(*) Napisz i wywołaj funkcję, która sprawdzi ile kolejnych liczb całkowitych należy dodać do siebie, aby otrzymać największą wartość sumy nie przekraczającej zadanej wartości MAX.

Zadanie 7.10

Menu programu

(*) Napisz program wyświetlający menu programu do wielokrotnego wyboru funkcji przez użytkownika. Zakończenie programu następuje po wybraniu opcji KONIEC.

8. Instrukcja iteracyjna FOR

Cel rozdziału

Zapoznanie z realizacją algorytmów wymagających wielokrotnego powtarzania sekwencji instrukcji z wykorzystaniem pętli licznikowej. Nabycie praktycznych umiejętności programowania algorytmów iteracyjnych z wykorzystaniem instrukcji FOR (*Dla*).

Podstawy teoretyczne

Instrukcja iteracyjna (pętli) FOR służy do powtarzania fragmentu programu dopóki wartość wyrażenia wpisanego w instrukcję pętli jest różna od zera, czyli wyrażenie jest prawdziwe.

Działanie FOR:

- nadanie zmiennej `zm` wartości `wp`,
- sprawdzenie wartości wyrażenia,
  jeśli jest prawdziwe:
  - wykonana jest instrukcja, zmiana wartości `zm`
    i powrót do sprawdzenia wartości wyrażenia;

  jeśli nie jest prawdziwe:
  - to wykonywane są instrukcje poza zasięgiem pętli.

```
for (zm=wp ; wyrażenie ; zmiana wartości zm)
instrukcja; //zasięg pętli
```

Jeżeli ma być powtórzonych wiele instrukcji należy je ująć w nawiasy { }, czyli zapisać w postaci instrukcji złożonej:

```
for (zm=wp ; wyrażenie ; zmiana wartości zm)
{instrukcja1;  //zasięg pętli
 instrukcja2;  //zasięg pętli
...            //zasięg pętli
}
```

Przykład 8.1

```
for(k=1; k<6; k++)
printf("*\n");
```

Na ekranie zostanie wyświetlonych w pionie 5 gwiazdek.

Przykład 8.2

```
int i, x=1;
for (i=1; i<4; i++)
{ x *= i;
  printf( "i=%d \t i!=%d\n", i, x);
}
```

W wyniku działania instrukcji na ekranie będzie wyświetlone:

i=1 i!=1
i=2 i!=2
i=3 i!=6

Przykład 8.3

```
int i, x=0;
for (i=-5; i<=5; i+=3)
x+=i;
printf( "%d\n", x);
```

Na ekranie zostanie wyświetlona suma liczb: -5, -2, 1, 4 czyli liczba ***-2****.*

Arkusz przykładowych ćwiczeń

Ćwiczenie 8.1

Obliczenie iloczynu

Dany jest zbiór N liczb rzeczywistych. Napisz funkcję wczytującą dane i obliczającą iloczyn tych liczb.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1     float iloczyn()
2 {   int i, n;
3     float s, il, x;
4     il=1;
```

```
5     printf("Podaj ilosc liczb:");
6     scanf("%d", &n);
7     for (i=1; i<=n; i++)
8     {     printf("Podaj liczbe:");
9           scanf("%f", &x);
10          il*=x;
11    }
12    return il;
13 }
```

Komentarz do programu:

Wiersze:

- 2,3: deklaracje zmiennych: n – ilość liczb, i – numer liczby, x – liczba podana przez użytkownika
- 4: wartość początkowa iloczynu
- 6,9: wczytywanie wartości zmiennych
- 7: instrukcja sterująca for
- 8-10: blok pętli for
- 12: przekazanie wyniku z funkcji

Użycie funkcji przedstawiono na listingu 8.1.

Ćwiczenie 8.2

Obliczenie średniej arytmetycznej liczb

Dany jest zbiór N liczb rzeczywistych. Napisz funkcję wczytującą dane oraz obliczającą i wyświetlającą średnią arytmetyczną dodatnich liczb. Jeśli nie ma liczb dodatnich należy wyświetlić odpowiedni komunikat.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1     void srednia()
2  {  int i, n, il;
3     float s, x;
4     s=0;
5     il=0;
6     printf("Podaj ile liczb:");
7     scanf("%d", &n);
8     for (i=1; i<=n; i++)
9     {     printf("Podaj liczbe:");
10          scanf("%f", &x);
11          if (x >0)
```

```
12          { il++;
13            s+=x;
14          }
15    }
16    if(il > 0)
17    {     s=s/il;
18          printf("srednia liczb dodatnich=%f\n", s);
19    }
20    else printf(" Brak liczb dodatnich\n");
21 }
```

Komentarz do programu:

Wiersze:

- 2,3: deklaracje zmiennych: n – ilość liczb, i – numer liczby, x – liczba podana przez użytkownika, s – suma liczb > 0, średnia, il – ile liczb jest > 0
- 4-5: nadanie wartości początkowych
- 9-15: blok pętli for
- 11-14: jeśli liczba > 0, to dodanie liczby do sumy i zwiększenie ilości liczb
- 16-19: jeśli wystąpiły liczby dodatnie, to obliczenie i wyświetlenie średniej
- 20: wysiedlenie komunikatu

Użycie funkcji przedstawiono na listingu 8.1.

Ćwiczenie 8.3

Obliczenie wartości wyrażenia w zadanym przedziale. Druga wersja rozwiązania ćwiczenia 7.2

Dla danych wartości A, B, K napisz funkcję obliczającą i wyświetlającą wartości wyrażenia x+sin x, dla x∈ <A,B>, z krokiem K, to znaczy w punktach: x = A, x = A+K, x = A+2K, …

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 void tablica_wartosci(float A, float B, float K)
2 {   float x;
3     x=A;
4     for(x=A; x<=B; x+=K)
5     {     printf("x=%f,  x+sin(x)=%f\n", x, x+sin(x));
6     }
7 }
```

Użycie funkcji przedstawiono na listingu 8.1.

Listing 8.1

Instrukcja iteracyjna FOR

prototypy funkcji

```
1 float iloczyn();
2 void srednia();
3 void tablica_wartosci(float, float, float);
4
5 int main ( )
6 {  int nr=1;
7    while (nr)
8    {    printf ("Wpisz numer funkcji (1, 2 lub 3)\n");
9         printf ("Koniec programu - wpisz 0  ");
10        scanf( "%d", &nr);
11        if (nr==1) printf( "iloczyn liczb = %f",
              iloczyn());
12        else
13        if(nr==2) srednia();
14        else
15        if(nr==3)
16        {  float a,b,k;
17           printf(" wpisz konce przedzialu i krok\n");
18             scanf("%f%f%f", &a, &b, &k);
19             tablica_wartosci(a, b, k);
20        }
21   }
22   printf("\nKONIEC\n");
23   return 0;
24   }
25 float iloczyn()
26 { int i, n;
27   float s, x, il;
28   il=1;
29   printf("Podaj ilosc liczb:");
30   scanf("%d", &n);
31   for (i=1; i<=n; i++)
32   {    printf("Podaj liczbe:");
33        scanf("%f", &x);
34        il*=x;
35   }
36   return il;
37 }
38   void srednia()
39 { int i, n, il;
40   float s, x;
41   s=0;
42   il=0;
43   printf("Podaj ilosc liczb:");
```

wywołanie funkcji

wywołanie funkcji

wywołanie funkcji

definicja funkcji iloczyn()

definicja funkcji srednia()

```
44    scanf("%d", &n);
45    for (i=1; i<=n; i++)
46    {     printf("Podaj liczbe:");
47          scanf("%f", &x);
48          if (x >0)
49          { il++;
50            s+=x;
51          }
52    }
53    if( il > 0 )
54    {     s=s/il;
55          printf("srednia liczb dodatnich=%f\n", s);
56    }
57    else printf(" Brak liczb dodatnich\n");
58 }
```

definicja funkcji tablica_ wartosci()

```
59 void tablica_wartosci(float A, float B, float K)
60 { float x;
61   x=A;
62   for(x=A; x<=B; x+=K)
63   {     printf("x=%f    x+sin(x)=%f\n", x, x+sin(x));
64   }
65 }
```

Komentarz do programu:

Wiersze:

1-3:	prototypy funkcji
6:	przypisanie wartości 1 zmiennej sterującej pętlą
7-33:	czynności wykonane w pętli: odczytanie opcji wybranej przez użytkownika, realizacja wywołanej funkcji. Pętla jest powtarzana dopóki użytkownik wpisuje numer opcji różny od 0
25-65:	definicje funkcji

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 8.1

Szlaczek

(*) Napisz funkcję wyświetlającą linię wybranym znakiem (np. 50 gwiazdek w jednym wierszu). Wywołaj tę funkcję.

Zadanie 8.2

Zliczanie liczb nieparzystych

(*) Napisz funkcję obliczającą ilość liczb nieparzystych spośród `n` liczb całkowitych podanych przez użytkownika. Wywołaj tę funkcję.

Zadanie 8.3

Średnia liczb parzystych

(*) Napisz funkcję obliczającą średnią arytmetyczną liczb parzystych spośród N liczb całkowitych podanych przez użytkownika. Wywołaj tę funkcję.

Zadanie 8.4

Suma lub iloczyn n liczb

(*) Napisz funkcję, która pobiera od użytkownika `n` liczb. Jeśli n < 10 to funkcja oblicza iloczyn liczb większych od pierwszej, jeśli n >= 10 oblicza sumę. Funkcja wyświetla wynik. Wywołaj tę funkcję.

Zadanie 8.5

Średnia arytmetyczna wybranych liczb

(*) Napisz funkcję, która pobiera od użytkownika `n` liczb, oblicza średnią arytmetyczną liczb większych od pierwszej. Wywołaj tę funkcję.

Zadanie 8.6

Lista płac

(*) Pracownicy mają otrzymać podwyżkę według następującego algorytmu:
gdy zarobki > 5000 – podwyżka wynosi 5%,
gdy zarobki <= 5000 – podwyżka wynosi 10%.
Napisz funkcję, która pobiera od użytkownika płace `n` pracowników, wyświetla nową listę płac oraz sumę nowych płac. Wywołaj tę funkcję.

Zadanie 8.7

Silnia n liczb nieujemnych

(*) Napisz funkcję, która oblicza liczbę `n`! (n mniejsze niż 13).
Wywołaj tę funkcję.

Zadanie 8.8

Liczby trzycyfrowe

(*) Napisz funkcję wyznaczania wszystkich liczb trzycyfrowych, które są równe sumie sześcianów swoich cyfr, np. $153 = 1^3 + 5^3 + 3^3$.
Wywołaj tę funkcję.

Zadanie 8.9

Ciąg arytmetyczny

(**) Napisz funkcję sprawdzającą, czy N liczb podanych przez użytkownika tworzy ciąg arytmetyczny. Wywołaj tę funkcję.

Wskazówka:

Liczby tworzą ciąg arytmetyczny, jeśli różnica pomiędzy sąsiednimi liczbami jest stała.

Zadanie 8.10

Suma szeregu

(**) $$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Napisz funkcję, która oblicza wartość sin x za pomocą powyższego wzoru dla K składników sumy. Wywołaj tę funkcję. Pobranie wartości kąta w radianach (jeśli w stopniach to przelicz na radiany), zmiennej K i wyświetlenie wyniku w main().

Zadanie 8.11

Trójki pitagorejskie

(**) Napisz program, którego zadaniem będzie odnalezienie n trójek pitagorejskich (tj. trójki liczb całkowitych a, b, c takich, że $a^2 + b^2 = c^2$), składających się z liczb mniejszych od n. Liczba n musi być z przedziału od 10 do 200.

- Wyniki przedstaw w kolejności od pierwszej znalezionej trójki do ostatniej.
- Dodatkowo, wyniki przedstaw w kolejności odwrotnej.

Zadanie 8.12

Liczby pierwsze

(**) Napisz program, którego zadaniem będzie odnalezienie i wypisanie n kolejnych liczb pierwszych, oraz podanie ilości liczb, które okazały się nie być pierwszymi. Liczba n musi być z przedziału od 1 do 20.

- Odnalezione liczby pierwsze należy przedstawić w kolejności od pierwszej do ostatniej znalezionej.
- Odnalezione liczby pierwsze przedstaw w kolejności odwrotnej.

Zadanie 8.13

Liczby

(**) Napisz funkcje wyznaczania liczb:

- automorficznych (liczby, które występują na końcu swego kwadratu, np. 25),
- pierwszych (liczby naturalne > 1, które dzielą się tylko przez 1 i samą siebie),
- pitagorejskich (liczby spełniające warunek: $a^2 + b^2 = c^2$) dla danego przedziału <m, n>, tzn., że (a, b, c)$\in$<m, n>.

Funkcja main powinna zawierać menu, pozwalające na wybór rodzaju poszukiwanych liczb.

Zadanie 8.14

Wyznaczanie współrzędnych

(**) Napisz program, którego zadaniem będzie wyznaczenie współrzędnych Y i Z n punktów, na podstawie podanych współrzędnych X.
Przyjmij, że X(n) to wartości całkowite, oraz:

Y(n)= X(n) + X(n-1) gdy n > 1, lub X(n) gdy n = 1

Z(n)= X(n)+Y(n)*Y(n)

- Zastosuj sprawdzenie, czy wartość n zawiera się w przedziale domkniętym od 1 do 10 i czy wartości X są nieujemne.
- Wyniki przestaw w kolejności od ostatniego do pierwszego punktu.

Przykład formatowania dla X(9) = 9 i X(10) = 10:

"10. X[10] = 10 Y[10] = 19 Z[10] = 371

Zadanie 8.15

Fundusz emerytalny

(**) Pracownik pewnej firmy wpłaca do funduszu emerytalnego X zł składki. Fundusz potrąca z niej Y%, a resztę inwestuje osiągając stały dochód Z% w skali roku. Jaką kwotą będzie dysponował pracownik po N latach pracy? Napisz i wywołaj funkcję realizującą ww. algorytm – zadanie 1.6.

Zadanie 8.16

Logowanie do systemu

(**) Dostęp do systemu realizowany jest po podaniu odpowiedniego hasła. Napisz i wywołaj funkcję umożliwiającą logowanie użytkownika (max. 5 prób) i przekazanie tajnej wiadomości. Po 5 nieudanych próbach następuje wyjście z programu. Po poprawnym zalogowaniu się użytkownik podaje wiadomość, która jest szyfrowana (do każdego znaku dodawana jest pewna liczba). Wiadomość może być wyświetlona w zaszyfrowanej formie, lub wyświetlona w oryginale (po zastosowaniu odpowiedniego dekodera) – zadanie 1.7.

9. Zmienne i wskaźniki. Wykorzystanie wskaźników do komunikacji między funkcjami

Cel rozdziału

Zapoznanie z pojęciem wskaźnika. Nabycie praktycznych umiejętności zastosowania wskaźników w programie.

Podstawy teoretyczne

Wskaźnik jest zmienną, której wartością jest adres innej zmiennej.

Deklaracja wskaźnika:

```
typ *nazwa;
```

Typ powinien być taki, jak typ zmiennej, której adres będzie przechowywany w zmiennej wskaźnikowej. Wskaźnik należy powiązać z zmienną.

```
float x, * wx;
wx = &x;
```

Zmienna wskaźnikowa będąca parametrem funkcji może służyć do przekazania wyniku z tej funkcji.

Przykład 9.1

```
int a, *wa;
wa = &a;
a = 7; // lub *wa = 7;
printf ("adres zmiennej a:%p\n", wa);
printf ("wartość zmiennej a:%d\n", a);
printf ("wartość pod adresem wa:%d\n", *wa);
```

Na ekranie będzie wyświetlony szesnastkowo adres zmiennej oraz dwukrotnie liczba 7.

Przykład 9.2

```
int f(int x, int *w);
int main()
{  int a=3, b, c;
   c = f(a, &b);
   printf( "%d,\t%d,\t%d\n", a, b, c);
   return 0);
}
int f(int x, int *w)
{ *w = x*x; //obliczenie kwadratu liczby
   return (*w)*x; //obliczenie sześcianu liczby
}
```

*Kwadrat liczby a będzie przekazany z funkcji poprzez zmienną wskaźnikową (*w), sześcian liczby a poprzez* ***return****. Na ekranie będą wyświetlone liczby: 3, 9, 27.*

Arkusz przykładowych ćwiczeń

Ćwiczenie 9.1

Obliczenie pola powierzchni i objętości prostopadłościanu

Dane są długości boków prostopadłościanu. Napisz funkcję obliczającą pole powierzchni całkowitej i objętość bryły. Pole jest przekazane poprzez return, objętość poprzez czwarty parametr będący wskaźnikiem. Jeśli dane są błędne, to pole ma mieć wartość 0.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 float pr(float a, float b, float c, float *V)
2 {  *V = 0 ; // ta instrukcja jest opcjonalna
3    if ( a <= 0 || b <= 0 || c <= 0) return 0;
4    *V= a*b*c;
5    return 2*(a*b + a*c + b*c);
6 }
```

Komentarz do programu:

Wiersze:

3: sprawdzenie poprawności danych. Jeśli są błędne, wyjście z funkcji z wynikiem 0
4: obliczenie objętości
5: obliczenie pola i przekazanie wyniku na zewnątrz funkcji

Użycie funkcji przedstawiono na listingu 9.1.

Ćwiczenie 9.2

Obliczenie pola powierzchni i objętości walca

Dane są: promień podstawy i wysokość walca. Napisz funkcję obliczającą pole powierzchni całkowitej i objętość bryły. Pole jest przekazane poprzez return, objętość przez trzeci parametr będący wskaźnikiem. Jeśli dane są błędne, to pole ma mieć wartość 0.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 float w(float r, float h, float *V)
2 {  *V = 0 ; // ta instrukcja jest opcjonalna
3    if ( r <= 0 || h <= 0) return 0;
4    * V= M_PI*r*r*h;
5    return M_PI*r*(2*h + r);
6 }
```

Użycie funkcji przedstawiono na listingu 9.1.

Listing 9.1

Zmienne wskaźnikowe

prototypy funkcji

```
1 float pr(float, float, float, float *);
2 float w(float, float, float *);
3 int main ( )
5 {   float x, y, pole, obj;
6     int nr;
7     printf ( "program oblicza pole i objętość bryl:\n");
8     printf ("prostopadloscian - wpisz 1 \n");
9     printf ("walec - wpisz 2 \n");
10    printf ("Wpisz numer funkcji 1 lub 2\n");
11    scanf( "%d", &nr);
12    if (nr==1)
13    { float z;
14      printf( "podaj długości boków prostopadloscianu ");
15      scanf ("%f%f%f", &x, &y, &z);
```

wywołanie funkcji

```
16      pole = pr(x, y, z, &obj);
17      if(pole != 0)
18         printf (" pole= %f\tobjetosc=%f\n", pole, obj);
19      else
20         printf (" niepoprawne dane\n");
21    }
22    else
23      if (nr==2)
24      {  printf( "podaj promien i wysokosc walca ");
```

wywołanie funkcji

```
25         scanf ("%f%f", &x, &y);
26         pole = w(x, y, &obj);
27       if(pole != 0)
```

```
28          printf (" pole= %f\tobjetosc=%f\n", pole, obj);
29        else
30          printf (" niepoprawne dane\n");
31       }
32     else
33     printf (" niepoprawny numer opcji dane\n");
34     return 0;
35 }
36 float pr(float a, float b, float c, float *V)
37 {  *V = 0 ; // ta instrukcja jest opcjonalna
38     if ( a <= 0 || b <= 0 || c <= 0) return 0;
39     * V= a * b * c;
40     return 2 * (a * b + a * c + b * c);
41}
42 float w(float r, float h, float *V)
43 {  *V = 0 ; // ta instrukcja jest opcjonalna
44     if ( r <= 0 || h <= 0) return 0;
45     * V= M_PI * r * r * h;
46     return M_PI * r * (2 * h +  r);
47 }
```

definicja funkcji pr()

definicja funkcji w()

Komentarz do programu:

Wiersze:

1,2:	prototypy funkcji
7-11:	menu programu
12-21:	wprowadzenie danych, wywołanie funkcji, wyświetlenie wyników dla prostopadłościanu
23-31:	wprowadzenie danych, wywołanie funkcji, wyświetlenie wyników dla walca
36-47:	definicje funkcji

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 9.1

Iloczyn liczb

(*) Napisz funkcję obliczającą iloczyn liczb z przedziału <1,100> spośród N liczb podanych przez użytkownika. Funkcja zwraca iloczyn poprzez parametr wskaźnikowy a informację, czy były liczby spełniające warunek poprzez return. Wywołaj tę funkcję, wyświetl wynik: wartość iloczynu lub komunikat o braku liczb.

Zadanie 9.2

Dwie największe liczby

(**) Napisz funkcję zwracającą 2 największe liczby spośród N różnych liczb podanych przez użytkownika. Wywołaj tę funkcję, wyświetl wyniki.

Zadanie 9.3

Lokaty bankowe

(**) Bank oferuje lokaty: półroczną oprocentowaną na p=1% (w skali roku) oraz roczną oprocentowaną na p=2%, p1<p2. Klient wpłaca X zł na rok. Napisz funkcję zwracającą w parametrach wskaźnikowych uzyskane po roku wypłaty dla obu wariantów lokat. Wczytanie danych w main(). Wywołaj tę funkcję, wyświetl wyniki.

Zadanie 9.4

Układ równań liniowych

(**) Napisz funkcję rozwiązującą układ 2 równań liniowych metodą wyznaczników. Wynik przekazany przez return:

1 – układ ma rozwiązanie,
0 – układ jest nieoznaczony,
-1 – układ jest sprzeczny.

Rozwiązanie układu czyli wartości x, y przekazane przez parametry wskaźnikowe. Wprowadzenie danych w main(). Wywołaj tę funkcję, wyświetl wyniki.

Wskazówka:

Metoda wyznaczników dla układu równań liniowych.
Układ równań:

$$A1*x + B1*y = C1$$
$$A2*x + B2*y = C2$$

Wyznaczniki:

$$W = A1*B2 - A2*B1$$
$$Wx = C1*B2 - C2*B1$$
$$Wy = A1*C2 - A2*C1$$

Interpretacja wartości wyznaczników:
Jeśli $W = 0$ i $Wx = 0$ i $Wy = 0$ to układ jest nieoznaczony.
Jeśli $W = 0$ i ($Wx \neq 0$ lub $Wy \neq 0$) to układ jest sprzeczny.
Jeśli $W \neq 0$ to $x = Wx / W$, $y = Wy / W$.

Zadanie 9.5

Położenie punktu

(**) Napisz funkcję, która zwraca odległość punktu od początku układu współrzędnych oraz informację o położeniu tego punktu (numer ćwiartki układu współrzędnych).

Zadanie 9.6

Zamówienie

(**) Dane: cena hurtowa 1 sztuki towaru, liczba zamówionych sztuk. Jeśli klient zamawia mniej niż 10 sztuk, to płaci cenę detaliczną, która jest o 20% wyższa. Napisz funkcję, która zwraca koszt zamówienia i cenę 1 sztuki towaru w tym zamówieniu.

Zadanie 9.7

Liczby

(**) Napisz funkcję, która pobiera od użytkownika n liczb, zwraca ilość liczb dodatnich i ilość liczb = 0.

10. Tablice statyczne. Wskaźniki do tablic

Cel rozdziału

Zapoznanie z typem złożonym – tablicą. Nabycie praktycznych umiejętności zastosowania tablic w programie.

Podstawy teoretyczne

Tablica jest uporządkowanym zbiorem zmiennych tego samego typu. Kolejność elementów w zbiorze określona jest poprzez indeks/ indeksy.

Deklaracja tablicy jednowymiarowej i dwuwymiarowej:

```
typ nazwa[ilość elementów];  //jedno
typ nazwa[ilość wierszy][ilość kolumn];  //dwu
```

Zmienna indeksowa musi być typu całkowitego i przyjmuje wartości nieujemne. Pierwszy element tablicy ma indeks równy 0.

Zapis:
`X[0]` oznacza pierwszy element tablicy jednowymiarowej
`T[0][0]` oznacza pierwszy element tablicy dwuwymiarowej

Nazwa tablicy jest jednocześnie wskaźnikiem na pierwszy element.

```
int tab[5]={1,2,3,4,5};
tab      <->  &tab[0];
tab+2    <->  &tab[2];  //ten sam adres
*(tab+2) <->  tab[2];   //ta sama wartość
```

Przykład 10.1

```
int t[10], i;
for (i=0; i<10; i++)
t[i]= i * i;
```

Tablica **`t`** *będzie zawierała10 elementów: kwadraty liczb od 0 do 9*
Zapis indeksowy

Przykład 10.2

```
int t[10], i;
for (i=0; i<10; i++)
*(t + i)= i *i;
```

Tablica **t** *będzie zawierała 10 elementów: kwadraty liczb od 0 do 9*
Zapis wskaźnikowy

Arkusz przykładowych ćwiczeń

Ćwiczenie 10.1

Wczytywanie liczb typu float do tablicy

Napisz funkcję wczytującą `N` liczb do tablicy jednowymiarowej. Tablica jest przekazywana przez parametr.

Wynik:

Funkcja może być zdefiniowana w następujący sposób, gdzie `N` zdefiniowana jest jako stała `(#define N 10` – więcej szczegółów w rozdziale 15. Dyrektywy preprocesora):

```
1 void czytaj(float a[])
2 {  int  i;
3    for (i=0; i<N; i++)
4    {    printf("wpisz liczbe a[%d]: ", i);
5         scanf("%f", &a[i]);
6    }
7 }
```

Komentarz do przykładu:

Wiersze:

1: nagłówek funkcji: parametrem funkcji jest nazwa tablicy jednowymiarowej (adres pierwszego elementu)

3-6: blok pętli: dla każdej wartości zmiennej ***i*** wyświetlenie komunikatu, wprowadzenia kolejnej liczby do tablicy

Ćwiczenie 10.2

Obliczenie różnicy pomiędzy największym i najmniejszym elementem w tablicy zawierającej n różnych liczb typu float

Napisz funkcję wyznaczającą maksimum i minimum w tablicy liczb oraz zwracającą poprzez return różnicę tych wartości. Elementy tablicy są parametrami przekazanymi do funkcji. Rozmiar tablicy okreslany jest jako stała `(#define N 10)`:

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 float R(float t[])
2 {  int  i;
3    float max, min=t[0];
4    max=min;
5    for (i=1; i<N; i++)
4    {    if (t[i] > max ) max=t[i];
5         else
6         if (t[i] < min ) min=t[i];
7    }
8    return max-min;
9 }
```

Komentarz do przykładu:

Wiersze:

2-4: deklaracje zmiennych lokalnych, przypisanie wartości początkowej zmiennym max, min
5-7: blok pętli: wyznaczenie wartości max, min dla całej tablicy
8: przekazanie wyniku z funkcji

Użycie funkcji przedstawiono na listingu 10.1.

Ćwiczenie 10.3

Utworzenie w funkcji nowej tablicy

Napisz funkcję tworzącą nową tablicę poprzez przepisanie do niej liczb dodatnich z danej tablicy. Rozmiar nowej tablicy jest przekazywany z funkcji poprzez return, tablica jest przekazywana z funkcji przez parametr.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 int nowa(float a[], float b[])
2 {  int  i, j=0;
3    for (i=0; i<N; i++)
4    if (a[i] > 0)
5    {    b[j] = a[i];
6         j++;
7    }
8    return j;
9 }
```

Komentarz do przykładu:

Wiersze:

2: deklaracje zmiennych lokalnych. Nadanie wartości `0` zmiennej `j`, która jest indeksem w nowej tablicy, na koniec zawiera informację o liczbie elementów w nowej tablicy

3-7: utworzenie nowej tablicy `b`

8: przekazanie wyniku funkcji

Użycie funkcji przedstawiono na listingu 10.1.

Listing 10.1

Tablice

prototypy funkcji

wywołanie funkcji czytaj(), *R()*, *nowa()*

definicja funkcji czytaj()

definicja funkcji R()

```
0 #define N 10
1 void czytaj(int n, float a[]);
2 float R(float t[]);
3 int nowa(float a[], float b[]);
4 int main ( )
5 {  int i, n=6, k;
6    printf( "Program wczytuje liczby do tablicy,\n");
7    printf( "wyswietla roznice pomiedzy największym\n");
8    printf( " i najmniejszym elementem tablicy,\n");
9    printf( "przepisuje liczby dodatnie do nowej
              tablicy\n");
10   printf( "oraz wyswietla te tablice\n\n");
11   float t[N];
12   float nt[N];
13   czytaj(t);
14   printf ("max-min = %5.2f \n", R(t));
15   k = nowa(t, nt);
16   printf ("\nNowa tablica \n");
17   for (i=0; i<k; i++)
18   printf ("%.2f\n", nt[i]);
19   return 0;
20 }
21 void czytaj(float a[])
22 { int  i;
23   for (i=0; i<N; i++)
24   {    printf("wpisz liczbe a[%d]: ", i);
25        scanf("%f", &a[i]);
26   }
27 }
28 float R(float t[])
29 { int  i;
30   float max, min=t[0];
31   max=min;
32   for (i=1; i<N; i++)
```

```
33    {     if (t[i] > max ) max=t[i];
34          else
35          if (t[i] < min ) min=t[i];
36    }
37    return max-min;
38  }
```

definicja funkcji nowa( )

```
39  int nowa(float a[], float b[])
40  { int  i, j=0;
41    for (i=0; i<N; i++)
42    if (a[i] > 0)
43    {     b[j] = a[i];
44          j++;
45    }
46    return j;
47  }
```

WYKONANIE PROGRAMU

dla:
#define N 6

```
Program wczytuje liczby do tablicy,
wyswietla roznice pomiedzy najwiekszym
 i najmniejszym elementem tablicy,
przepisuje liczby dodatnie do nowej tablicy
oraz wyswietla te tablice

wpisz liczbe A[0]: 3
wpisz liczbe A[1]: 7
wpisz liczbe A[2]: -10
wpisz liczbe A[3]: 21
wpisz liczbe A[4]: 7
wpisz liczbe A[5]: 16
max-min = 31.00

Nowa tablica
3.00
7.00
21.00
7.00
16.00
```

Komentarz do programu:

Wiersze:

1-3:	prototypy funkcji
10:	podwójny znak nowej linii (\n\n)
11,12:	deklaracje tablic typu float
13-15:	wywołanie funkcji
17,18:	druk w pętli elementów tablicy `nt`
42:	w nowej tablicy będą tylko elementy > 0

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 10.1

Wyznaczenie największej i najmniejszej liczby w tablicy, szukanie w tablicy konkretnej liczby

(*) **Dane:** tablica N liczb rzeczywistych uporządkowanych rosnąco.
Napisz funkcję sprawdzającą, czy wśród tych liczb jest
liczba = (maksymalna + minimalna)/2
i wyświetlająca odpowiedź.
Napisz program, w którym wczytane są dane, wywołana funkcja.

Zadanie 10.2

Tworzenie nowej tablicy

(**) **Dane:** tablica N liczb rzeczywistych uporządkowanych rosnąco.
Napisz funkcję tworzącą nową tablicę zawierającą liczby większe od 100.
Napisz program, w którym wczytane są dane, wywołana funkcja, wyświetlona nowa tablica.

Zadanie 10.3

Funkcje modyfikujące i tworzące tablicę liczb

(***) **Dane:** tablica N różnych liczb rzeczywistych.
Napisz funkcje realizujące następujące zadania:

- wyświetlenie informacji o dostępnych funkcjach (menu) oraz pobrać od użytkownika numer wybranej funkcji przekazać go do `main()`,
- zamiana miejscami elementu maksymalnego i minimalnego,
- zapisanie elementów tablicy w odwrotnej kolejności,
- utworzenie tablicy tab_y zawierającej kwadraty danych liczb, tablicy tab_z zawierającą sześciany danych liczb.

Napisz program, w którym wczytane są dane, wywołana funkcja menu, a następnie funkcja wybrana przez użytkownika.
Wyświetl wyniki działania funkcji.

Zadanie 10.4

Przekątne tablicy

(**) **Dane:** tablica liczb rzeczywistych o wymiarach N wierszy, N kolumn.
Napisz funkcje realizujące następujące zadania:

- obliczenie iloczynu elementów na głównej przekątnej.
- obliczenie sumy elementów nad główną przekątną.

Napisz program, w którym wczytane są dane, wywołane funkcje, wyświetlone wyniki.

Wskazówka:

Elementy tablicy będące na głównej przekątnej mają taki sam numer wiersza i kolumny np.: `x[2][2]`

Zadanie 10.5

Suma elementów w kolumnie tablicy

(*) **Dane:** tablica liczb rzeczywistych o wymiarach N wierszy, K kolumn.
Napisz funkcję obliczającą sumę elementów w kolumnie wybranej przez użytkownika.
Napisz program, w którym wczytane są dane, wywołana funkcja, wyświetlony wynik.

UWAGA:

Sprawdź w programie, czy podany numer kolumny jest poprawny.

Zadanie 10.6

Zamiana miejscami elementów tablicy

(*) **Dane:** tablica liczb rzeczywistych o wymiarach N wierszy, K kolumn.
Napisz funkcję zamieniającą miejscami elementy tablicy z pierwszej i ostatniej kolumny.
Napisz program, w którym wczytane są dane, wywołana funkcja, wyświetlona zmieniona tablica.

Zadanie 10.7

Szukanie zera w tablicy

(*) **Dane:** tablica liczb rzeczywistych o wymiarach N wierszy, K kolumn.
Napisz funkcję sprawdzającą, czy w tablicy jest liczba 0. Wynik w postaci liczby 1 (jest) lub liczby 0 (nie ma) funkcja przekazuje poprzez return.
Napisz program, w którym wczytane są dane, wywołana funkcja, wyświetlony wynik.

Zadanie 10.8

Tworzenie tablic

(**) **Dane:** tablica liczb rzeczywistych o wymiarach N wierszy, K kolumn.
Napisz funkcję która:

- tworzy tablicę tab_d zawierającą elementy dodatnie,
- tablicę tab_u zawierającą elementy ujemne,
- oblicza liczbę elementów równych 0.

Napisz program, w którym wczytane są dane, wywołana funkcja, wyświetlone wyniki.

Zadanie 10.9

Badanie symetryczności tablicy dwuwymiarowej

(**) **Dane:** tablica liczb rzeczywistych o wymiarach N wierszy, K kolumn.
Napisz funkcję, która pobiera od użytkownika dwuwymiarową tablicę liczbową. Jeśli liczba kolumn jest nieparzysta, to funkcja sprawdza, czy tablica jest symetryczna względem środkowej kolumny. Jeśli liczba kolumn jest parzysta, to funkcja sprawdza, czy brzegowe wiersze są takie same. Funkcja wyświetla wynik. Wywołaj tę funkcję.

Zadanie 10.10

Modyfikacja tablicy dwuwymiarowej

(***) **Dane:** tablica liczb rzeczywistych o wymiarach N wierszy, K kolumn.
Napisz funkcję, która modyfikuje tablicę w następujący sposób: jeśli ponad połowa elementów w kolumnie jest równa 0, to należy wyzerować pozostałe elementy w tej kolumnie (powtórzyć dla wszystkich kolumn).
Utworzyć tablicę tab_b zawierającą tylko numery kolumn, w których dokonano operacji zerowania elementów.
Napisz program, w którym wczytane są dane, wywołana funkcja, wyświetlone wyniki.

11. Dynamiczna alokacja pamięci. Tablice dynamiczne

Cel rozdziału

Zapoznanie z funkcjami przydziału i zwolnienia pamięci. Zastosowanie tych funkcji do typu tablicowego.

Podstawy teoretyczne

Funkcja `malloc` przydziela wolną pamięć. Liczba przydzielonych bajtów jest parametrem funkcji. Funkcja zwraca wskaźnik na pierwszy bajt tego obszaru. Jeśli nie jest możliwe przydzielenie pamięci, to funkcja zwraca wskaźnik `null`. W starszych implementacjach funkcja należała do typu „wskaźnik do char", w ANSI C należy do „wskaźnik do void". Bezpośrednie nadanie wartości wskaźnika do void wskaźnikowi innego typu nie jest błędem. To ze względu na czytelność zalecane jest użycie jawnego rzutowania.

```
float *t= (float*)malloc (100 *sizeof(float))
```

Funkcja `free` zwalnia obszar pamięci alokowany dla wskaźnika będącego jej parametrem.

```
free (t)
```

Przykład 11.1

```
int *t, i;
t = (int*)malloc(10 *sizeof(int));
if (!t)
 {printf (" brak wolnej pamięci "); exit(1);}
for (i=0; i<10; i++)
 {*(t + i)= i * i;
  printf("\n%d", *(t + i));
 }
free (t);
```

*Funkcja **malloc** przydziela pamięć dla tablicy 10 liczb całkowitych (stąd rzutowanie na int). Jeśli funkcja zwróci null czyli 0, nastąpi wyjście z program. Jeśli przydzielenie pamięci powiedzie się, to w pętli for będą obliczone i wyświetlone elementy tablicy **t**: kwadraty liczb od 0 do 9. Następnie pamięć zajęta przez tablicę **t** będzie zwolniona.*

Przykład 11.2

```
int **p, i;
p = (int*)malloc(sizeof(int *)*5);
for(i=0; i < 5; i++)
p[i] = (int*)malloc(sizeof(int)*10);
```

***p** to tablica wskaźników do int zaalokowana dynamicznie. Deklaracja dynamiczna tablicy liczb całkowitych składającej się z 5 wierszy i 10 kolumn. Wiersz jest tutaj traktowany jak tablica jednowymiarowa zawierająca 10 liczb, dlatego przydział pamięci wykonany jest w pętli dla każdego wiersza.*

Arkusz przykładowych ćwiczeń

Ćwiczenie 11.1

Wczytywanie liczb typu float do tablicy

Napisz funkcję wczytującą n liczb do tablicy jednowymiarowej i przekazującą wskaźnik do tej tablicy poprzez return. Rozmiar tablicy jest przekazywany do funkcji przez parametr.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 float *czytaj(int n)
2 { int  i;
3   float *tab = (float*)malloc (n *sizeof(float));
4     for (i=0; i<n; i++)
5     {   printf("wpisz liczbe tab[%d]: ", i);
6         scanf("%f", tab + i);
7     }
8    return tab;
7 }
```

Komentarz do przykładu:

Wiersze:

3: deklaracja i alokacja pamięci dla wynikowej tablicy

6: wczytywanie danych, wskaźnikowy zapis elementów tablicy
8: przekazanie z funkcji adresu tablicy

Użycie funkcji przedstawiono na listingu 11.1.

Ćwiczenie 11.2

Wyświetlenie elementów tablicy, której adres jest parametrem funkcji

Napisz funkcję wyświetlającą liczby dodatnie z tablicy, której adres jest parametrem funkcji. Rozmiar tablicy jest także przekazywany do funkcji przez parametr.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 void wynik(int n, float * k)
2 {  int  i;
3    printf ("\nliczby dodatnie:\n ");
4    for (i=0; i<n; i++)
5    if (*(k+i)>0) printf ("%.2f\t", *(k+i));
4    printf ("\n ");
5 }
```

Użycie funkcji przedstawiono na listingu 11.1.

Listing 11.1

Tablice jednowymiarowe

prototypy funkcji

```
0 #define N 7
1 float *czytaj();
2 void wynik(float *k);
```

```
3 int main ( )
4 {
```

wywołanie funkcji

```
5    float *t;
6    t = czytaj(n);
7    wynik(t);
8    return 0;
9 }
```

definicja funkcji czytaj()

```
10 float * czytaj()
11 { int  i;
12   float *tab=(float*)malloc (N * sizeof( float));
13   for (i=0; i<N; i++)
14   {    printf("wpisz liczbe tab[%d]: ", i);
```

```
15        scanf("%f", tab + i);
16    }
17    return tab;
18  }
19  void wynik(float *k)
20  { int  i;
21    printf ("\nliczby dodatnie:\n ");
22    for (i=0; i<N; i++)
23    if (*(k+i)>0) printf ("%.2f\t", *(k+i));
24    printf ("\n ");
25  }
```

definicja funkcji wynik()

Komentarz do programu:

Wiersze:

0:	definiowanie stałej N
1,2:	prototypy funkcji
4,5:	deklaracje zmiennych
6,7:	wywołanie funkcji czytaj i wynik
10-25:	definicje funkcji

Ćwiczenie 11.3

Obliczenie sum w wierszach tablicy kwadratowej

Napisz funkcję obliczającą sumy w wierszach i sumy w kolumnach w dwuwymiarowej tablicy liczb i zwracającą wyniki jako dwie tablice. Rozmiar i elementy danej tablicy są parametrami przekazanymi do funkcji. Tablica sum w kolumnach jest przekazana z funkcji jako parametr wskaźnikowy, wskaźnik do utworzonej tablicy sum w wierszach jest przekazany poprzez return.

Wynik:

Funkcja może być zdefiniowana w następujący sposób:

```
1 float * sumy(int n, int k, float t[][k], float *k)
2 {  int  i, j;
3    float *w, sum;
4    w = (float*)malloc (n * sizeof( float));
5    for (i=0; i<n; i++)
4    {    sum=0;
5         for (j=0; j<k; j++)
6         sum + =t[i][j];
7         w[i] = sum;
8    }
9    for (j=0; j<k; j++)
```

```
10    {     sum=0;
11          for (i=0; i<n; i++)
12          sum + =t[i][j];
13          k[j] = sum;
14    }
15    return w;
16  }
```

Komentarz do przykładu:

Wiersze:

3-8: deklaracja, alokacja pamięci i obliczanie sum w wierszach
9-14: obliczanie sum w kolumnach

Użycie funkcji przedstawiono na listingu 11.2.

Listing 11.2

Tablice wielowymiarowe

prototyp funkcji

```
1 float *sumy(int n, int k, float t[][k], float *kw);
2 int main ( )
3 {  int  i, j, n=3, k=4;
4    float t[n][k];
5    float sk[k], *sw;
6    for (i=0; i<n; i++)
7    for (j=0; j<k; j++)
8    {    printf("wpisz liczbe t[%d][%d]: ", i, j);
9         scanf("%f", &t [i][j]);
10   }
11   sw = sumy(n, k, t, sk);
12   printf ("\nsumy w wierszach:\n ");
13   for (i=0; i<n; i++)
```

wywołanie funkcji

```
14   printf ("%.1f\t", sw[i]);
15   printf ("\nsumy w kolumnach:\n ");
16   for (j=0; j<k; j++)
17   printf ("%.1f\t", sk[j]);
18   return 0;
19 }
20 float *sumy(int n, int k, float t[][k], float *kw)
```

*definicja funkcji *sumy()*

```
21 { int  i, j;
22   float * w, sum;
23   w = malloc (n *sizeof(float));
24   for (i=0; i<n; i++)
```

przydział wolnej pamięci

```
25   {    sum=0;
26        for (j=0; j<k; j++)
```

```
27          sum = sum + t[i][j];
28          w[i] = sum;
29     }
30     for (j=0; j<k; j++)
31     {    sum=0;
32          for (i=0; i<n; i++)
33          sum = sum + t[i][j];
34          kw[j] = sum;
35     }
36     return w;
37  }
```

Komentarz do przykładu:

Wiersze:

1:	prototyp funkcji
3-10:	deklaracje zmiennych, wprowadzenie danych
11:	wywołanie funkcji
12-17:	wyświetlenie wyników
36:	przekazanie wyniku funkcji

DZIAŁANIE PROGRAMU

```
wpisz liczbe t[0][0]: 2
wpisz liczbe t[0][1]: 3
wpisz liczbe t[0][2]: 7
wpisz liczbe t[0][3]: 2
wpisz liczbe t[1][0]: -8
wpisz liczbe t[1][1]: -11
wpisz liczbe t[1][2]: 4
wpisz liczbe t[1][3]: 17
wpisz liczbe t[2][0]: 8
wpisz liczbe t[2][1]: 7
wpisz liczbe t[2][2]: -9
wpisz liczbe t[2][3]: 5

sumy w wierszach:
 14.0   2.0     11.0
sumy w kolumnach:
 2.0    -1.0    2.0     24.0
```

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 11.1

Tablica statyczna i dynamiczna

(*) Napisz program, który zapełni 100 elementową tablicę statyczną wylosowanymi liczbami, obliczy ile z nich jest z podanego przedziału <A,B> a następnie utworzy tablicę dynamiczną odpowiedniego rozmiaru i zapełni ją tymi liczbami. Obydwie tablice powinny zostać wyświetlone.
Wykorzystaj odpowiednie funkcje.

Zadanie 11.2

Maraton

(*) Napisz program ewidencjonujący czasy osiągnięte przez zawodników maratonu i wyszukujący zwycięzcę. Liczba zawodników N nie jest dokładnie znana, zakłada się, że startowa pula numerów jest ograniczona do 300 osób. Należy ewidencjować dokładnie tyle czasów ile zawodników. Program powinien wyszukać najlepszy wynik i wyświetlić numer startowy/ numery osoby/ osób z tym wynikiem.
Wykorzystaj odpowiednie funkcje.

Zadanie 11.3

Tworzenie tablicy dynamicznej z tablicy jednowymiarowej

(*) **Dane: tab** – tablica N liczb całkowitych. Napisz funkcję która tworzy i zwraca poprzez parametr nową dynamiczną tablicę `nt` zawierającą liczby różne od zera z tablicy `tab`. Liczba elementów w nowej tablicy jest wyznaczana w main().
Napisz program, w którym wczytane są dane, wywołana funkcja, jeśli nowa tablica nie będzie pusta, oraz wyświetlone wyniki.

Zadanie 11.4

Tworzenie tablic dynamicznych z tablicy dwuwymiarowej

(*) **Dane:** tablica liczb rzeczywistych o wymiarach N wierszy, K kolumn.
Napisz funkcje f_1, f_2 które:

- f_1 – tworzy tablicę tab_d zawierającą elementy dodatnie,
- f_2 – tablicę tab_u zawierającą elementy ujemne.

Napisz program, w którym wczytane są dane, wywołane funkcje, wyświetlone wyniki. Rozmiary nowych tablic deklarowanych dynamicznie określ w main ( ).

Zadanie 11.5

Tworzenie tablicy dynamicznej dwuwymiarowej

(*) W hurtowni jest N towarów. Tablica tab_dane zawiera w kolumnie zerowej ceny, w kolumnie pierwszej liczbę towarów. Napisz funkcję tworzącą tablicę informacji o towarach (cena, liczba), których wartość jest większa od liczby podanej przez użytkownika. Napisz program, w którym wczytane są dane, wywołana funkcja, wyświetlone wyniki. Tablice są deklarowane dynamicznie.

Zadanie 11.6

Tablica dynamiczna zwracana przez wskaźnik

(*) Tablica tab_punkty zawiera współrzędne N punktów na płaszczyźnie. Napisz funkcję tworzącą tablicę odległości punktów od początku układu współrzędnych. Wskaźnik do tablicy jest przekazany z funkcji przez return. Napisz program, w którym wczytane są dane, wywołana funkcja, wyświetlone wyniki.

12. Łańcuchy znakowe i funkcje łańcuchowe

Cel rozdziału

Zapoznanie z metodami inicjalizacji łańcuchów znakowych i funkcjami działającymi na łańcuchach znakowych.

Podstawy teoretyczne

Łańcuch znakowy to ciąg składający się jednego lub wielu znaków, który przechowywany jest w tablicach zbudowanych z elementów typu `char`. Ostatni element tablicy to znak ASCII `\0` o kodzie równym 0, który służy do oznaczenia końca łańcucha. Stąd tablica musi mieć rozmiar o jeden większy niż liczba znaków tworzących łańcuch.

Stała łańcuchowa (literał łańcuchowy) – sekwencja znaków w "".

```
"Stoi na stacji lokomotywa."
```

Znaki cudzysłowu nie należą do łańcucha. Znak cudzysłowu wprowadzamy przez zastosowanie kodu `\"`.

Zmienna łańcuchowa – inicjalizacja (tylko w definicji)

```
char array1[] = "Nic nie mam";
```

```
char array2[] = {'N','i','c',' ','n','i','e',' ',
                 'm','a','m','.','\0'};
```

Do deklaracji i inicjalizacji łańcucha można zastosować również zapisu wskaźnikowego.

```
char *array3 = "Stoi na stacji lokomotywa.";
```

PAMIĘTAJ: Inicjalizacje łańcucha pokazane powyżej są możliwe przy jednoczesnej deklaracji typu tablicy.

Wczytywanie łańcuchów – funkcje: `scanf(), gets(), fgets()`

- Funkcja `scanf()` służy do wczytywania różnych zmiennych, w tym łańcuchów, pobierając znaki ze standardowego urządzenia wejścia do chwili napotkania znaku niedrukowanego (nowej linii `'\n'`, tabulatora, spacji, który pozostaje). Funkcja ta dodaje na końcu łańcucha znak `'\0'`.

```
scanf("%s", tablica_znakowa);
```

- Funkcja `gets()` służy do wczytywania łańcuchów pobierając znaki (w tym spacje) ze standardowego urządzenia wejścia do chwili napotkania znaku nowej linii `'\n'`(ale bez niego), dodaje na końcu znak `'\0'` i przekazuje łańcuch do programu. ***Nie sprawdza*** liczby wprowadzanych znaków z zadeklarowanym rozmiarem tablicy.

```
gets(tablica_znakowa);
```

- Funkcja `fgets()` umożliwia podanie liczby znaków, które zostaną wprowadzone. Po odczytaniu znaku `'\n'` pozostawia go w łańcuchu.

```
fgets(nazwa_tablicy, liczba_znakow, strumien_wej);
```

Wyprowadzanie łańcuchów – funkcje: `printf(), puts()` `fputs()`

- Funkcja `printf()` służy do wyprowadzania wartości różnych zmiennych, w tym łańcuchów, na standardowe urządzenie wyjścia i można ją użyć w postaciach (znak \n należy dodać do łańcucha formatującego, by przejść do nowej linii):

```
printf(tablica_znakowa);
printf("%s\n", tablica_znakowa);
printf("%ps\n", tablica_znakowa);
```

gdzie: p – minimalna liczba znaków wyprowadzanego pola, jeżeli p jest ze znakiem minus, zmienna w polu wyrównywana jest do lewej

- Funkcja `puts()` wyprowadza łańcuch zakończony znakiem '\0' i do wyprowadzanego łańcucha dodaje znak końca linii

```
puts(stala_lancuchowa);
puts(tablica_znakowa);
```

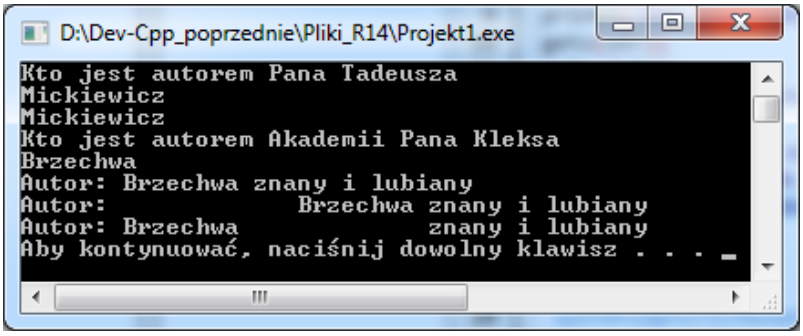
- Funkcja `fputs()` wyprowadza łańcuch na wskazane wyjście (plik lub ekran – stdout)

```
fputs(tablica_znakowa, stdout);
```

Przykład 12.1

```
char str[5], str2[20];
printf("Kto jest autorem Pana Tadeusza\n");
gets(str);
puts (str);
printf("Kto jest autorem Akademii Pana
Kleksa\n");
scanf("%s",str2);
printf ("Autor: %s znany i lubiany\n",str2);
printf ("Autor: %20s znany i lubiany\n",str2);
printf ("Autor: %-20s znany i lubiany\n",str2);
```

WYKONANIE PROGRAMU

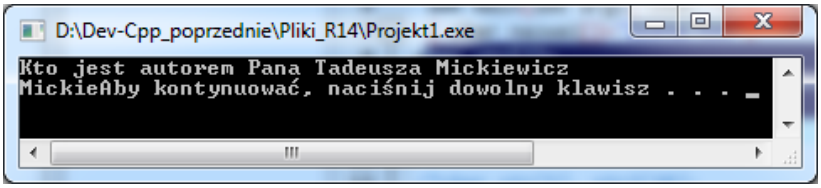


Funkcja `scanf()` *umożliwia formatowanie tekstu określając minimalną szerokość pola i wyrównanie.*

Przykład 12.2

```
char str[7];
printf("Kto jest autorem Pana Tadeusza ");
fgets(str,7,stdin);
fputs(str,stdout);
```

WYKONANIE PROGRAMU



Pomimo wpisania pełnego nazwiska funkcja `fgets()` *pobrała tylko 6 znaków, siódme miejsce zajął znak końca linii '*`\n`*'.*
Przy wyświetleniu `fputs()` *nie dodaje znaku przejscia do nowego wiersza.*

Funkcje łańcuchowe znajdują się bibliotece **string.h** i umożliwiają przetwarzanie łańcuchów.

Operacja podstawienia (kopiowania).

```
strcpy (dokad, skad);
```

dokad – *tablica znakowa lub napis,*
skad – *napis lub stała napisowa,*

UWAGA!

Poza inicjalizacją łańcuchów w chwili definiowania ich typów napisów nie można podstawiać do tablic przez zastosowanie znaku '='.

Operacja łączenia (konkatenacja).

```
strcat (napis1, napis2);
```

Funkcja dołącza do dotychczasowej zawartości **napis1** zawartość **napis2**, pierwszy argument zmienia się, drugi pozostaje bez zmian.

Porównywanie napisów.

```
strcmp (napis1, napis2);
```

Funkcja zwraca wartość `0` gdy napisy są równe leksykograficznie (słownikowo), wartość dodatnią gdy `napis1` jest słownikowo większy (leży dalej w słowniku), wartość ujemną gdy `napis1` jest mniejszy (leży bliżej początku słownika). Przy porównywaniu korzystamy z kodu ASCII (cyfry < duże litery < małe litery).

Obliczanie długości łańcucha znaków.

```
strlen (napis1);
```

Funkcja oblicza długość łańcucha znaków od pierwszego znaku do znaku terminalnego '\0'.

Arkusz przykładowych ćwiczeń

Ćwiczenie 12.1

Podstawienie stałej napisowej do łańcucha

Wykorzystując funkcje łańcuchowe napisz instrukcje, które pozwalają kopiować napis `Witam` oraz `tutaj` do dwóch różnych tablic. Rozmiar pierwszej tablicy zdefiniuj na 30, zaś drugiej nie podawaj.

Wynik:

Funkcje mogą być wykorzystane w następujący sposób:

```
1 char str1[30], str2[30];
2 strcpy(str1, "Witam ");
3 printf ("  %s\n", str1);
4 strcpy(str2, "tutaj");
5 printf(str2);
```

Komentarz do przykładu:

Wiersze:

1: deklaracja tablic typu `char`, w drugim przypadku bez podania rozmiaru
2,4: kopiowanie stałej napisowej odpowiednio do tablicy `str1`, `str2`
3: wypisanie łańcucha z formatowaniem
5: wypisanie łańcucha z bez formatowania

WYKONANIE PROGRAMU

```
Kopiowanie: stala napisowa do lancucha

  Witam
tutaj
```

Ćwiczenie 12.2

Łączenie stałych napisowych w jeden łańcuch

Połącz napisy: `'Witam'` oraz `'tutaj'` w jeden napis, a następnie skopiuj go do nowej tablicy. Wykorzystaj kod z ćwiczenia 12.1.

Wynik:

Ćwiczenie może być zrealizowane w następujący sposób:

```
1 char str3[50];
2 printf ("Konkatenacja str1 i str2:\n ");
3 strcat(str1, str2);
4 printf ("%s\n", str1);
5 printf("Kopiowanie str1 do str3:\n");
6 strcpy(str3, str1);
7 printf ("%s\n", str3);
```

Komentarz do programu:

Wiersze:

1: deklaracja nowej tablicy
3: łączenie dwóch łańcuchów `str1` i `str2` w łańcuchu `str1`
4: wypisanie zawartości łańcucha
6: kopiowanie zawartości łańcucha `str1` do łańcucha `str3`
7: wypisanie zawartości łańcucha

WYKONANIE PROGRAMU

```
Konkatenacja str1 i str2:
 Witam tutaj
Kopiowanie str1 do str3:
Witam tutaj
```

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 12.1

Wyśrodkowanie wprowadzonego napisu

(*)

Napisz program pozwalający na wyśrodkowywanie dwóch napisów wprowadzanych z klawiatury. Przyjmij, że liczba znaków w linii wynosi 80, zaś napis jest wprowadzany bez kontrolowania długości wprowadzanych znaków. Wypisz napis wprowadzony, a następnie wyśrodkowany. Procedurę wyśrodkowania napisz w osobnej funkcji.

Zadanie 12.2

Wyrównywanie wprowadzonego napisu do prawej

(*) Napisz program pozwalający na wyrównywanie do prawej linii wiersza Lokomotywa Juliana Tuwina:

Stoi na stacji lokomotywa,
Ciężka, ogromna i pot z niej spływa:
Tłusta oliwa.
Stoi i sapie, dyszy i dmucha,
Żar z rozgrzanego jej brzucha bucha:
Buch – jak gorąco!
Uch – jak gorąco!

Przyjmij, że liczba znaków w linii wynosi 80, zaś napis jest wprowadzany bez kontrolowania długości wprowadzanych znaków. Procedurę wyrównania do prawej napisz w osobnej funkcji. Wiersz zapisz w jednej tablicy dwuwymiarowej.

Zadanie 12.3

Obliczanie długości napisu

(*) Napisz program obliczający długość zdania powstałego przez kontatenację następujących stałych łańcuchowych: `"Biblioteka"`, `"string.h"`, `"pozwala"`, `"na"`, `"zastosowanie"`, `"funkcji"`, `"do"`, `"lancuchow"`, `"znakowych"`. Pamiętaj o wprowadzeniu spacji, aby powstałe zdanie wyglądało właściwie.

Zadanie 12.4

Zamiana liter tekstu na duże litery

(*) Napisz funkcję pozwalająca na zamianę wprowadzonego napisu na napis, który będzie wypisany dużymi literami. Wprowadzony tekst może zawierać duże, małe litery oraz inne znaki. Do tego celu zastosuj funkcję kontrolującą liczbę wprowadzonych znaków. Wypisz napis wprowadzony i zmieniony. Wykorzystując funkcję z zadania 12.1 wyśrodkuj zmieniony napis.

Zadanie 12.5

Zakończenie działania programu

(*) Napisz funkcję, w której zakończenie programu nastąpi po wprowadzeniu napisu `"quit"`. Program pozwala na wprowadzanie wielu napisów (pętla nieskończona).

Zadanie 12.6

Słownikowe porządkowanie wyrazów

(**) Napisz funkcję, która uporządkuje leksykograficznie (od litery a do z) pięć trzyliterowych wyrazów wprowadzonych z klawiatury uwzględniając tylko pierwszą literę wyrazów.

Zadanie 12.7

Słownikowe porządkowanie wyrazów

(**) Napisz funkcję, która uporządkuje leksykograficznie (od litery a do z) pięć trzyliterowych wyrazów wprowadzonych z klawiatury uwzględniając kolejne litery wyrazów.

Zadanie 12.8

Tworzenie napisów w odbiciu lustrzanym.

(**) Napisz program pozwalający odwrócić kolejność znaków wprowadzonego napisu. W wersji rozbudowanej zastosuj funkcję zamieniającą napis na duże litery (zadanie 12.4) oraz wyśrodkowujący go (zadanie 12.1, liczba znaków w linii 80).

Zadanie 12.9

Szyfr Cezara

(**) Napisz funkcję, która pozwoli na szyfrowanie wprowadzonego teksu stosując szyfr Cezara. Zastosuj przesunięcie znaków o wartość 3. Zakończenie wprowadzania napisów nastąpi po wpisaniu napisu "`quit`". Wypisz napisy zaszyfrowane.

Zadanie 12.10

Szyfr Cezara

(**) Napisz funkcję dekodującą teksty uzyskane w zadaniu 12.9.

13. Struktury i unie

Cel rozdziału

Zaznajomienie z wykorzystaniem struktur i unii jako typów złożonych do organizacji skomplikowanych danych. Nabycie praktycznych umiejętności wyboru dobrego sposobu reprezentacji danych oraz programowania z zastosowaniem struktur złożonych.

Podstawy teoretyczne

Pojęcie struktura można interpretować jako typ lub jako zmienną. **Struktura** jako typ danych składa się z pól i pozwala na przechowywanie danych różnych typów. Struktura jako zmienna to zmienna typu strukturowego.

Deklaracja struktury: typ z nazwą + zmienna:

```
struct nazwaTypu {    //typ
                  typPola1 nazwaPola1;
                  typPola2 nazwaPola2;
                  ...
                 };
struct nazwaTypu nazwazmiennej; //zmienna
```

Przykład 13.1

```
struct student{
               char nazwisko[25];
               int wiek;
              };
struct student st1;
```

typ o nazwie student – struktura złożona z 2 pól: nazwisko *typu łańcuch znakowy i* wiek *typu całkowitego*
zmienna o nazwie st1 *typu struktura student*

Deklaracja struktury: zmienna podanego typu bez nazwy

```
struct {
         typPola1 nazwaPola1;
         typPola2 nazwaPola2;
         ...
       } zmienna;
```

Przykład 13.2

```
struct {
        char nazwisko[25];
        int wiek;
       } st2;
```

zmienna o nazwie st2 typu struktura złożona z 2 pól: nazwisko typu łańcuch znakowy i wiek typu całkowitego

Definicja struktury: deklaracja+inicjalizacja

```
struct nazwaTypu zmienna={wartP1, wartP2,..};
```

Dostęp do składowych (pola struktury) za pomocą zmiennej:

```
zmienna.nazwaPola
```

Przykład 13.3

```
struct student st3={"Kos",25};
printf("Witaj %s masz %d lat\n", st3.nazwisko,
st3.wiek);
```

Witaj Kos masz 25 lat

Struktura może być polem innej struktury – struktura zagnieżdżona lub elementem tablicy:

Przykład 13.4

```
struct absolwent{
       struct student dane;
       int wynik;
       };
struct absolwent abs1; //struktura zagnieżdżona
struct absolwent grupa[10]; //tablica struktur
```

Dostęp do składowych (pola struktury) za pomocą wskaźnika do struktury:

```
struct nazwaTypu *wskaźnik;//deklaracja wskaźnika
wskaźnik->nazwaPola //dostęp do pola
```

Przykład 13.5

```
struct student st3={”Kos”,25};
struct student *wsk=&st3;
printf("%d \n", st3.wiek);
printf("%d \n", wsk->wiek);
printf("%d \n", (*wsk).wiek);
```

Unia – typ złożony do przechowywania różnych rodzajów danych w tym samym obszarze pamięci (jednak nie równocześnie)
Definicja unii jest analogiczna jak struktury tylko z innym słowem kluczowym:

```
union nazwaTypu {    //typ
                  typPola1 nazwaPola1;
                  typPola2 nazwaPola2;
                  ...
                 };
union nazwaTypu nazwazmiennej; //zmienna
lub
union {
         typPola1 nazwaPola1;
         typPola2 nazwaPola2;
         ...
       } zmienna;
```

Dostęp do składowych (pola unii):

```
zmienna.nazwaPola //za pomocą zmiennej
lub
union nazwaTypu *wskaźnik;//deklaracja wskaźnika
wskaźnik->nazwaPola //za pomocą wskaźnika
```

Przykład 13.6

```
union magazyn {
                int calkowita;
                double rzeczywista;
                char litera; };
union magazyn koszyk;
koszyk.calkowita=22;//22 zapisane w koszyk, zajęte 2 bajty
koszyk.rzeczywista=2.0; //22 usuniete, 2.0 zapisane,
                        //zajęte 8 bajtow
koszyk.litera='a'; // 2.0 usuniete, 'a' zapisane, zajęty 1 bajt;
```

Arkusz przykładowych ćwiczeń

Ćwiczenie 13.1

Dane osobowe

Wczytaj i wyświetl imię i nazwisko 2 osób wykorzystując strukturę oraz przetwarzanie jej przez zmienną i przez wskaźnik.

Wynik:

Program przetwarzający struktury może być zrealizowany w sposób przedstawiony na listingu 13.1.

Listing 13.1

Przetwarzanie struktur

deklaracja struktury

deklaracje dwóch zmiennych strukturowych

definicja funkcji wczytaj()

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  struct daneos {
4       char imie[15];
5       char nazwisko[25];
6       };
7  struct daneos wczytaj();
8  void wyswietl(struct daneos os);
9
10 void wczytaj2(struct daneos *wsk);
11 void wyswietl2(struct daneos *wsk);
12 //==================================
13 int main(int argc, char *argv[])
14 { struct daneos osoba, osoba2;
15   printf("Osoba 1\n");
16   osoba=wczytaj();
17   wyswietl(osoba);
18   printf("Osoba 2\n");
19   wczytaj2(&osoba2);
20   wyswietl2(&osoba2);
21   system("PAUSE");
22   return 0;
23 }
24 //==================================
25 struct daneos wczytaj()
26 { struct daneos os;
27   printf("Podaj imie "); gets(os.imie);
28   printf("Podaj nazwisko "); gets(os.nazwisko);
29   return os;
30 }
```

definicja funkcji wyswietl1()

```
31 void wyswietl(struct daneos os)
32 {
33      printf("%s %s \n", os.imie, os.nazwisko);
34 }
```

definicja funkcji wczytaj2()

```
35 void wczytaj2(struct daneos *wsk)
36 { printf("Podaj imie ");
37    gets(wsk->imie);
38    printf("Podaj nazwisko ");
39    gets(wsk->nazwisko);
40 }
```

definicja funkcji wyswietl2()

```
41 void wyswietl2(struct daneos *wsk)
42 {
43   printf("%s %s \n",  wsk->imie,wsk->nazwisko);
44 }
```

Komentarz do programu:

Wiersze:

- 1,2: dyrektywa kompilatora dołączająca pliki nagłówkowe `stdio.h, stdlib.h`
- 3-6: deklaracja struktury
- 7,8: prototypy funkcji działających na kopiach struktur
- 10,11: prototypy funkcji działających na adresie struktury
- 13: nagłówek funkcji głównej `main`
- 14: deklaracje dwóch zmiennych strukturowych
- 16: wczytywanie danych osoby 1 – przetwarzanie przez zmienną
- 17: wyświetlenie danych osoby 1 – przetwarzanie przez zmienną
- 19: wczytywanie danych osoby 2 – przetwarzanie przez wskaźnik
- 20: wyświetlenie danych osoby 2 – przetwarzanie przez wskaźnik
- 21: wywołanie funkcji `system` – zatrzymanie ekranu
- 25-44: definicje zapowiedzianych wcześniej funkcji

Ćwiczenie 13.2

Dane klienta

Wczytaj i wyświetl dane 2 klientów: jeden to osoba fizyczna opisana imieniem, nazwiskiem i peselem, drugi to firma opisywana nazwą. Wykorzystaj unię oraz przetwarzanie jej przez zmienną i przez wskaźnik.

Wynik:

Program przetwarzający struktury i unie może być zrealizowany w sposób przedstawiony na listingu 13.2.

Listing 13.2

Przetwarzanie struktur i unii

deklaracja struktury

deklaracja unii

deklaracja struktury

deklaracje funkcji

deklaracje zmiennych strukturalnych

wywołanie funkcji

wywołanie funkcji

wywołanie funkcji

definicja funkcji wczytaj()

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  struct daneos {
4       char imie[15];
5       char nazwisko[25];
6       char pesel[11];
7       };
8  union dane {
9      struct daneos osoba;
10     char firma[60];
11      };
12 struct daneklienta {
13       int typ; //0-osoba,  inne-firma
14       union dane klient;
15       };
16 struct daneklienta wczytaj(int t);
17 void wyswietl(struct daneklienta kl, int t);
18
19 void wczytaj2(struct daneklienta *wsk, int t);
20 void wyswietl2(struct daneklienta *wsk, int t);
21 //===================================
22 int main(int argc, char *argv[])
23 { struct daneklienta klient1, klient2;
24  int wybor;
25  printf("Klient 1\n");
26  printf("Podaj typ klienta 0 - osoba,
27         inna wartość - firma\n");
28  scanf("%d", &wybor);
29  klient1=wczytaj(wybor);
30  wyswietl(klient1, wybor);
31  printf("Klient 2\n");
32  printf("Podaj typ klienta 0 - osoba,
33          inna wartość - firma\n");
34  scanf("%d", &wybor);
35  wczytaj2(&klient2, wybor);
36  wyswietl2(&klient2, wybor);
37  system("PAUSE");
38  return 0;
39 }
40 //===================================
41 struct daneklienta wczytaj(int t)
42 { struct daneklienta temp;
43   if(t==0)
44     {printf("Podaj imie ");
45      scanf("%s",temp.klient.osoba.imie);
46      printf("Podaj nazwisko ");
47      scanf("%s",temp.klient.osoba.nazwisko);
```

```
48      printf("Podaj pesel ");
49      scanf("%s",temp.klient.osoba.pesel);
50      return temp;
51      }
52      else
53      {
54       printf("Podaj nazwe firmy ");
55       scanf("%s",temp.klient.firma);
56       return temp;
57      }
58 }
59 void wyswietl(struct daneklienta kl, int t)
60 {
61  if(t==0)
62   {printf("%s %s %s \n",  kl.klient.osoba.imie,
63    kl.klient.osoba.nazwisko,
64    kl.klient.osoba.pesel);}
65   else
66   {printf("%s\n", kl.klient.firma);}
67 }
68 void wczytaj2(struct daneklienta *wsk, int t)
69 { if(t==0)
70    { printf("Podaj imie ");
71     scanf("%s",wsk->klient.osoba.imie);
72     printf("Podaj nazwisko ");
73     scanf("%s",wsk->klient.osoba.nazwisko);
74     printf("Podaj pesel ");
75     scanf("%s",wsk->klient.osoba.pesel);
76    }
77    else
78    { printf("Podaj nazwe firmy ");
79     scanf("%s",wsk->klient.firma);}
80 }
81 void wyswietl2(struct daneklienta *wsk, int t)
82 { if(t==0)
83   {printf("%s %s %s \n",
84      wsk->klient.osoba.imie,
85    wsk->klient.osoba.nazwisko,
86    wsk->klient.osoba.pesel);}
87   else
88   {printf("%s\n", wsk->klient.firma);}
89 }
```

definicja funkcji wyswietl()

definicja funkcji wczytaj2()

definicja funkcji wyswietl2()

Komentarz do programu:

Wiersze:

1,2: dyrektywa kompilatora dołączająca pliki nagłówkowe `stdio.h, stdlib.h`

3-7: deklaracja struktury `daneos`

8-11: deklaracja unii `dane`
12-15: deklaracja struktury `daneklienta`
16,17: prototypy funkcji działających na kopiach struktur
19,20: prototypy funkcji działających na adresie struktury
22: nagłówek funkcji głównej `main`
23: deklaracje dwóch zmiennych strukturowych
28,34: wybór typu klienta
29 : wczytywanie danych klienta 1 – przetwarzanie przez zmienną
30: wyświetlenie danych klienta 1 – przetwarzanie przez zmienną
35: wczytywanie danych klienta 2 – przetwarzanie przez wskaźnik
36: wyświetlenie danych klienta 2 – przetwarzanie przez wskaźnik
37: wywołanie funkcji `system` – zatrzymanie ekranu
41-89: definicje zapowiedzianych wcześniej funkcji

WYKONANIE PROGRAMU

```
Klient 1
Podaj typ klienta 0 - osoba, inna wartosc - firma
0
Podaj imie Jerzy
Podaj nazwisko Montusiewicz
Podaj pesel 42041100001
Jerzy Montusiewicz 42041100001
Klient 2
Podaj typ klienta 0 - osoba, inna wartosc - firma
3
Podaj nazwe firmy MAG
MAG
Press any key to continue . . .

--------------------------------
Process exited after 51.04 seconds with return value 0
Press any key to continue . . .
```

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 13.1

Płaca pracownika

(*) Oblicz płacę pracownika fizycznego. Zadeklaruj strukturę o polach: imię nazwisko, liczba godzin, stawka, premia w %.Napisz funkcje do

wczytywania danych, obliczania płacy i wyświetlania. Zastosuj przetwarzanie struktur przez zmienną i wskaźnik. Wywołaj te funkcje.

Zadanie 13.2

Płace grupy pracowników

(**) Oblicz sumaryczną kwotę do wypłaty dla N pracowników fizycznych. Zadeklaruj tablicę struktur o polach: imię nazwisko, liczba godzin, stawka, premia w %. Wyświetl listę płac i sumaryczna kwotę. Podaj nazwiska osób zarabiających najwięcej. Zdefiniuj odpowiednie funkcje.

Zadanie 13.3

Nagrody dla pracowników

(**) Zadeklaruj unię do przechowywania danych o nagrodach w postaci: albo kwota pieniężna, albo wycieczka w dane miejsce i w danym terminie albo list pochwalny. Dla N pracowników przypisz nagrody różnego typu. Wyświetl przydział nagród w postaci: imię, nazwisko i dane nagrody. Zdefiniuj odpowiednie funkcje.

Zadanie 13.4

Nagrody dla studentów

(**) Zadeklaruj strukturę do przechowywania danych o szczęśliwych numerkach wylosowanych dla studentów (imię, nazwisko, numerek). Dla grupy N studentów wpisz imiona i nazwiska, wylosuj im szczęśliwe numerki. Ustal zwycięzców gry wg jednej z zasad:

- wygrywa osoba/osoby z największym numerkiem,
- wygrywa osoba/osoby, której numerek jest najbliższy średniej wylosowanych liczb.

Zadanie 13.5

Średnia ocen studenta i grupy

(**) Zadeklaruj strukturę student o polach: imię, nazwisko, oceny (tablica 5 ocen). Dla grupy N studentów (tablica) oblicz średnią ocen każdego studenta i średnią grupy. Zdefiniuj odpowiednie funkcje.

Zadanie 13.6

Cukierki

(**) Zadeklaruj strukturę cukierek zawierającą informacje o nazwie producenta, nazwie cukierków, cenie za 1 kg, ilości w kg, dacie produkcji, dacie przydatności do spożycia.
Wprowadź dane.
Wyświetl informacje o cukierkach wybranego producenta, których cena za 1kg jest zawarta w przedziale od X do Y zł.
Wyświetl informację o nazwach i ilości cukierków, którym skończył się termin przydatności do spożycia.
Zdefiniuj odpowiednie funkcje.

Zadanie 13.7

Plan zajęć

(**) Zadeklaruj strukturę PozycjaPlanu zawierającą informację o zajęciach: dzień tygodnia, godzina rozpoczęcia, godzina zakończenia, przedmiot, wykładowca, nr sali.
Wprowadź dane o swoim planie zajęć.
Wyświetl informacje o: zajęciach w danym dniu tygodnia, zajęciach z danego przedmiotu.
Podaj ile razy w tygodniu rozpoczynasz zajęcia o 8 rano.
Zdefiniuj i wywołaj odpowiednie funkcje.

Zadanie 13.8

Ankieta

(**) Zadeklaruj strukturę ankieta zawierającą informacje: imię, nazwisko, adres (ulica, nr domu, nr mieszkania), płeć i wiek. Określ:

- ile kobiet i ilu mężczyzn mieszka w jednym wybranym domu,
- ilu mężczyzn w wieku <18 – 60> mieszka na jednej wybranej ulicy.

Zdefiniuj i wywołaj odpowiednie funkcje.

Zadanie 13.9

Stypendium dla studentów

(**) Napisz program wyznaczania stypendium studentom na podstawie wyników sesji zgodnie z zasadami:
Jeżeli wszystkie oceny to 5 – przyznawane jest podwyższone stypendium
Jeżeli wszystkie oceny to 5 lub 4 przyznawane jest zwykłe stypendium
Jeżeli wśród ocen jest 3 – stypendium się nie należy.

Program powinien wyświetlać:
- listę studentów z ocenami i ich średnią,
- listę studentów z podwyższonym stypendium,
- listę studentów ze zwykłym stypendium.

Zadanie 13.10

Biblioteka

(**) W bibliotece są książki, gazety, czasopisma. Dla każdej pozycji należy podać:
- tytuł,
- rok wydania (dla książek), datę druku (dla gazet i czasopism),
- autora (dla książki), redaktora naczelnego (dla gazet), skład Rady programowej (dla czasopism),
- objętość.

Wyświetlić informacje o pozycjach z danego roku.

Zadanie 13.11

Współrzędne

(**) Napisz program, który wyznaczy współrzędne Z i Y z 'n' punktów podanych przez użytkownika sprawdzając przy tym czy liczba ta jest z przedziału od 3 do 11. Przyjmujemy, że X(n) to wartości całkowite, oraz:

$$Y(n)= X(n) + 2*X(n-1)$$
$$Z(n)= 2*(X(n)+Y(n)*Y(n))$$

Punkty należy potraktować jako struktury. Wyniki przestaw w kolejności od pierwszego do ostatniego punktu. Przykład formatowania dla:

X(5)=5 i X(4)=4: Y[5]=13; Z[5]=348

Zadanie 13.12

Korporacja

(**) Napisz program przechowujący i przetwarzający dane pracowników Korporacji. Dane pracowników wprowadza użytkownik (z poziomu okna konsoli). Dane pracownika: imię, nazwisko, wiek, rok zatrudnienia, wysokość pensji netto oraz przydział do jednego z trzech działów firmy (numer oznaczający dział w firmie, np. księgowość to 1, produkcja to 2).

a) Program przedstawia, który z pracowników: ma najniższą pensję, ma najwyższą pensję, jest zatrudniony najdłużej, jest zatrudniony najkrócej, kto jest najstarszy, a kto najmłodszy.

b) Program powinien zmienić pensje pracowników w taki sposób że: osoby, które pracują <5, 10) lat dostają 10% podwyżki, ci co pracują od <10, 20) lat, dostają 20% podwyżki, a ci co pracują >20 lat dostana podwyżki 25%. Po tej operacji program ponownie sprawdza dane z punktu a).

c) Program powinien sprawdzić: czy najmłodszy pracownik jest zatrudniony najdłużej, czy najstarszy pracownik jest zatrudniony najkrócej.

d) Program powinien policzyć i przedstawić jaki jest średni wiek pracowników w każdym z działów firmy, jaki jest ich średni staż, ilu pracowników pracuje w każdym z działów, oraz wyliczyć wielkość średniej pensji bez i z podziałem na działy.

14. Pliki

Cel rozdziału

Zaznajomienie z obsługą plików do przechowywania danych w zewnętrznej pamięci. Nabycie praktycznych umiejętności pracy z plikami tekstowymi i binarnymi.

Podstawy teoretyczne

Plik to wydzielony fragment pamięci (najczęściej dyskowej) posiadający nazwę (ciąg bajtów).

Rodzaje plików:

- ***tekstowe*** – dane w postaci znakowej(konwersja liczb na znaki),
- ***binarne*** – dane w postaci wewętrznej reprezentacji (brak konwersji, dokładne dane).

Obsługa plików:

- niskopoziomowa – obsługa poprzez funkcje:
  `read(), open(), write(), close() – <io.h>`
- wysokopoziomowa – obsługa poprzez funkcje: **`fopen(), fclose(), fread(), fprintf(),...– <stdio.h>`**

Przetwarzanie plików metodą wysokopoziomową:

1. Otwarcie pliku: **`fopen()`**
2. Wykonanie operacji na pliku
   - Zapis:
     `putc(), fputs(), fprintf(), fwrite()`
   - Odczyt:
     `getc(), fgets(), fscanf(), fread()`
   - Funkcje pomocnicze:
     `feof(); fseek(), rewind(), ftell(),...`
3. Zamknięcie pliku: **`fclose()`**

Otwarcie pliku – zwraca wskaźnik na strukturę typu FILE (identyfikator pliku) lub zerowy (NULL).
Zamknięcie pliku – zwraca 0, jeśli operacja się powiodła lub EOF, jeśli nie.

```
fopen(nazwa_pliku, tryb_otwarcia);//otwarcie
fclose(wskaźnik_pliku); //zamkniecie
```

Tryb otwarcia dla plików tekstowych:

- "r" – odczyt;
- "w" – zapis (nadpisywanie lub tworzenie);
- "a" – zapis na końcu (dopisywanie lub tworzenie)
- "r+" – odczyt i zapis;
- "w+" – odczyt i zapis (nadpisywanie lub tworzenie);
- "a+" – odczyt i zapis na końcu (dopisywanie lub tworzenie)

Tryb otwarcia dla plików binarnych:

"rb"; "wb"; "ab" ; "rb+"; "r+b"; "wb+" ; "w+b" ; "ab+"; "a+b"

Przykład 14.1

```
FILE *fp;
fp=fopen("test.txt", "w");
fclose(fp);
```

Zapis do pliku:

```
putc(znak, wskaźnik_pliku);
fputs( *tekst, wskaźnik_pliku);
fprintf(wskaźnik_pliku, format, dane);
fwrite(adres_w_pamięci, rozmiar_bloku,
ilość_bloków, wskaźnik_pliku);
```

Przykład 14.2

```
FILE *fp; char ch; char * slowo; int dane;
putc(ch,fp);//zapis znaku
fputs(slowo,fp); // zapis łańcucha znaków
fprintf(fp,"%d",dane);//zapis z formatem
```

Odczyt z pliku:

```
getc(wskaźnik_pliku);
fgets( *tekst, dlugosc, wskaźnik_pliku);
fscanf(wskaźnik_pliku, format, dane);
fread(adres_w_pamięci, rozmiar_bloku,
ilość_bloków, wskaźnik_pliku);
ilość_bloków, wskaźnik_pliku);
```

Przykład 14.3

```
FILE *fp; char ch; char buf[256]; int dane;
...
ch= getc(fp);//zapis znaku
fgets(buf, 256,fp); // zapis łańcucha znaków
fscanf(fp,"%d",dane);//zapis z formatem
```

Arkusz przykładowych ćwiczeń

Ćwiczenie 14.1

Imiona przyjaciół w pliku tekstowym

Utwórz i wyświetl plik tekstowy z nazwiskami przyjaciół. Zdefiniuj funkcje zapisu i odczytu nazwisk do pliku. Wywołaj te funkcje.

Wynik:

Program przetwarzający plik tekstowy może być zrealizowany w sposób przedstawiony na listingu 14.1.

Listing 14.1

Przetwarzanie plików tekstowych

deklaracje funkcji

wywołanie funkcji

wywołanie funkcji

definicja funkcji zapisT1()

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int zapisT1(char nazwa[20], FILE *fplik);
5  int odczytT1(char nazwa[20], FILE *fplik);
6 //==================================
7 int main(int argc, char *argv[])
8 { char nazwa1[]="lista.txt";
9  FILE *f;
10 int wynik;
11 wynik=zapisT1(nazwa1,f);
12 if(wynik==0)printf("Operacja zapisu ok\n");
13 wynik=odczytT1(nazwa1,f);
14 if(wynik==0)printf("Operacja odczytu ok\n");
15 system("PAUSE");
16 return EXIT_SUCCESS;
17 }
18 //==================================
19 int zapisT1(char nazwa[20], FILE *fplik)
20 {char nazwisko[25]; int i=0;
21 if((fplik=fopen(nazwa, "a"))==NULL)
22   {printf("Blad otwarcia\n");
23    system("PAUSE"); abort();}
24 printf("Podaj nazwiska kończąc enterem\n");
```

zapis do pliku

```
25 while(gets(nazwisko)!=NULL && nazwisko[0]!='\0')
26 {fprintf(fplik, "%s\n", nazwisko);i++;
27 }
28 if(fclose(fplik)!=0){exit(2);}
29 printf("Do pliku zapisano %d nazwisk\n",i);
30 return 0;
```

definicja funkcji odczytT1()

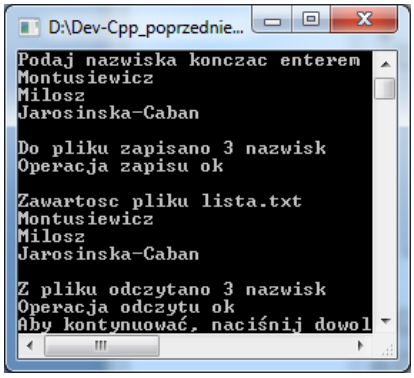
```
31 }
32 //-------------------------
33 int odczytT1(char nazwa[20], FILE *fplik)
34 {char nazwisko[25]; int i=0;
35 if((fplik=fopen(nazwa, "r"))==NULL)
36     {printf("blad otwarcia\n");
37      system("PAUSE"); abort();}
38 printf("\nZawartosc pliku %s\n",nazwa);
```

odczyt z pliku

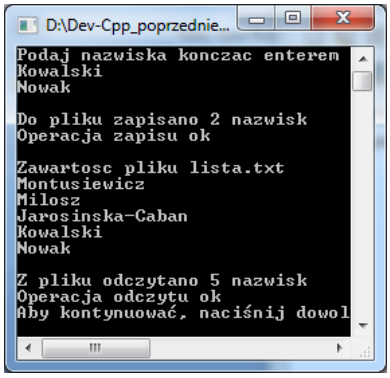
```
39 while(fscanf(fplik,"%s",nazwisko)==1)
40 {puts(nazwisko);i++;
41 }
42 if(fclose(fplik)!=0){exit(2);}
43 printf("\nZ pliku odczytano %d nazwisk\n",i);
44 return 0;
45 }
```

WYKONANIE PROGRAMU

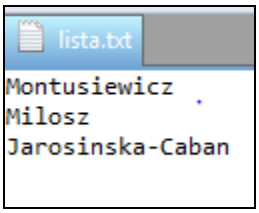
Pierwsze uruchomienie

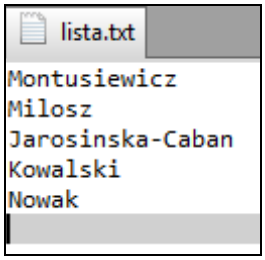


Drugie uruchomienie



Podgląd zawartości pliku `lista.txt`





Komentarz do programu:

Wiersze:

1,2:	dyrektywa kompilatora dołączająca pliki nagłówkowe `stdio.h, stdlib.h`
4,5:	prototypy funkcji zapisu i odczytu z pliku
7:	nagłówek funkcji głównej `main`
9:	deklaracje wskaźnika plikowego
11:	wywołanie funkcji zapisu do pliku
12:	weryfikacja czy zapis sie udał
13:	wywołanie funkcji odczytu z pliku
14:	weryfikacja czy odczyt sie udał
15:	wywołanie funkcji `system` – zatrzymanie ekranu
19-31:	definicja funkcji zapisu nazwisk do pliku
21-23:	weryfikacja otwarcia pliku w trybie dopisywania na końcu
26:	zapis do pliku nazwiska
28,42:	weryfikacja operacji zamknięcia pliku
33-45:	definicja funkcji odczytu nazwisk z pliku
35-37:	weryfikacja otwarcia pliku w trybie odczytu
40:	odczyt z pliku nazwiska

Ćwiczenie 14.2

Dane studenta w pliku binarnym

Utwórz i wyświetl plik binarny z nazwiskami i ocenami studentów. Zdefiniuj funkcje zapisu i odczytu struktury do pliku binarnego. Wywołaj te funkcje.

Wynik:

Program przetwarzający plik binarny może być zrealizowany w sposób przedstawiony na listingu 14.2

Listing 14.2 Przetwarzanie plików binarnych

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
```

```
4  int zapisB(char nazwa[20], FILE *fplik);
5  int odczytB(char nazwa[20], FILE *fplik);
6 //==================================
7 int main(int argc, char *argv[])
8 { char nazwa2[]="studenci.txt";
9  FILE *f;
10 int wynik;
11 wynik=zapisB(nazwa2,f);
12 if(wynik==0)printf("operacja zapisu ok\n");
13 wynik=odczytB(nazwa2,f);
14 if(wynik==0)printf("operacja odczytu ok\n");
15 system("PAUSE");
16 return EXIT_SUCCESS;
17 }
18 //==================================
19 int zapisB(char nazwa[20], FILE *fplik)
20 {struct student st; int i;
21  int rozmiar=sizeof(struct student);
22  int licznik=1; //liczba zapisów
23  if ((fplik=fopen(nazwa, "ab"))==NULL)
24     { printf ("błąd");exit(1); }
25  printf("Podaj liczbę zapisów ");
26  scanf("%d", &licznik);
27 for(i=1;i<=licznik;i++)
28  {printf("Podaj nazwisko %d: ",i);
29   scanf("%s", st.nazwisko);
30   printf("Podaj ocene %d: ",i);
31   scanf("%d", &st.ocena);
32   fwrite(&st, rozmiar, 1, fplik);
33  }
34  if (fclose(fplik) !=0) {printf ("Blad "); }
35 return 0;
36 }
37 //--------------------------
38 int odczytB(char nazwa[20], FILE *fplik)
39 { struct student st;
40   int rozmiar=sizeof(struct student);
41   int licznik=0;
42   if ((fplik=fopen(nazwa, "rb"))==NULL)
43       { printf ("błąd");exit(1); }
44   printf("Zawartość pliku %s\n", nazwa);
45   while(fread(&st,rozmiar,1,fplik)==1)
46      {printf("student: %s ocena: %d\n",
47       st.nazwisko,st.ocena);
48       licznik++;}
49    printf("liczba pozycji: %d\n",licznik);
50    if (fclose(fplik) !=0)
51       {printf ("Blad przy zamykaniu pliku"); }
52 return 0;
53 }
```

Komentarz do programu:

Wiersze:

1,2:	dyrektywa kompilatora dołączająca pliki nagłówkowe `stdio.h, stdlib.h`
4,5:	prototypy funkcji zapisu i odczytu z pliku
7:	nagłówek funkcji głównej `main`
9:	deklaracje wskaźnika plikowego
11:	wywołanie funkcji zapisu do pliku
12:	weryfikacja czy zapis sie udał
13:	wywołanie funkcji odczytu z pliku
14:	weryfikacja czy odczyt sie udał
15:	wywołanie funkcji `system` – zatrzymanie ekranu
19-36:	definicja funkcji zapisu nazwisk do pliku
21:	pobranie rozmiaru struktury student do zmiennej
23,24:	weryfikacja otwarcia pliku w trybie binarnym dopisywania na końcu
29,31:	wczytanie pól struktury
32:	zapis struktury do pliku studenci
34,50:	weryfikacja operacji zamknięcia pliku
38-53:	definicja funkcji odczytu nazwisk z pliku
42-43:	weryfikacja otwarcia pliku binarnego w trybie odczytu
45:	odczyt z pliku struktury

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 14.1

Płaca pracownika w pliku tekstowym

(*) Oblicz płacę pracownika fizycznego i zapisz ją do pliku tekstowego. Zadeklaruj strukturę o polach: imię nazwisko, liczba godzin, stawka, premia w % i do wypłaty (pole wyliczane). Napisz funkcje zapisu i odczytu z pliku. Wywołaj te funkcje kilkakrotnie. Odczytaj plik.

Zadanie 14.2

Płaca pracownika w pliku binarnym

(*) Oblicz płacę pracownika fizycznego i zapisz ją do pliku binarnego. Zadeklaruj strukturę o polach: imię nazwisko, liczba godzin, stawka, premia w % i do wypłaty (pole wyliczane). Napisz funkcje zapisu i odczytu z pliku. Wywołaj te funkcje kilkakrotnie. Odczytaj plik. Wyświetl dane pracowników, których kwota do wypłaty przekracza podaną wartość.

Zadanie 14.3

Baza danych książek

(**) Załóż plik, będący prostą kartotekową bazą danych książek. Zadeklaruj strukturę opisującą pozycję bibliograficzną. Napisz funkcje zapisu i odczytu z pliku. Zapis pozycji (liczba zapisów nie jest określona) zakończ umownym znakiem (np. * zamiast nazwiska autora). Napisz funkcję wyświetlającą tytuły książek podanego autora.

Zadanie 14.4

Pomiary temperatur w pliku

(*) Załóż plik z pomiarami temperatur. Napisz funkcje zapisu i odczytu z pliku. Wywołaj te funkcje. Odczytaj plik. Oblicz średnią arytmetyczną z pomiarów przechowywanych w pliku.

Zadanie 14.5

Dostęp swobodny do elementu pliku

(*) Załóż plik z N wylosowanymi liczbami całkowitymi. Napisz funkcje zapisu i odczytu z pliku. Wywołaj te funkcje. Odczytaj plik. Wyświetl element pliku na podanej pozycji

Wskazówka: Wykorzystaj funkcję fseek()

Zadanie 14.6

Zawody sportowe

(*) Załóż plik z wynikami zawodów sportowych. Nazwa pliku to nazwa konkurencji. Zawartość pliku: imię, nazwisko i wynik zawodnika. Napisz funkcje zapisu i odczytu z pliku. Wywołaj te funkcje. Odczytaj plik. Wyświetl trzy najlepsze wyniki i dane zawodników, którzy je otrzymali.

Zadanie 14.7

Eksport towarów

(*) Załóż plik z danymi o eksportowanych za granicę towarach zawierający dane: nazwa towaru, kraj eksportu i wielkość eksportu w sztukach. Wyświetlić listę krajów, które eksportują podany towar i podać ogólną wielkość importu.

Zadanie 14.8

Kopiowanie plików

(*) Napisz program kopiujący co trzeci znak z jednego pliku do drugiego. Wyświetl zawartość obydwu plików.

15. Dyrektywy preprocesora

Cel rozdziału

Zapoznanie z podstawowymi poleceniami preprocesora. Nabycie praktycznych umiejętności zastosowania dyrektyw w programowaniu w języku C.

Podstawy teoretyczne

Preprocesor jest wyspecjalizowanym edytorem tekstowym służącym do definiowania makr, które mogą być stałymi lub funkcjami. Może być stosowany niezależnie od kompilatora programów. Preprocesor wykonuje swoje operacje przed zadziałaniem kompilatora.

Dyrektywa preprocesora:

```
#nazwa_dyrektywy nazwa_obiektu
```

PAMIĘTAJ ! *# – znak hash musi poprzedzać dyrektywę, zakończeniem jest znak nowego wiersza (Enter). Kontynuowanie dyrektywy w nowym wierszu wymaga wprowadzenia na końcu linii znaku '\'.*

Przykłady najczęściej stosowanych dyrektyw.

```
#include        #define        #undef
#if             #elif          #endif
#ifdef          #ifend         #line
```

Dyrektywa #include – dołączanie.

Pozwala włączyć wskazany plik do postaci źródłowej programu.

```
#include <math.h>
#include "herfile.h"
```

Zastosowanie nawiasów ostrych < > powoduje, że plik poszukiwany jest w systemowej bibliotece – w folderze znanym kompilatorowi. Gdy nazwa pliku ujęta jest w cudzysłów **" "** to plik poszukiwany jest najpierw w bieżącym folderze, a następnie w folderze systemowym.

Dyrektywy `#define` – makrodefinicja
`#undef` – odwoływanie

Dyrektywa **`#define`** służy do definiowania stałych i makr – funkcji o kodzie wstawianym bezpośrednio w tekst programu. Definiowane stałe i makra pisane są dużymi literami, aby łatwiej odszukać miejsca makropodstawienia. Dyrektywa **`#undef`** pozwala na odwoływanie wcześniej zdefiniowanych stałych i makr.

```
# define STAW
# undef NOWY
```

Przykład 15.1

Stała **FORMAT** *została zdefiniowana w pliku* **herfile.h**

```
…
#include "herfile.h"
main ()
{
float waga;
waga = 46.7;
printf (FORMAT, waga);
exit(0);
}
```

***Zawartość pliku* herfile.h**

```
#define FORMAT "Jej waga to %6.2f kg\n"
```

W przyłączanym pliku przy użyciu dyrektywy **`#define`** zdefiniowano makro zawierające sposób wypisania wartości zmiennej **waga**.

WYKONANIE PROGRAMU

Na ekranie będzie wyświetlony następujący wydruk.

```
Jej waga to  46.70 kg
Press any key to continue . . .
```

Przykład 15.2

Zdefiniowanie makra typu funkcyjnego z argumentami

Definicja makra jako operatora ternarnego

```
...
#define MIN(k,l) ((k)<(l)?(k):(l))
main ()
{
int a, b;
a = 12; b = 7;
printf ("Mniejsza zmienna to %d\n", MIN(a,b));
exit(0);
}
```

Preprocesor wykona makropodstawienia przekazując kompilatorowi następującą postać

```
printf ("Mniejsza zmienna to %d\n",((k)<(l)?
       (k):(l)));
```

WYKONANIE PROGRAMU

Na ekranie będzie wyświetlony następująca wydruk.

```
Mniejsza zmienna to 7
```

PAMIĘTAJ ! Podstawienie stałych nie jest dokonywane wewnątrz łańcuchów znakowych, czyli tekstów umieszczonych między cudzysłowem.

Kompilacja warunkowa

Dyrektywy:

`#if, #elif, #endif #else, #ifdef, #ifndef`

pozwalają na realizację operacji pomijania lub uwzględniania sekwencji kodu źródłowego programu.

```
#if wyrażenie_relacyjne_1
        sekwencja_instrukcji_1
#elif wyrażenie_relacyjne_2
        sekwencja_instrukcji_2
#else
        sekwencja_instrukcji_3
#endif
```

Jeżeli **`wyrażenie_relacyjne_1`** ma wartość „prawda” wtedy w kodzie źródłowym programu przekazywana jest do kompilacji **`sekwencja_instrukcji_1`**. W przeciwnym przypadku, gdy **`wyrażenie_relacyjne_2`** jest „prawdą” to do kompilacji

przechodzi kod źródłowy ze zbiorem instrukcji **sekwencja_instrukcji_2**. W przeciwnym przypadku **sekwencja_instrukcji_3** znajdzie się w kodzie przekazanym kompilatorowi.

Przykład 15.3

definicja stałej **ROZMIAR**

sprawdzenie wartości wyrażenia relacyjnego

```
...
#define  ROZMIAR 333
int main()
{  int a;
#if ROZMIAR<=111
    int tab[ROZMIAR];
    printf ("Wersja int\n");
#else
    float tab[ROZMIAR];
    printf ("Wersja float\n");
#endif
return 0);
}
```

WYKONANIE PROGRAMU

W sytuacji gdy przez dyrektywę **#define** *stałej* **ROZMIAR** *nadano wartość* 333 *wyrażenie relacyjne ma wartość logiczną „fałsz" stąd wyprowadzony zostanie następujący tekst.*

```
Wersja float
```

Predefiniowane makra systemowe

__func__	–	**predefiniowany identyfikator przekazuje informację w jakiej funkcji się znajduje,**
__DATE__	–	**data w chwili kompilacji,**
__TIME__	–	**godzina w chwili kompilacji,**
__FILE__	–	**łańcuch, który zawiera nazwę pliku kompilowanego,**
__LINE__	–	**definiuje numer linijki.**

Przykład 15.4

Wykorzystanie makr systemowych

```
...
int main()
{
  printf("Wew.funkcji: %s\n", __func__);
  printf("       data: %s\n", __DATE__);
  printf("       czas: %s\n", __TIME__);
```

```
  printf("      linia: %d\n", __LINE__);
  printf("       file: %s\n", __FILE__);
return 0;
}
```

WYKONANIE PROGRAMU

```
Wew.funkcji: main
       data: Sep 21 2015
       czas: 22:56:27
      linia: 13
       file: D:\0 Jurek-WEiI-IInf\Dydaktyka\Jŕzyk C\Podrŕcznik\rozdzia| 15\p15-4.c
```

Arkusz przykładowych ćwiczeń

Ćwiczenie 15.1

Definiowanie rozmiaru tablicy dwuwymiarowej

Przekształć program tak aby rozmiar tablicy był zdefiniowany przez dyrektywy preprocesora.

Pierwotna wersja programu

Zdefiniowanie zmiennych typu `int` `rows` *oraz* `cols` *i nadanie im wartości*

```
#include <stdio.h>
int main() {
  int rows=3, cols=4;
  int array[rows][cols];
  int ix, iy;
  for (ix=0; ix<cols; ix++)
      for (iy=0; iy<rows; iy++)
      array[iy][ix] = (iy*4) +ix +1;
printf("Drukowanie - wiersz po wierszu\n");
  for(ix=0; ix<rows; ix++)
  {printf ("wiersz nr %d ma elementy:\n", ix);
      for(iy=0; iy<cols; iy++)
      printf ("%d   ", array[ix][iy]);
      printf("\n");
  }
return 0; }
```

WYKONANIE PROGRAMU

```
Drukowanie wiersz po wierszu
wiersz nr 0 ma elementy:
1   2   3   4
wiersz nr 1 ma elementy:
5   6   7   8
wiersz nr 2 ma elementy:
9   10   11   12
```

ROZWIĄZANIE

Należy zastosować dyrektywę **`#define`**.

Zastosowanie dyrektywy `#define`

```
#include <stdio.h>
#define     ROWS 3
#define     COLS 4
int main() {
  int array[ROWS][COLS];
  int ix, iy;
  for (ix=0; ix<COLS; ix++)
      for (iy=0; iy<ROWS; iy++)
      array[iy][ix] = (iy*4) +ix +1;
printf("Drukowanie wiersz po wierszu\n");
  for(ix=0; ix<ROWS; ix++)
  {printf ("wiersz nr %d ma elementy:\n", ix);
      for(iy=0; iy<COLS; iy++)
      printf ("%d   ", array[ix][iy]);
      printf("\n");
  }
return 0; }
```

Arkusz zadań do wykonania

Liczba gwiazdek oznacza stopień trudności zadania.

Zadanie 15.1

Obliczanie pola powierzchni koła i obwodu okręgu

(*) Napisz funkcję obliczającą pole powierzchni koła i obwodu okręgu definiując liczbę Pi w preprocesorze.

Zadanie 15.2

Makra SQUARE i ABS

(*) Zdefiniuj marko funkcyjne SQUARE(x) pozwalające na podnoszenie argumentu do kwadratu i makro ABS(x) zwracające wartość bezwzględną. Sprawdź działanie makr w programie dla argumentów:
x, x+1, 2*x, gdzie x =5 lub -5

Zadanie 15.3

Wyszukiwanie największej liczby

(**) Napisz funkcję zwracającą największą liczbę spośród N różnych liczb podanych przez użytkownika. Wczytaj liczbę N, zdefiniuj makro wykorzystujące operator ternarny. Wyświetl wynik.

Zadanie 15.4

Instrukcje języka Pascal zapisane za pomocą dyrektyw preprocesora

(**) Uzupełnij program definiując odpowiednie dyrektywy preprocesora (makra) pozwalające zamienić instrukcje w języku Pascal na odpowiednie instrukcje w języku C.

```
#include <stdio.h>
#include <stdlib.h>

.......//tu wstaw dyrektywy
```

```
int main(int argc, char *argv[])
begin
      int a,b;
      write("Podaj a ");//wyprowadzenie tekstu
      readln(a);  //wczytanie zmiennej
      write("Podaj b ");
      readln(b);
      write("Podales: ");
      writeln(a);       //wyprowadzenie zmiennej
      writeln(b);
      if (a>b) then begin
                    write("a wieksze od b ");
                    write("--> Kwadrat a= ");
                     writeln(sqr(a)); //a*a
               end;
      if (a<b) then write("b wieksze od a");
      if (a==b) then write(" a i b takie same");

      return 0;
end
```

Wskazówki:

skorzystaj z pseukokodu ze stron 9-10,
w języku C używany ***printf()*** *zamiast* ***write*** *oraz* ***scanf( )*** *zamiast* ***read****,*
zwróć uwagę na poprawność składni,
zastosować dyrektywę ***#define****.*

16. Standardowa biblioteka ANSI C

Cel rozdziału

Zapoznanie z podstawowymi bibliotekami języka C.

Podstawy teoretyczne

Twórcy języka C przyjęli zasadę aby nie tworzyć słów kluczowych pozwalających obsługiwać wejścia i wyjścia. W języku C brakuje również wielu innych słów, które pozwalałyby wykonywać wiele konkretnych działań, np. modyfikować zmienne łańcuchowe, czy obliczać wartość bezwzględną zmiennej. Prowadzone prace nad opracowaniem standardu języka C (ANSI C) dotyczyły przygotowania zestawu instrukcji, które będą identyczne w każdej implementacji C. Opracowane instrukcje tworzą *Bibliotekę Standardową*, która pozwala na wykonanie wielu różnych zadań i działań, ale nie zawiera żadnych funkcji zależnych od sprzętu lub systemu operacyjnego. Biblioteka ANSI C dzieli funkcje na kilka grup – bibliotek, z których każda posiada osobny plik nagłówkowy.

Biblioteki do programu są przyłączane przy zastosowaniu dyrektywy **#include.** Po niej następuje nazwa biblioteki zamknięta w nawiasach **< >**, co powoduje, że biblioteka poszukiwana jest przez preprocesor w katalogu zawierającym systemowe pliki dołączane (pliki dołączane przez kompilator C). Gdy zastosowano znaki **" "** (stosowane do plików nagłówkowych definiowanych przez użytkownika) to preprocesor poszukuje dołączanego pliku w pierwszej kolejności w bieżącym katalogu, a później w katalogu systemowym.

```
#include <stdio.h>
#include "funkcje.h"
```

Standardowa biblioteka ANSI C:

NAZWA	SKRÓCONY OPIS
time.h	***Data i czas***. Zawiera funkcje obsługujące czas, dotyczące działań na zmiennych wyrażających datę, dzień miesiąca, tygodnia, czas pracy procesora, np. zwraca liczbę sekund, które upłynęły od dnia 1.01.1970, 00:00:00 GMT (tzw. epoka uniksowa).
string.h	***Obsługa łańcuchów.*** Zawiera funkcje umożliwiające przeprowadzania operacji na zmiennych znakowych i łańcuchowych, np. porównuje, kopiuje czy łączy łańcuchy.
stdlib.h	***Funkcje ogólnego użytku.*** Biblioteka zawiera najbardziej podstawowe funkcje, jest bardzo różnorodna, od funkcji przydzielających pamięć dynamiczną – ***malloc***, przekształcenie łańcucha na wartość całkowitą – ***atoi***, czy generator liczb pseudolosowych – ***rand***, czy obliczanie wartości bezwzględnej – ***labs***.
stdio.h	***Standardowa biblioteka wejścia-wyjścia***. Biblioteka przede wszystkim zawiera różne funkcje pozwalające na wprowadzenie danych do programu: ***getc***, ***getchar***, ***scanf***, czy ich wyprowadzenie: ***fprint***, ***puts***.
stdarg.h	***Zmienna liczba argumentów.*** Zawiera narzędzia dla funkcji ze zmienną liczbą argumentów, np.: ***va_arg***, ***va_start***.
signal.h	***Obsługa sygnałów***. Biblioteka zawiera funkcje dotyczące obsługi sygnałów.
setjmp.h	***Skoki nielokalne.*** Zawiera funkcje pozwalające na wykonywanie nielokalnych skoków przydatnych w obsłudze błędów.
math.h	***Biblioteka matematyczna.*** Biblioteka zawiera funkcje pozwalające obliczać wartości różnych operacji matematycznych, np.: ***cosh*** – obliczanie cosinusa hiperbolicznego, ***hypot*** – pierwiastek kwadratowy z sumy kwadratów argumentów. Funkcje mogą być jednoargumentowe oraz dwuargumentowe.
locale.h	***Lokalizacja***. Zawiera funkcję pobierającą informacje o formatowaniu liczb oraz ustawia bieżące właściwości regionalne.
ctype.h	***Obsługa znaków***. Biblioteka zawiera funkcje pozwalające sprawdzać czy dany znak jest np.: małą literą – ***islower***, czy cyfrą szesnastkową – ***isxdigit***.
assert.h	***Diagnostyka***. Biblioteka zawiera funkcję pozwalającą na przerwanie pracy programu jeśli wyrażenie jest fałszywe (makro).

17. Список обраної термінології
Zestaw wybranej terminologii
List of selected terminology

Standard C89

zawiera 32 słowa kluczowe / has 32 keywords / містить 32 ключових слова

	Słowa kluczowe	Keywords	Ключові слова
auto	*zmienna, lokalna*	local variable	локальна змінна
break	*instrukcja sterująca, skok*	jumping	інструкція управління, стрибок <перехід>
case	*instrukcja sterująca, wybór*	selection	інструкція управління, вибір
char	*zmienna, znakowa, łańcuchowa*	character	змінна, знакова, ланцюгова
const	*modyfikator, definiuje tzw. stałe*	modifier, unmodifiable variable	модифікатор, так звана постійна нередагована змінна
continue	*instrukcja sterująca, skok na początek pętli*	passes control to the beginning of the loop	інструкція управління, перехід на вершину циклу передає управління до початку циклу
default	*instrukcja sterująca, domyślne*	default	інструкція управління, автоматично <тобто за замовчуванням, або за відсутності, інших> дефолт
do	*instrukcja sterująca, pętla*	loop	інструкція управління, петля
double	*zmienna , podwójna precyzja*	variable double-precision	змінна подвійної точності
else	*instrukcja sterująca, inaczej, w innym wypadku*	conditional statement	інструкція управління: інакше, в іншому випадку
enum	*zmienna, typ złożony, stałe całkowite*	groups of variables, type int	змінна, складний тип, ціле постійне (тобто типу ціле число)
extern	*zmienna, modyfikator*		змінна, модифікатор
float	*zmienna, zmiennoprzecinkowa*	floating point variables	змінні з плаваючою комою
for	*instrukcja sterująca, pętla*	loop	петля
goto	*instrukcja sterująca, skok bezwarunkowy*	unconditional transfer control	інструкція управління, безумовний перехід
if	*instrukcja sterująca, instrukcja warunkowa*	conditional statement	інструкція управління, умовна інструкція

int	*zmienna, całkowita*	integer variable	змінна типу ціле число
long	*zmienna, modyfikator, całkowita długa*	modifier variable type, long integer	змінна, модифікатор, довге ціле число
register	*zmienna, deklaracja w rejestrze procesora*	variables, being declared in a CPU register	змінні, які оголошені в реєстрі процесора
return	*wyjście z funkcji*	exits the function	процедура, вихід з функції
short	*zmienna, modyfikator, całkowita krótka*	modifier variable type integer	змінна, модифікатор, коротке ціле число
signed	*zmienna, modyfikator, ze znakiem*	modifier variable, type signed	змінна, модифікатор, типу зі знаком
sizeof	*zmienna, określa rozmiar w bajtach*	returns the size in bytes	змінна, повертає розмір в байтах
static	*zmienna, modyfikator, nie jest usuwana po zakończeniu funkcji*	modifier, is not removed at the end of the function	модифікатор, не видаляється в кінці функції
struct	*zmienna, typ złożony, heterogeniczne*	group variables, heterogeneous	змінна, групи змінних, гетерогенна
switch	*instrukcja sterująca, przełącznik*	switch	інструкція управління, перемикач
typedef	*zmienna, definiuje nową nazwę istniejącemu typowi*	defines a new name for an existing type	змінна, визначає нове ім'я для існуючого типу
union	*zmienna, typ złożony, dzielą tą samą przestrzeń pamięci*	group variables, share the same storage space	група змінних, що поділяють той же простір пам'яті комп'ютера
unsigned	*zmienna, modyfikator, bez znaku*	modifier variable, type unsigned	змінні, модифікатор, типу без знаку
void	*zmienna, typ pusty*	empty data type	порожній тип даних
volatile	*zmienna, zmieniana w tle*	variable can be changed by a background routine	змінна може бути змінена за допомогою фонової процедури
while	*instrukcja sterująca, pętla*	**while** loop	інструкція управління, петля, типу ‘в той час як’

Standard C99

dodaje następujące słowa/adds the following terms/додає наступні слова

_Bool	*zmienna, logiczna*	variable, logical	змінна, логічна
_complex	*zmienna, zespolona*	variable, complex	змінна, комплексова
_imaginary	*zmienna, część urojona*	variable, the imaginary part	змінна, уявна частина
inline	*modyfikator, wywołanie funkcji*	modifier, function call	модифікатор, виклик функції
restrict	*modyfikator do wskaźnika*	modifier to pointer	модифікатор до вказівника

	Operatory	**Operators**	**Оператори**
==	operator sprawdzający równość (pamiętaj dwa znaki ”=”)	operator checking equality (remember the two characters "=");	оператор перевірки рівності (пам'ятайте, два символи "=");
<	czy pierwszy mniejszy od drugiego;	whether the first is smaller than the second;	чи перший менший, ніж другий;
>	czy pierwszy większy od drugiego;	the first is greater than the second;	чи перший більший другого;
<=	czy pierwszy mniejszy lub równy od drugiego;	is less than or equal to the first of the other;	чи перший менший або дорівнює другому;
>=	czy pierwszy większy lub równy od drugiego;	the first more or equal to the second	чи перший більший або дорівнює другому;
!=	czy pierwszy nierówny (różny) od drugiego	the first different to the second	чи перший відрізняється від другого

	Znaki specjalne	**Special signs**	**Спеціальні символи**
\t	*znak tabulacji*	a tab character	табуляція
\n	*powoduje przejście do nowej linii,*	moves to a new line	переходить на новий рядок

	Kody konwersji	**Conversion codes**	**Коди перетворення**
%d	calkowitoliczbowe	integer	ціле число
%f	zmiennoprzecinkowe	float	з плаваючою комою
%s	napisowe	string	строка
%o	oktnalne	octal	вісімковий
%x	heksagonalne	hexagonal	гексагональний

Słownictwo programistyczne	**Programming vocabulary**	**Словник програмування**
& – ampersand	ampersand	амперсент
argument	argument	аргумент
ciało funkcji	function body	тіло функції
funkcja zwraca pewną wartość	function returns a value	функція повертає деяке значення
if (warunek) instrukcja	**if** (condition) statement	**if** (умова) оператор
inicjalizacja tablic	table initiation	заснування таблиць
instrukcja warunkowa	conditional instruction	умовна інструкція
instrukcja warunkowa **if-else**	conditional statement **if-else**	умовний оператор **if-else**
instrukcja warunkowa **if-else-if**	conditional statement **if-else-if**	умовний оператор **if-else-if**
jawne rzutowanie typu	explicit type casting	явне кидання типів
komentarz	commentary	коментар
koniec funkcji	end of the function	кінець функції
konwersja plików	file conversion	перетворення файлів

lista argumentów	list of arguments	список аргументів
moduł	module	модуль
nazwa_funkcji	function_name	імя_функції
niejawne rzutowanie typu	implicit type casting	неявне кидання типів
operatory arytmetyczne	arithmetical operators	арифметичні оператори
operatory relacyjne	relational operators	реляційні оператори
pętla **do-while**;	loop **do-while**	петля **do-while**
pętla **for**;	**for** loop	петля **for**;
pętla **while**;	**while** loop	петля **while**;
przełącznik **switch** i wybór **case**	switch **switch** and **case** selection	вимикач **switch** і вибір **case**
stałe napisowe	string constants	строковий постійні
stałe znakowe	character constants	символьна постійна
tablice jednowymiarowe	unidimensional tables	одновимірні таблиці
tablice o zmiennych rozmiarach	tables of variable dimensions	таблиці змінних розмірів
tablice wielowymiarowe	multidimensional tables	багатовимірні таблиці
typ zwracany	return type	типу повернення
zmienne globalne	global variables	глобальні змінні

Zakończenie/Conclusion/Закінчення

Autorzy podręcznika są w pełni świadomi, że przedstawiony zakres materiału jest subiektywnym wyborem i nie wyczerpuje w żadnym wypadku wszystkich możliwości zastosowania języka C. Z uwagi na fakt, że studenci nie mieli wcześniej zagadnień z zakresu metod numerycznych oraz algorytmów, prezentowane ćwiczenia i zadania musiały zostać zawężone do przykładów, które nie wymagały specjalistycznej wiedzy.

Autorzy położyli szczególny nacisk na stronę praktyczną omówionych poleceń języka C, mając na uwadze stworzenie warunków zapewniających użytkownikowi poczucie swobody podczas tworzenia własnych programów.

The authors of the manual are fully aware that the present range of material is a subjective choice and does not in any case exhaust all the possibilities of using the C language. Due to the fact that students have no prior experience of numerical methods and algorithms, the exercises and tasks presented here had to be limited to examples which did not require specialist knowledge.

The authors put special emphasis on the practical side the C language commands discussed, keeping in mind the creation of conditions ensuring for the user a feeling of freedom when creating their own programs.

Автори підручника повною мірою усвідомлюють, що нинішній асортимент матеріалу є суб'єктивним вибором і в будь-якому випадку не вичерпує всі можливості використання мови С. У зв'язку з тим, що студенти не мають попереднього досвіду чисельних методів і алгоритмів. Представлені вправи і завдання обмежується прикладами, які не вимагають спеціальних знань.

Автори приділили більше уваги на практичну сторону представлених команд мови С, маючи на увазі створення умов для забезпечення користувачу почуття свободи при написанні власних програм.

Literatura

1. Kernighan B. W., Ritchie D.: *Język C*, wydaw. WNT 1987.
2. King. K. N.: Język C. *Nowoczesne programowanie*, wydaw. Helion, Gliwice 2011.
3. Kwiatkowska A., Łukasik E.: *Schematy zwarte NS*, wydaw. MIKOM, Warszawa 2004.
4. Prata S.: *Język C. Szkoła programowania*, wydaw. Robomatic, Wrocław 1999.
5. Schildt H.: *Programowanie C*, wydaw. RM, Warszawa 2002.
6. Stabrowski M. M.: *Język C w przykładach*, wydaw. WSEI, Lublin 2011.