Język C i C++– podstawy

Krótka charakterystyka języków C i C++,
1) środowisko, struktura programu, Środowiska programistyczne.
Pierwszy program. Kompilacja BCC32.
Turbo C. Słowa kluczowe. Instrukcje podstawowe.
Typy danych. Stałe, zmienne, deklaracje, wyrażenia.
Wejście i wyjście programu – funkcje. Operatory i wyrażenia.
Instrukcje warunkowe if, instrukcja wyboru switch, instrukcje cyklu (pętle while, do while, for):

Cele nauczania: Podstawowe zasady programowania.

Osiągnięcia: Praktyczna znajomość elementów programowania w języku C/C++

Krótka charakterystyka języka programowania C

- **Języki C i C++ - uniwersalne, do celów ogólnych.**
- W językach tych powstają aplikacje dla różnych systemów operacyjnych, w tym m.in. dla Windows, Unix, Linux, MacOS, BeoS.
- Język C został zdefiniowany w 1972 r. przez **Denisa M Ritchie** z Bell Laboratories w New Jersey.
- W 1988 r. powstało opracowanie tzw. standardu **ANSI** lub "ANSI C".
- Język C jest językiem ogólnego zastosowania.
  Język w miarę prostoty, niezbyt obszerny, szerokiego zastosowania.
  Charakteryzuje się także nowoczesnymi strukturami danych i bogatym zestawem operatorów.

- Język C pierwotnie miał ułatwiać tworzenie oprogramowania systemowego - UNIX.
  UNIX z definicji wyposażony w kompilator C.
- Język C można zakwalifikować jako język wysokiego poziomu (1:10 - źródło wynik).
- Można również wykonać operacje zastrzeżone dla niskiego poziomu - elementy programowania niskiego poziomu.
- Możliwość operowania adresami.
  Pozwala to wyeliminować wstawki w języku Asemblera.

Co potrzebne do programowania w C, C++

- Do programowania w C potrzebne są
  - komputer z SO Windows lub Linux,
  - **kompilator języka C**,
  - **linker** (zwykle jest z kompilatorem),
  - przydatny tez **debuger**,
  - **edytor tekstowy.**

Zintegrowane środowiska programistyczne

Do tworzenia programów można wykorzystywać **zintegrowane środowiska programistyczne** jak:

- firmy **Borland**:
  Turbo C, Borland C, Borland C++ **Builder**
- **Microsoft Visual C++,**
- **Dev C++,**
- **CodeBlocks**
- **KDevelop** (Linux) do KDE,
- **Eclipse**

Pisanie kodu programu w C (.C) lub C++ (.CPP)

- Do edycji pliku programu (rozszerzenie **C** lub **CPP**) można użyć dowolnego **edytora tekstowego** zapisującego pliki w postaci czystego kodu ASCII, typu

  Notatnik (Notepad.exe) ,

  Edit

  lub lepiej

  Notepad ++ czy

  ConTEXT,

  które pozwalają sprawdzać składnię i podkolorowują odpowiednio tekst po wybraniu języka.

Kompilacja programów przy pomocy BCC32

- Do **kompilacji** można użyć środowiska programistycznego **Borland C++ Compiler 5.5 – program BCC32.exe.** Kompilator ten jest darmowy, generuje 32-bitowy kod dla Windows, zawiera tylko niezbędne narzędzia uruchamiane z linii poleceń, jest prosty w instalacji i konfiguracji.
- Najlepiej zainstalować w **C:\BCC55**.
  W katalogu tym są podkatalogi: BIN, INCLUDE, LIB, HELP, EXAMPLES.
- Pliki konfiguracyjne w katalogu C:\BCC55\BIN:
  BCC32.cfg i **ILINK32.cfg**

Zawartość plików konfiguracyjncych:

Plik **BCC32.cfg** *-I"c:\Bcc55\include -L"c:\Bcc55\lib„*

Plik **ILINK32.cfg** - składa się z jednej linii: *-L"c:\Bcc55\lib"*

- **Kompilacja** programów przy pomocy BCC32.EXE:
  Bcc32 plik_programu.c lub bcc32 plik_programu.cpp
  – w linii poleceń
  *po skopiowaniu pliku programu do katalogu **C:\BCC55\BIN** lub z dowolnego miejsca na dysku po dopisaniu do zmiennej PATH w* wierszu *poleceń*:
  PATH=%PATH%;C:\BCC55\BIN

Najprostszy wykonywalny program C:

Plik program1.c

```
int main(void)
  {
    return 0;
  }
```

- Kompilacja: BCC32 program1.c

Analiza programu program1.c

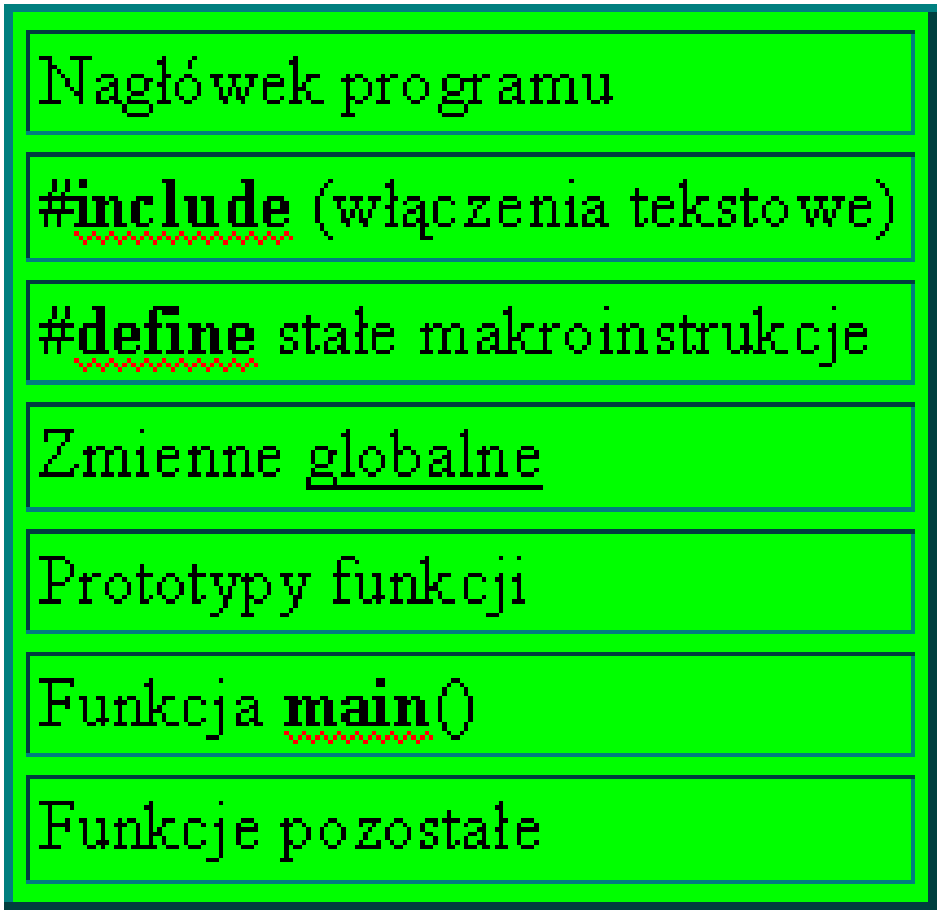
- Program ten składa się tylko z bezparametrowej funkcji głównej int **main(void**), inaczej int **main(),**
  której wartość jest typu całkowitego (int).
- Ciało funkcji zbudowane jest z **instrukcji**, które powinny znaleźć się w bloku wyznaczonym nawiasami kwadratowymi.
- Funkcja ta zwraca wartość 0 za pomocą instrukcji **return**.
- Skompiluje się również program w postaci:

  main() { }

  void main() { }

  void main(void) { }

Ogólna struktura programu w C/C++



Ogólna budowa programu w języku C

- Cechą języka C/C+ jest możliwość budowy programu z wielu modułów.
- **Modułem** może być każdy zbiór zawierający poprawny kod źródłowy.
- Program w C zbudowany jest z **funkcji**.
- Każda z funkcji może posiadać parametry oraz określony typ wartości.
- Aby można było wygenerować kod wynikowy programu (w DOS zbiór .EXE), w jednym i tylko w jednym z modułów musi się znaleźć **funkcja główna main(),** od której rozpocznie się wykonywanie programu.
- Moduł zawierający funkcję **main()** nazywa się modułem głównym.

Pierwszy przykładowy program hello.c

```
/* Komentarz: plik hello.c */
#include <stdio.h> /* komentarz wielowierszowy –
    dołączenie pliku nagłówkowego stdio.h potrzebnego do użycia funkcji printf */
int main (void) /* funkcja główna*/
{ /* początek */
printf ("Hello World!"); /* funkcja printf() wyświetla napis */
return 0; /* funkcja main() zwraca 0 */
} /* koniec funkcji głównej */
```

Kompilacja: ***bcc32 hello.c*** – utworzy się plik ***hello.exe***

Skierowanie wyników do pliku: hello > hello.txt

TYPE hello.txt – treść pliku

Wersja programu hello w C++ - plik hello.cpp

```
//Program   hello.cpp  - komentarz jednowierszowy w C++
#include <iostream.h> // preprocessor - dołączenie biblioteki
    //systemowej do funkcji cout – input ootput stream
int main()        // funkcja główna
{
cout << "Hello World!"; // wyświetlenie napisu
return 0;         // funkcja główna zwraca 0
}
```

Ogólna postać programu w języku C

/* **Nagłówek programu** - komentarz: (nazwa, autor, data, kompilator, uwagi itp.) */

#include (**preprocesor** - włączenia tekstowe), np. *#include <stdio.h>*

#define (**preprocesor** - stałe makroinstrukcje), np. *#define PI 3.14*

Zmienne globalne (przed main), np. *float liczb1;*

Prototypy funkcji - deklaracje, np.

int dodawanie(float a, float b);

Funkcja main() *{ treść_funkcji }*

Funkcje (definicja funkcji), np.

int dodawanie(float a, float b) {return a+b; }

Objaśnienia – nagłówek, #include, #define, zmienne globalne, funkcje

- W **nagłówku**: nazwa programu, nazwa kompilatora, uwagi odnośnie kompilowania, linkowania, wykonania
- Sekcja **#include** <plik> zawiera (#include <program.h> oraz własnych plików specyfikację włączanych plików bibliotecznych"program.h"
- Sekcja **#define** zawiera definicje stałych i makroinstrukcji
- Następne sekcje to **zmienne globalne** oraz **prototypy funkcji**. Muszą wystąpić przed ciałami funkcji, aby w dalszej części programu nie było odwołań do obiektów nieznanych kompilatorowi. Jeżeli funkcje pozostałe umieszczone byłyby przed funkcją **main()** to specyfikowanie prototypów byłoby zbyteczne.
- *Włączenia tekstowe **#include** mogą w zasadzie wystąpić w dowolnym miejscu programu. Zaleca się aby dłuższe funkcje lub grupy funkcji były umieszczane w osobnych plikach.*
- **Funkcja main() musi być w treści programu!**

Funkcja main()

- W programie definiujemy główną funkcję **main()**, uruchamianą przy starcie programu, zawierającą właściwy kod.
- Definicja funkcji zawiera, oprócz nazwy i kodu, także typ wartości zwracanej i argumentów pobieranych.
- Typem zwracany przez funkcję jest ***int*** (*Integer*),
  czyli liczba całkowita (w przypadku ***main()*** będzie to kod wyjściowy programu).
  W nawiasach umieszczane są *argumenty* funkcji, tutaj zapis ***void*** oznacza ich pominięcie.
  Funkcja ***main()*** jako argumenty może pobierać parametry linii poleceń, z jakimi program został uruchomiony, i pełną ścieżkę do katalogu z programem.
- Kod funkcji umieszcza się w nawiasach klamrowych ***{*** i ***}***.
- Wszystkie polecenia kończymy średnikiem.
- ***return;*** określa wartość jaką zwróci funkcja (tu program).
  Liczba zero zwracana przez funkcję ***main()*** oznacza, że program zakończył się bez błędów;
  błędne zakończenie często (choć nie zawsze) określane jest przez liczbę jeden.
- Funkcję ***main()*** kończymy nawiasem klamrowym zamykającym.

Środowiska programistycznego Borlanda

- **Turbo C** lub **Borland C** to zintegrowane pakiety składające się z następujących części:
  - **edytora** - służy on do pisania i poprawiania programów;
  - **kompilatora** - zamienia on plik źródłowy na instrukcje wykonywane przez komputer;
  - **debuggera** - umiejscawia i usuwa on usterki w programie oraz pozwala śledzić wykonywanie programu krok po kroku.

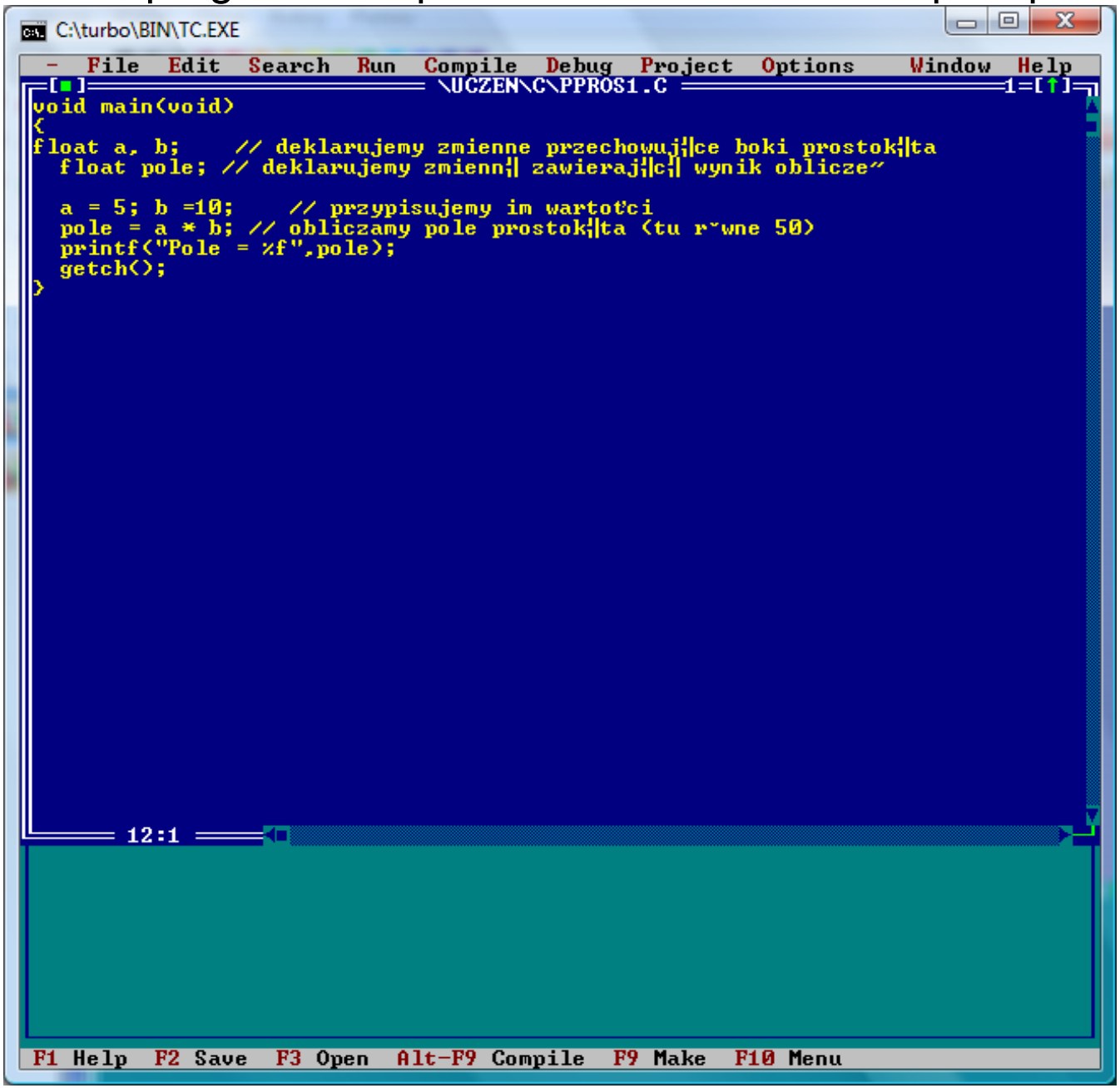
Turbo C

- **Turbo C** – program **TC.EXE**
  (zwykle w katalogu **C:\TC\BIN** lub C:\Turbo\Bin)
  Uruchamiamy **TC.EXE**.
- Otwieramy istniejący plik – **File, Open** – wskazujemy plik
  lub piszemy treść programu i zapisujemy
- Kompilujemy program ALT C, Build lub **Alt F9.**
  Utworzony zostanie plik p1.exe
- Uruchamiamy program ALT R lub **CTRL F9.**
- By zobaczyć wyniki, należy przejść do
  ekranu użytkownika:
  Alt W – Menu Windows, User Screen lub bezpośrednio: **Alt F5**.
- Pomoc w TURBO C:
  Alt H; Np. **ALT H, Index**, wpisujemy szukane słowo

Menu programu TC.EXE

- **File** – **Plik**: opcje: New – Nowy, Open – Otwórz, Save – Zapisz, Save as – Zapisz jako, Save All – zapisz wszystko, Change dir – zmień katalog, DOS Shell – okno DOS (exit – wyjście), Quit – wyjście
- **Edit - edycja**
- **Search**– **szukaj**
- **Run - uruchom**
- **Compile**: **kompiluj**: Compile Alt F9 – kompiluj, Make – utwórz, Link – linkuj, Build All – zbuduj wszystko, Information – informacja, Remove messages – usuń komunikaty (wiadomości)
- **Debug -** debugowanie
- **Project -** projekt
- **Options - opcje**
- **Window -** okno
- **Help – pomoc** → Help, Index, wpisujemy szukane słowo, np. void i naciskamy Enter

Okno TC – program do wpisania i uruchomienia na pole prostokąta



Program w Dev C++

```
/* Plik PolProst2.c */
#include <cstdlib>
#include <iostream>
#define NL printf("\n")
using namespace std;

int main(int argc, char *argv[])
{
   float a, b, pole;
   a=5; b=10;
   pole=a*b;
   printf("Pole=%f\n",pole);
   NL;  /* zdefiniowane w #define */
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Obsługa programu

Dev C++

Plik Nowy Projekt,

Console Application

Kompiluj i uruchom

Wydawanie poleceń, skróty klawiszowe w TC

- **myszą** - poprzez kliknięcie na odpowiednim wyrazie menu, a następnie na odpowiedniej komendzie;
- **klawiaturą** - wciskamy kombinację **Alt** i pierwszą literę wyrazu w menu, następnie wciskamy wyróżnioną literę komendy. Np.
  - **Alt F** – menu File – Plik
  - **Alt E** – menu Edit – edycja
  - **Alt R** – menu Run – Uruchom
  - **Alt C** – menu Compile – Kompilacja
  - **Alt O** – menu Options – Opcje
  - **Alt D** – menu Debug – debugowanie
  - Alt T – menu Tools - Narzedzia
- **klawiszem skrótowym** - wciskamy klawisz lub kombinację klawiszy, która znajduje się obok komendy lub w pasku z klawiszami.

Niektóre **skróty klawiszowe**

- **F2** - zapamiętanie edytowanego programu (z rozszerzeniem .**pas**)
- **F3** – otwarcie programu istniejącego pliku programu (C lub CPP)
- **Alt-F9** - kompilacja programu
- **F9** – *make* – tworzy program wykonywalny, kompiluje program do pliku .EXE kompilując tylko zmienione pliki
- **Ctrl-F9** - uruchomienie skompilowanego programu: (z ewentualną kompilacją)
- **Alt-F5** - przełącza na wirtualny ekran użytkownika
- **Alt-X** - wyjście z systemu TC
- **F4** - *Go to cursor* - wykonuje program aż do linijki zawierającej kursor, po czym zatrzymuje się i czeka na dalsze polecenia;
- **F7** - *Trace Into* - wykonuje bieżącą instrukcję programu, jeśli jest to wywołanie procedury, to przechodzi do pierwszego wiersza tej procedury;
- **F8** - *Step Over* - wykonuje bieżącą instrukcję programu, jeśli jest to wywołanie procedury, to wykonuje ją w całości;

Podstawowe informacje o języku C i C++

- **Język programowania** – zapis algorytmu w postaci zrozumiałej dla człowieka.
- **Program źródłowy** – tekst: zbiór słów, cyfr, znaków – rozszerzenie **C** lub **CPP**
- **Słowa zastrzeżone** - **kluczowe**, m.in. **auto, break, case, char, class, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, private, register, return, short, signed, sizeof, static, struct, switch, try, typedef, unsigned, void, while** (czasem też inne, zależy od kompilatora)
- **Łańcuchy znaków** muszą być ograniczone przy pomocy cudzysłowa, np. **”Ala ma kota”**, **”Podaj nazwisko: ”**. Mogą zawierać spacje.
- **Liczby dziesiętne** mogą zawierać **kropkę** a *nie przecinek*, np. 15.67
  Mogą być zapisane w **postaci wykładniczej**, z literą **E**, np. **2.83E+3** (czyli 2.83*10^3)
- **Każde polecenie programu musi kończyć się średnikiem**.
  W jednej linii może być kilka poleceń ale to nie jest zalecane
- W pisaniu programów rozróżniane są duże i małe litery - np. w main()
- **Komentarze** tworzymy przy pomocy znaków **/*** wielowierszowy ***/** lub **//** jednoliniowy np. **/***To jest komentarz może być wieloliniowy */
  // to komentarz w jednym wierszu

Pojęcia podstawowe

- **Słowo Kluczowe** – zastrzeżone – nie można wykorzystać jako nazwy zmiennych lub stałych
- **Stałe i zmienne** to określone miejsca w pamięci komputera.
  Utworzenie stałej lub zmiennej *to przypisanie do jej identyfikatora określonego adresu pamięci, pod którym znajduje się wartość stałej lub zmiennej*
- **Stała** - jest to pewna wartość przypisana znakowi/wyrazowi której nie można zmienić np.
  #define PRAWDA 1
  const PI= 3.1415; const string NI = ”Nowak Adam”;
  Jako stałą należy również traktować liczbę, literę a także ciąg znaków w cudzysłowach. *Np. **7** to stała, której wartość wynosi 7.*
- **Zmienna** - nazwa (identyfikator) reprezentująca określony typ danych.
  W chwili rozpoczęcia pracy programu zmienna powinna posiadać nadaną wartość początkową (nie powinna być przypadkowa).
  Np. **int liczba1;** ***/* deklaracja zmiennej liczba1 */***
  long i=25L; float s=123.16E10; char a=‘a’;*/* definicje zmiennych */*
  liczba1=21; ***// inicjacja zmiennej liczba1***
- W trakcie pracy programu wartości zmiennych ulegają zwykle zmianom.
  Należy przewidzieć zakres zmienności tych zmiennych.

Operatory i wyrażenia

- **Operatory** umożliwiają zapisywanie wyrażeń.
- Operatory matematyczne: +, -. *. /, %
  Operator przypisania =, np. a=5;
  Dwa przypisania naraz: x=y=10; *(najpierw y=10, potem x=y czyli 10)*
  Operator łączenia, np. int x, y, z;
  Operator inkrementacji: ++ i dekrementacji --, np. i++; --j;
- **Wyrażenie** jest kombinacją stałych, zmiennych i operatorów.
  Np. float a, b, pole; a=10.0; b=5.0; pole=a*b;

Przykład prostego programu na obliczenie pola koła

Zad. Obliczyć pole koła o promieniu 5 .

Algorytm: Pole koła wyraża się **S =Pi*r*r**

1) Wersja najprostsza

```
include <stdio.h>  main()  {  float PI=3.141593;   printf("%f",PI*5*5);   }
```

2) Wersja zmieniona – stała, zmienna, czyszczenie ekranu

```
/* program polkola2.c */
/* dyrektywy załączajace tzw. nazwane pliki */
#include <stdio.h>  #include <conio.h>
/* funkcja glowna */
main()
{
  const float PI=3.141593; /* stala */
  float r=5.0;     /* zmienna */
  clrscr();     /* kasowanie ekranu - okreslone w conio.h */
  printf("Pole kola o promieniu %.0f = %7.3f\n",r, PI*r*r); /* wydruk, \n – nowa linia */
  getch();  /* czeka na nacisniecie klawisza - w stdio.h */
}
```

3) Pole koła – wprowadzenie promienia i formatowanie wydruku

```
/* program polkola3.c */
/* dyrektywy załączajace tzw. nazwane pliki */
#include <stdio.h>
#include <conio.h>
/* funkcja glowna */
main()
{
  const float PI=3.141593; /* stala */
  float r, p;
  clrscr();    /* kasowanie ekranu - okreslone w conio.h */
  puts("Obliczenie pola kola o promieniu r ");  /* Napis na ekranie */
  printf("Podaj r => ");
  scanf("%f",&r);  /* wczytanie r */
  p=PI*r*r;    /* obliczenie pola - wyrazenie */
  printf("Pole kola o promieniu %.0f = %7.3f\n",r, p);  /* wydruk */
  getch();  /* czeka na nacisniecie klawisza - w stdio.h */
}
```

Kompilacja programu: BCC32 polkola3.c lub Alt F9 w TC

4) Pole koła w języku C++. Funkcja

```
//  Polkola4.cpp
 #include <iostream.h>
#include <conio.h>
  float pole_kola(float r)        // funkcja
  {
   float p;
   p = 3.1415 * r * r;
   return p;
  }
  void main(void)
  {
   float a, p;
   cout << "Podaj promien: ";    cin >> a;
   p = pole_kola(a);     cout << "Pole kola wynosi: " << p << endl;  getch();
  }
```

Pole prostokąta 1) dane wpisane w programie

```
/* ---------- polepros1.c  - boki a i b w programie ---------- */
void main(void)
{
float a, b;          /* deklarujemy zmienne przechowujące boki prostokąta */
  float pole;        /* deklarujemy zmienną zawierającą wynik obliczeń */
  a = 5; b =10;                /* przypisujemy im wartości */
  pole = a * b;                /* obliczamy pole prostokąta (tu równe 50) */
  printf("Pole = %f",pole);          /* wydruk */
  getch();                           /* czeka na klawisz */
}
```

Pole prostokąta 2) dane w programie, funkcja przed main()

```
/*  ----------------polepros2.c  - z funkcja  ----------------------/*
#include <stdio.h>
#include <conio.h>
float PoleProstokata(float bok1, float bok2) /* definicja funkcji (/
{
  /* w tym miejscu bok1 jest równy 5,  natomiast b jest równe 10 */
  float wynik;
  wynik = bok1 * bok2;
  return wynik;
}
void main(void)
{
  float a, b, p;
  a = 5; b = 10; /* zmienne */
  p= PoleProstokata(a, b); /* wywolanie funkcji */
  printf("Pole prostokata o bokach %f i %f = %f ", a, b, p);
  getch();
}
```

Pole prostokąta 3) dane wprowadzone, funkcja na końcu

```
/* ppros3.c - z funkcja */
#include <stdio.h>
#include <conio.h>
 float PoleProstokata(float bok1, float bok2); /* zapowiedz funkcji */
/* funkcja glowna */
int main(void)
{
  float a, b, p;
  puts("Obliczenie pola prostokata o bokach a i b ");
  printf("Podaj a i b oddzielone spacja: ");
  scanf("%f %f",&a, &b);
  /* wywolanie funkcji na obliczenie pola */
  p= PoleProstokata(a, b);
  printf("Pole prostokat o bokach %f i %f = %f ", a, b, p);
  getch();
  return 0;
}
float PoleProstokata(float bok1, float bok2)     /* definicja funkcji */
{
  /*   w tym miejscu bok1 jest równy a,   natomiast bok2 jest równy b  */
  float wynik;
  wynik = bok1 * bok2;
  return wynik;
}
```

Pole prostokąta 4) wprowadzanie danych, funkcja

```
* ppros4.c - z funkcja i wprowadzaniem danych */
#include <stdio.h>
#include <conio.h>

float PoleProstokata(float bok1, float bok2); /* zapowiedz
      funkcji */

/* funkcja glowna */
int main()
{
  float a, b, p;
  puts("Obliczenie pola prostokata o bokach a i b ");
  printf("Wprowadz a   ");
  scanf("%f", &a);  /* wprowadzenia a */
  printf("Wprowadz b   ");
  scanf("%f", &b); /* wprowadzenia b */
  p= PoleProstokata(a, b); /* wywolanie funkcji z parametrami a i
      b */
  printf("Pole prostokat o bokach %f i %f = %10.2f \n", a, b, p);
  printf("\nNacisnij cos ");
  getch(); /* czeka na znak */
  return 0;
}
```

```
/* Funkcja - definicja */
float PoleProstokata(float
bok1, float bok2)
{
  /* w tym miejscu bok1 jest
równy a,
   natomiast bok2 jest równy b */
  float wynik;
  wynik = bok1 * bok2;
  return wynik;
}
```

/* Program daneos1.c - dane osobowe*/

```
#include <stdio.h>
int main(void)
/* funkcja glowna, brak argumentow, zwraca typ calkowity int */
{
    char imie[20]; /* deklaracja tablicy znakow – łańcuch – tekst na 19 znaków */
        int i;  /* deklaracja zmiennej całkowitej i */
    printf("\nPodaj swoje imie "); /* wydruk napisu na ekran,\n – nowa linia */
    gets(imie);  /* wprowadzenie imienia */
    puts("Ile masz lat? "); /*  wydruk napisu – funkcja puts() */
    scanf("%d",&i); /* wprowadzenie lat */
    printf("\n%s ma %d lat.",imie, i); /* wyświetlenie imienia i lat */
    return 0;  /* funkcja glowna zwraca 0 – pomyślny koniec */
```

Wyświetlenie czasu C++ - program w Dev C++

```
// Program czas.cpp w jezyku C++ - wyświetlenie czasu - komentarz w C++
// instrukcja do…while
#include <cstdlib>
#include <iostream>
#include <time.h>
using namespace std;

int main (int argc, char *argv[])
{
  int a;    time_t czas;

 do
{
        cout << "1 - Wyswietl aktualny czas" << endl;
        cout << "2 - Zakoncz program" << endl;
        cout << "Twoj wybor?";     cin >> a;
        if (a == 1)    {     time(&czas);     cout << ctime(&czas);     }
  } while (a != 2);

 cout << "Do zobaczenia" << endl;
  system("PAUSE");
  return EXIT_SUCCESS;
}
```

Podsumowanie 1 – podstawy programowania w C

- Są w **C** **2 rodzaje komentarzy** /* wielowierszowy w **C i C++** */ i jednowierszowy w **C++** **//** - i można w nich pisać uwagi, pomijane przez kompilator.
- Aby zatrzymać pracę programu, używamy funkcji ***getch();***.
- Do czyszczenia ekranu służy funkcja ***clrscr();*.**
- Na końcu funkcji **main** jest zwykle ***return 0;***.
- W tekście możemy używać tzw. znaków specjalnych, np. przejście do następnej linii ***n*.**
- **Program** składa się z ciągu rozdzielonych średnikami instrukcji położonych pomiędzy słowami kluczowymi **{** i **}**
- **Instrukcje** mogą zawierać wyrażenia oraz wywołania funkcji.
- **Wyrażenia** składają się ze stałych, operatorów i identyfikatorów.
- **Identyfikatory** są nazwami obiektów składających się na program. Mogą one zawierać litery, cyfry i znaki podkreślenia, nie mogą jednak zaczynać się od cyfr.

Podstawowe elementy języka C

- **zestaw znaków** (litery, cyfry, znaki specjalne: ! * + \ " < # (= | { > %) ~ ; } / ^ - [: , ? & _] ')
- **nazwy i słowa zastrzeżone** (np. **auto break case char const goto if int long register unsigned void volatile while**)
- **typy danych** (np. int, char, float, doble)
- **stałe** (np. 121, 23L, 021, 0x12, 25L, 'a', ***const int a=10;***)
- **zmienne i tablice** (np. *float zm1, zm2; double tab[10];***)**
- **deklaracje** (np. *int i, j; double k; char znak; int main()*)
- **Wyrażenia -** to co zwraca element, np. 3.2, a=a+b; k==m; y=x=a+b; b=a[j+1];
- **Instrukcje -** fragmenty tekstu programu powodujące jakąś czynność komputera podczas wykonywania programu, np. *p=a*b; printf("Pole=%f",p); k=funkcja_d(5, 6, 17.33);*

Zestaw znaków C:

- **Litery** małe i duże języka łacińskiego (angielskiego)
- **cyfry 0..9**
- **znaki specjalne**:
  ! * + \ " < # (= | { > %) ~ ; } / ^ - [: , ? & _] ' .
  oraz znak odstępu (spacja)

Nazwy i słowa zastrzeżone (kluczowe, zarezerwowane)

- **Nazwy** służą do identyfikowania elementów programu (stałych, zmiennych, funkcji, typów danych, itd.).
- Nazwa składa się z ciągu liter i cyfr, z tym, że pierwszym znakiem musi być litera.
  Znak podkreślenia traktowany jest jako litera.
- W języku C rozróżniane są duże i małe litery w identyfikatorach.
- Użycie odstępu w nazwie jest niedozwolone.
- Niektóre implementacje rozpoznają w nazwie do 8 znaków, inne więcej (do 32)

Słowa zastrzeżone

- Słowa zastrzeżone są to słowa o szczególnym znaczeniu dla języka, których nie wolno używać programiście np. jako nazw zmiennych.
- **Standardowy zestaw znaków zastrzeżonych języka C:**
  auto break case char const goto if int long register unsigned void volatile while
- Niektóre kompilatory mają niektóre lub cześć z następujących słów kluczowych
  ada asm entry far fortran huge near pascal
- Niektóre kompilatory mogą mieć też inne słowa zastrzeżone

Podstawowe typy danych i rozmiary danych

- W języku C występuje tylko kilka podstawowych typów danych:
  - **char** - jeden bajt, zdolny pomieścić 1 znak
  - **int** - typ całkowity
  - **float** - typ zmiennopozycyjne pojedynczej precyzji
  - **double** - typ zmiennopozycyjny podwójnej precyzji
- Dodatkowo występuje kilka kwalifikatorów stosowanych z tymi podstawowymi typami.
- **Kwalifikatory short** i **long** odnoszą się do obiektów całkowitych

Typy, opis, zakres wartości i reprezentacja danych

Typ	Opis	Zakres wartości	Reprezentacja
char (signed char)	pojedynczy znak (np. litera)	-128...127	1 bajt
unsigned char	znak (bez znaku - dodatni)	0...255	1 bajt
short	liczba całkowita krótka	-32768...32767	2 bajty
unsigned short	liczba krótka bez znaku	0...65535	2 bajty
int (signed int)	liczba całkowita	-32768...32767	2 bajty
unsigned int	całkowita bez znaku	0...65535	2 bajty
long (long signed int)	liczba całkowita długa	-2147483648...2147483647	4 bajty
unsigned long (long unsigned int)	j.w. bez znaku	0...4294967295	4 bajty
float	l. rzeczywista	-3.4E-38..-3.4E-38,0,3.4E-38..3..3.4E38	4 bajty
double	l. rzeczywista podwójna	-1.7E308..-1.7E-308,0,1.7E-308..1.7E308	8 bajtów
long double	l. rzecz. długa podwójna	-1E4932..-3.4E-4932,0,3.4E-4932..1.1E4932	10 bajtów

Szczegóły zależne są od konkretnej implementacji kompilatora języka C.

Są jeszcze inne typy danych jak: **void** (typ funkcji nie zwracającej wartosci), **enum** (typ wyliczeniowy) oraz **typ wskaźnikowy**. Typ wyliczeniowy (enum) reprezentowany jest na 2 bajtach, wskaźnikowy na 2 lub 4 (w zależności od modelu pamięci), void nie jest reprezentowany.

ZMIENNE i STAŁE, TYPY ZMIENNYCH, DEKLARACJE, OPERATORY, WYRAŻENIA

- **Zmienne i stałe** są podstawowymi obiektami danych, jakimi posługuje się program.
- **Deklaracje** wprowadzają potrzebne zmienne oraz ustalają ich typy i ewentualnie wartości początkowe.
- **Operatory** określają co należy z nimi zrobić. **Wyrażenia** wiążą zmienne i stałe, tworząc nowe wartości.

Stałe, zmienne

- **Deklaracje** służą do określenia struktury danych programu.
- **Instrukcje** programu oddzielamy za pomocą średnika
- Program powinien być zapisywany z wykorzystaniem **wcięć**.
- Podstawowymi obiektami programu są **stałe** i **zmienne**.
- **Stała** to jakaś konkretna, niezmienna wartość, którą posługujemy się używając konkretnej nazwy symbolicznej,
  np. **const** float PI=3.141593;
  Każde wystąpienie nazwy-stałej w programie zostanie podczas kompilacji zastąpione wartością.
  Stałe definiuje się za pomocą słowa kluczowego **const**
- **Zmienna** jest wielkością, która może się zmieniać.
  Posługujemy się nią korzystając z nazwy symbolicznej zwanej też ***identyfikatorem***, np. **float r;**
- Możliwe jest definiowanie wielu zmiennych tego samego typu w jednej linii, wystarczy je wtedy rozdzielić przecinkami.
- KAŻDA ZMIENNA PRZED UŻYCIEM W PROGRAMIE MUSI BYĆ WCZEŚNIEJ **ZADEKLAROWANA**.

Stałe

- W języku C występują następujące *rodzaje stałych*:
  - stałe **całkowitoliczbowe**
    dziesiętne – znaki 0..9, +., - np. **0 876 -122 +909**
    ósemkowe – znaki 0..7, +, -; 0 na początku, np. **0 011 0247**
    szesnastkowe – znaki 0 .. F, +, -; 0x lub 0X na początku, np. 0x, 0X1234
  - stałe **rzeczywiste**, - znaki 0.9, +, -, E, e, np. **0. 2. 0.2 876.45 2.4E12, 1.2e-3**
  - stałe **znakowe** - pojedyncze znaki ASCII ograniczone apostrofami, np. ‘A’ ‘#’ ‘’ (spacja)
  - **Escape sekwencje**, np. **\n** (nowa linia), **\”** – cudzysłów, **\b** - dźwięk
- **łańcuchy znaków** - ciąg znaków ograniczony znakami cudzysłowu, np.

  "Wynik = " "Linia nr 1\nLinia nr2” "A + B = „

Stałe całkowitoliczbowe

Nazwa	Dozwolony zestaw znaków	Uwagi	Przykłady
Stałe dziesiętne	0..9 + -	Jeśli więcej niż 1 znak, pierwszym nie może być 0	0 1 876 -122 +909
Stałe ósemkowe	0..7 + -	Pierwszą cyfrą musi być 0	0 0111 0777 -0777 +0222
Stałe szesnastkowe	0..9 a..f A..F + -	Pierwszymi znakami muszą być 0x lub 0X	0x 0X1234 0XAFDEC 0xffff

W celu zainicjowania zmiennych typów **int** i **long** po znaku równości podajemy całkowitą stałą liczbową.

Przykłady:

```
int l=100;
unsigned k=121;

int la = 0x2ffa;
long i = 25L; /* long */
long unsigned z = 1000lu; /* lub 1000UL */
```

Stałe rzeczywiste

- **Stałe rzeczywiste**, zwane **zmiennoprzecinkowymi** reprezentują liczby dziesiętne.
  Dozwolony zestaw znaków: **0..9 . + - E e**
  (E lub e reprezentuje podstawę systemu tj. 10)
  *Uwagi: 1.2*10^-3 można zapisać 1.2E-3 lub 1.2e-3.*
- Stałe rzeczywiste zawierają albo kropkę dziesiętną (np. 123.4), albo wykładnik **e** (np. 1e-2) albo jedno i drugie.
- Typem stałej zmiennopozycyjnej jest **double**, chyba, że końcówka stanowi inaczej.
- Występująca na końcu litera **f** lub **F** oznacza obiekt typu **float**, a litera **l** lub **L** - typu **long double**.
- Przykłady: 0. 2. 0.2 876.543 13.13E13 2.4e-5 2e8

Deklaracja i inicjalizacja zmiennych rzeczywistych

Przykłady

int i, j; */* deklaracja */*

float s=123.16e10; */* liczba 123.16*10^16 */*

i=10; */*inicjalizacja */*

double x=10.; */*definicje: deklaracja i inicjacja */*

long double x=.12398;

double xy=-123.45;

Stałe znakowe

- Stała znakowa jest liczbą całkowitą;
- Taką stałą tworzy jeden znak ujęty w apostrofy, np. **'x'**.
  Są to więc pojedyncze znaki zamknięte dwoma apostrofami.
  Zestaw dowolnych widocznych znaków ASCII.
- Wartością stałej znakowej jest wartość kodu znaku w maszynowym zbiorze znaków.
  Np. wartością stałej '0' jest liczba 48 - liczba nie mająca nic wspólnego z numeryczną wartością 0.
- Pewne znaki niegraficzne mogą być reprezentowane w stałych znakowych i napisowych przez sekwencje specjalne, takie jak **\n** (znak nowego wiersza).
- Przykłady: **'A' '#' ' '**

```
char a='a';
```

Escape-sekwencje - sekwencje specjalne

Niektóre znaki "**niedrukowalne**" mogą być przedstawione w postaci tzw. **escape-sekwencji**, np. znak nowej linii jako sekwencja \n.
Pierwszym znakiem tej sekwencji jest **backslash** (\).
Sekwencja specjalna wygląda jak 2 znaki, ale reprezentuje tylko jeden znak

Sekwencja znaków	Wartość ASCII	Znaczenie
\a	7	Sygnał dźwiękowy (BEL)
\b	8	Cofniecie o 1 znak (BS)
\t	9	Tabulacja pozioma (HT)
\v	11	Tabulacja pionowa (VT)
\n	10	Nowa lina (LF)
\f	12	Nowa strona (FF)
\r	13	Powrót karetki (CR)
\"	34	Cudzysłów
\'	39	Apostrof
\?	63	Znak zapytania
\\	92	Backslash
\0	0	Znak pusty (null) - nie jest równoważne stałej znakowej '0'

Stałe napisowe - napisy, łańcuchy znaków (stałe łańcuchowe)

- **Stała napisowa** lub napis jest ciągiem złożonym z zera lub więcej znaków, zawartym między znakami cudzysłowu, np. "Jestem napisem".

Przykłady:

"Wynik = "

" + 2 mln $"

"Linia nr 1\nLinia nr2"

""

"A + B = "

- Łańcuchy mogą zawierać escape-sekwencje.
- Łańcuchem pustym są same cudzysłowy .
- Łańcuch w sposób niejawny jest zakończony znakiem nul czyli **\0**.
- Dlatego np. stała znakowa **'K'** nie jest równoważna łańcuchowi **"K"**.

Łańcuch znaków

- **Łańcuch znaków** (napis) można traktować jako tablicę złożoną ze znaków, uzupełnioną na końcu znakiem **'\0'** Taka reprezentacja oznacza, że praktycznie nie ma ograniczenia dotyczącego długości tekstów.
- Programy muszą jednak badać cały tekst, by określić jego długość, np. **strlen(s)** ze standardowej biblioteki.
- Przykład:
  Napis "Katowice”, który może być zadeklarowany jako tablica jednowymiarowa, której elementami są znaki, np.
  char napis[] = "Katowice";

Deklaracja i inicjalizacja łańcuchów

Przykłady

```
Char tekst[] =  "Katowice”
unsigned char teKst[30]=  "Taki sobie teKst"
char s[10]="\n\fAndrzej\x60";
char str[20]={'\n','\f', 'A', 'n', 'd', 'r', 'z', 'e', 'j', '\x60', '\0'};
char *str1 = "Programowanie w języku C/C++";
```

Stałe wyliczeniowe (enumeration constant)

- **Stałe wyliczeniowe** *tworzą zbiór stałych o określonym zakresie wartości.*
- Wyliczenie jest listą wartości całkowitych, np.
  enum boolean {NO, YES};
  Pierwsza nazwa na liście wyliczenia ma wartość **0**, następna **1** itd., chyba że nastąpi jawnie podana wartość.
- Przykłady:
  enum KOLOR {CZERWONY, NIEBIESKI, ZIELONY, BIAŁY, CZARNY}
  enum KOLOR {red=100, blue, green=500, white, black=700};
  red przyjmie wartość 100,
  blue 101, green 500,
  white 501, black 700

Stałe symboliczne - makrodefinicje

- **Stała symboliczna** jest nazwą przedstawiającą inną **stałą** - numeryczną, znakową lub tekstową. Definicję stałej symbolicznej umożliwia instrukcja **#define**:
  #define NAZWA tekst
  gdzie NAZWA jest nazwą stałej symbolicznej, a tekst jest związanym z tą nazwą łańcuchem znaków
- Przykłady:

Makrodefinicje proste:

#define identyfikator <ciąg-jednostek-leksykalnych>

#define PI 3.14159

#define TRUE 1

#define FALSE 0

#define NAPIS1 Siemianowice

#define IMIE "Andrzej" *// (puts(IMIE) rozwija w tekst puts("Andrzej")*

#define IMIE_I_NAZWISKO IMIE+"Zalewski"

#define WCZYTAJ_IOSTREAM_H #include <iostream.h>

Makrodefinicje parametryczne

- **#define identyfikator(idPar1, idPar2,...) ciąg_jedn_leksykalnych**
- Np.
- **#define ILORAZ(a,b) ((a)/(b))**
  //- makrodefinicja ILORAZ - parametry w nawiasach!
- **#define SUMA(a,b) ((a)+(b))**
- W trakcie kompilacji nazwy stałych symbolicznych są zastąpione przez odpowiadające im łańcuchy znaków. Ułatwia to parametryzację programu, a także umożliwia zastępowanie często niewygodnych w pisaniu sekwencji programu, tworzenie makrodefinicji, tworzenie własnych dialektów języka, czy nawet języków bezkompilatorowych (na bazie kompilatora C).

/* Przykład zastosowana pseudoinstrukcji #define - Program p8.c */

```
#include <stdio.h>
#include <conio.h>
#define PI       3.1415926
#define PROMIEN   3.3
#define WYSOKOSC 44.4
#define WYNIK    printf("Objetosc walca = %f", objetosc)
main()
{
  float promien, wysokosc, objetosc;
  promien  = PROMIEN;
  wysokosc = WYSOKOSC;
  objetosc = PI * promien * promien * wysokosc;
  WYNIK;
 getch();
}
```

Zmienne

- **Zmienną** nazywamy nazwę (identyfikator) reprezentującą określony typ danych. **Zmienna** jest to pewien fragment pamięci o ustalonym rozmiarze, który posiada własny identyfikator (nazwę) oraz może przechowywać pewną wartość, zależną od typu zmiennej.
- W chwili rozpoczęcia pracy programu zmienna powinna posiadać nadaną wartość początkową (nie powinna być przypadkowa).
  W trakcie pracy programu wartości zmiennych ulegają zwykle zmianom.
  Należy przewidzieć zakres zmienności tych zmiennych.
- W języku C wszystkie zmienne muszą być zadeklarowane przed ich użyciem, zwykle na początku funkcji przed pierwszą wykonywaną instrukcją.
- **Deklaracja** zapowiada właściwości zmiennych.
  Deklaracja składa się ona z nazwy **typu** i **listy zmiennych**, jak np.
  int fahr, celsius;

Zmienne c.d.

- W języku C oprócz podstawowych typów:
  char, short, int, long, float, double
  występują także
  tablice, struktury, unie, wskaźniki do tych obiektów oraz **funkcje** zwracające ich wartości.
- W przypadku zmiennych należy zwrócić uwagę na
  2 zasadnicze rzeczy: *(przykład poniżej – program **p9.c**)*
  - nadanie wartości początkowej
  - oszacowanie zakresu zmienności
- Zmienne mogą być **automatyczne** oraz zmienne **globalne** - **zewnętrzne**.
- **Zmienne automatyczne** pojawiają się i znikają razem z wywołaniem funkcji. Zmienne zewnętrzne to zmienne **globalne**, dostępne przez nazwę w dowolnej funkcji programu.
- **Zmienna zewnętrzna** musi być zdefiniowana dokładnie jeden raz na zewnątrz wszystkich funkcji - definicja przydziela jej pamięć.

```
/*  Obliczanie objetosci walca - zmienne    p9.c */
#include <stdio.h>
#include <conio.h>
#define PI        3.1415926
#define PROMIEN   3.3
#define WYNIK    printf("Objetosc walca = %f", objetosc)
#define NL printf("\n");
main()
{
  float promien, wysokosc, objetosc;
  int i;
  promien  = PROMIEN;
  objetosc = PI * promien * promien * wysokosc;   /* uwaga  nie zainicjowana zmienna wysokosc */
  WYNIK;   /* wydruk wyniku */
  i=40000;    printf("\ni = %i ", i);
  wysokosc=10;    NL; /* ustalenie zmiennej wysokosc i nowa linia */
  objetosc = PI * promien * promien * wysokosc;
  WYNIK; /* wynik prawidlowy */
  getch();
}
```

Deklaracje

- **Deklaracje** umożliwiają wyspecyfikowanie grup zmiennych określonego typu. Większość kompilatorów dopuszcza nadanie wartości początkowej zmiennej w deklaracji (inicjalizację).
- Wszystkie zmienne muszą być zadeklarowane przed użyciem.
  W deklaracji określa się typ, a następnie wymienia jedną lub kilka zmiennych tego typu, np. **int lower, upper; char c, line[1000];**
- W **deklaracjach** można nadawać zmiennym wartości początkowe, np.
  char esc='\'; int i=0; float eps=1.0e-5; int limit=MAXLINE+1;
- Jeśli zmienna nie jest automatyczna, to jej wartość początkową nadaje się tylko raz - jakby przed rozpoczęciem programu; jej inicjatorem musi być wyrażenie stałe. Zmiennym automatycznym jawnie określone wartości początkowe nadaje się za każdym razem przy wywołaniu funkcji lub przy wejściu do zawierającego je bloku. Zmiennym zewnętrznym i statycznym przez domniemanie nadaje się wartość pocztkowa zero.
  Zmienne automatyczne bez jawnie określonej wartości poczatkowej mają wartości przypadkowe (śmiecie).
- Kwalifikator **const** mówi, że wartość zmiennej będzie stała, np.
  const double e=2.71828182; const char mas[]="Uwaga:";

Wyrażenia

Wszystko co zwraca wartość jest wyrażeniem.

Wyrażeniem może być samodzielny element, np. liczba, stała, zmienna.

Może to być też kombinacja w/w elementów połączonych znakami operatorów, np. arytmetycznych lub logicznych.

Przykłady:

```
3.2          PI
a=a+b;       x=y;
x<LAFA;      k == m;
y = x = a+b;
x != y;
b[i-2] = a[j+1];
```

Instrukcje

- **Instrukcje** są to fragmenty tekstu programu (zwykle linie), które powodują jakąś czynność komputera w trakcie wykonywania programu.
- Instrukcje można podzielić na grupy:
  - **Instrukcje obliczające wartości wyrażeń,**
    np. a=b+c;
    Każda taka instrukcja musi być zakończona średnikiem
  - **Instrukcje grupujące (złożone) -** ograniczone klamrami **{ },** np.
    { a=3; b=2.2; p=a*b; printf("Pole=%f",p);}
  - **Instrukcje sterujące**, warunkowe: if, if…else, switch, while, do…while, for
  - **Instrukcje wywołania funkcji: *funkcja(parametry_akualne);***
    Np.
    int funkcja_d(int a, int b, float c); *//deklaracja funkcji, gdzieś definicja*
    int k;
    k=funkcja_d(5, 6, 17.33); *//wywołanie funkcji*

Przykład programu z użyciem zmiennych - prosty kalkulator - liczby całkowite

```
/* Program kalk1.c - początek*/
#include <stdio.h>
#include <conio.h>
int main() /* funkcja glowna */
{
int a,b; /* deklaracja zmiennych
    całkowitych a i b */
int suma,roznica,iloczyn;
float iloraz; /* deklaracja zmiennej
    rzeczywistej */
clrscr();  /* kasowanie ekranu */
printf("Prosty kalkulator\n");
printf("\nPodaj liczbe całkowitą a: ");
scanf("%d",&a);  /* wczytanie liczby a */
printf("Podaj liczbe b: ");
scanf("%d",&b); /* wczytanie liczby b */
```

```
/* c.d. -  obliczenia */
suma=a+b;
roznica=a-b;
iloczyn=a*b;
iloraz=(float)a/(float)b; /* operator
rzutowania w dzieleniu */
/* Wydruk wynikow */
printf("\nWyniki dzialan:\n");
printf("\nSuma:    %d  ",suma);
printf("\nRoznica: %d
",roznica);
printf("\nIloczyn: %d  ",iloczyn);
printf("\nIloraz:  %f  ",iloraz);
getch(); /* czeka na naciśniecie
klawisza */
return 0 ;
}
```

Objaśnienia do programu kalkulator

- Typ *int* (integer) oznacza, że zmienna może przyjmować wartości całkowite z zakresu **+/- 32767**.
  Jeżeli zamierzamy używać w programie dużych liczb wtedy zamiast typu *int* używamy typu *long*.
  Kompilator już wie, że w kodzie programu będą pojawiać się zmienne *a* i *b* i wie ile pamięci musi zarezerwować na ich przechowanie.
- Następnie wypisujemy na ekranie komunikat, który prosi nas o wpisanie liczby.
  Program zatrzyma się w tym miejscu i będzie czekał dotąd, aż napiszemy liczbę i zatwierdzimy je ENTERem.
- Funkcja ***scanf()*** służy do odczytu wprowadzonych danych z klawiatury
  Wzorzec konwersji określa typ zmiennej, którą wpiszemy z klawiatury lub wypiszemy na ekranie.

Zmienne **typów liczbowych** mogą przyjmować wartości tylko z określonych przedziałów liczbowych.

typ zmiennej	Zakres
int	-32 768 do 32 767
long	-2 147 483 648 do 2 147 483 647
float	3,4* 10^(-38) do 3,4*10^(38)
double	1,7* 10^(-308) do 1,7*10^(308)
long double	3,4* 10^(-4932) do 3,4*10^(4932)

Prosty kalkulator – liczby rzeczywiste

```
/* Program kalk2.c */
#include <stdio.h>
#include <conio.h>
int main() /* funkcja glowna */
{
float a,b; /* deklaracja zmiennych
     rzeczywistych a i b */
float suma,roznica,iloczyn;
float iloraz; /* deklaracja zmiennej
     rzeczywistej */
clrscr();  /* kasowanie ekranu */
printf("Prosty kalkulator\n");
printf("\nPodaj liczbe a: ");
scanf("%f",&a);  /* wczytanie liczby a */
printf("Podaj liczbe b: ");
scanf("%f",&b); /* wczytanie liczby b */
printf("\nLiczba wprowadzona a =    %f  ",a);
printf("\nLiczba wprowadzona b =    %f  ",b);
/* c.d - obliczenia  */
suma=a+b;
roznica=a-b;
iloczyn=a*b;
iloraz=a/b;
/* Wydruk wynikow */
printf("\nWyniki dzialan:\n");
printf("\nSuma:    %f  ",suma);
printf("\nRoznica: %f  ",roznica);
printf("\nIloczyn: %f  ",iloczyn);
printf("\nIloraz:  %f  ",iloraz);

getch(); /* czeka na naciśniecie
klawisza */
return 0 ;
}
```

Podsumowanie 2 – liczby, zmienne, komunikaty

- Aby wczytać liczbę należy użyć funkcji ***scanf*** w postaci: ***scanf("wzorzec",&zmienna);***
- Aby wypisać wczytaną w ten sposób liczbę należy użyć funkcji ***printf***, która służy do wypisywania komunikatów. Postać funkcji: ***printf("Komunikat wzorzec",zmienna);***
- W funkcji ***scanf*** zawsze przed nazwą zmiennej używamy znaku ***&***, a nie robimy tego przy używaniu funkcji ***printf***.
- Zmienna służy do przechowania danych, których wartość ustala się w trakcie działania programu i może być zmieniana.
- Każda zmienna musi być zadeklarowana przed jej użyciem jako zmienna odpowiedniego typu: ***int, float, char*** itp.
- Do wypisywania komunikatów służy funkcja ***printf***, lub ***puts*** a do wczytywania zmiennych funkcja ***scanf***.
- Do poprawnego użycia obu funkcji należy znać podstawowe wzorce konwersji: ***%d, %f, %s***.

Operacje na znakach i łańcuchach znaków

- **Typy znakowe:** - deklaracja **char** zmienna;
  pojedynczy znak: char znak;
  np. *char zn1='a'; char litera; litera = 'F';*
  łańcuch znaków (napis, słowo)
  - char *napis; np. *char *str1=„Programowanie";*
  - char napis[n]; np. *char linia[80]; gets(linia);*
    char imie[20]; scanf("%s",imie);
  - np.char napis[]="Tekst",
    np. *char pozdrowienie="Jak się masz"*
- **Wzorzec konwersji** przy wyświetleniu lub wczytywaniu zmiennej typu **char**
  - ***%c*** dla pojedynczego znaku (łańcucha jednoznakowego)
    np. *char znak1; scanf("%c",&znak1);*
  - ***%s*** dla łańcucha dłuższego niż 1 znak
    np. *char slowo2[20]; scanf("%s",slowo2);*

```c
/* Program znaki1.c */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main (void)
{
char znak1,znak2,znak3; //deklaracja znaków
char slowo1[10],  slowo2[15];
char slowo3[]="Adam"; // definicja
clrscr();  // czyszczenie ekranu
znak1='a';  // inicjalizacja zmiennej znak1
znak2=102; // znak w postaci kodu dziesiętnego ASCII - litera f
printf("Podaj znak3: "); scanf("%c",&znak3); // podajemy jakiś znak
printf("\nPodaj slowo1: "); scanf("%s",slowo1); // wpisujemy słowo 1
printf("\nPodaj slowo2: "); scanf("%s",slowo2); // wpisujemy słowo 2
printf("\nZmienne zawieraja znaki: ");
printf("znak1 (a), znak2 (102), znak3 (wprowadzony): %c %c %c  ",znak1,znak2,znak3);
printf("\noraz slowa: ");
printf("slowo1, slowo2, slowo3: %s %s %s ",slowo1, slowo2, slowo3);
getch();
return 0 ;
}
```

Przypisanie wartość zmiennej znakowej i napisom

- Przypisać wartość zmiennej jednoznakowej możemy na kilka sposobów:
  - w apostrofach: *zmienna='a';*
  - poprzez przypisanie kodu znaku: *zmienna=97;*
  - poprzez wczytanie znaku z klawiatury funkcją *scanf()*: *scanf("%c",&zmienna);*
- Przypisanie wartości do zmiennej dla łańcucha znaków dłuższego niż jeden znak odbywa się podobnie jak w pierwszym przypadku, z tą jednak różnicą, że zamiast apostrofów ' należy używać cudzysłowia "

char slowo3[]="Adam"; // definicja

Program, który podaje kod wciśniętego przez nas klawisza i znak kodu

```
/* Program  kodkl1.c -  podaje kod wciśniętego przez nas klawisza i znak kodu */:
#include <stdio.h>
#include <conio.h>
#include <string.h>
main (void)
{
char znak;
int kod;
clrscr();

printf("Wciśnij znak na klawiaturze: \n");
scanf ("%c", &znak);
printf("Kod wciśniętego znaku to: %d \n",znak);

printf("Podaj kod znaku: \n");
scanf ("%d", &kod);
printf("Znak o podanym kodzie to: %c \n",kod);

getch();
return 0 ;
}
```

Odczytanie długości łańcucha znaków – strlen(lancuch)

```
/* dluglanc.c */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main (void)
{
int dlugosc;
char *lancuch;
clrscr();
lancuch="Adam Nowak";
Dlugosc = strlen (lancuch);
printf("Lancuch '%s' ma: %d znaków \n",lancuch, dlugosc);
printf("Pierwsza litera łańcucha to: %c \n",lancuch[0]);
printf("Ostatnia litera łańcucha to: %c \n",lancuch[dlugosc-1]);
getch();
return 0 ;
}
```

Inne funkcje operujące na łańcuchach:

strlwr, strupr, strcat, strrv, streset – z biblioteki string.h

strcat() - łączy dwa łańcuchy, ***strcmp()*** - porównuje dwa łańcuchy rozróżniając małe i duże litery,
strlwr() i ***strupr()*** - zamienia w danym łańcuchu duże litery na małe i odwrotnie,
strrev() - odwraca kolejność znaków w łańcuchu, ***strset()*** - wypełnia łańcuch danym znakiem.

```
/* znaki5.c - operacje na tekstach */
#include <stdio.h>
#include <conio.h>
#include <string.h>
main (void)
{
char *lancuch1, *lancuch2, *lancuch3;
char znakwyp='x';  // znak wypełniający
clrscr();
puts("Operacje tekstowe\n");
lancuch1="Janusz";
lancuch2="Kowalski";
printf("Lancuch1 to: %s \n",lancuch1);
printf("Lancuch2 to: %s \n",lancuch2);
printf("\nZmieniamy duze litery na male: \n");
strlwr(lancuch1);
printf("Lancuch1 wyglada teraz tak: %s \n",lancuch1);
printf("\nZmieniamy male litery na duze: \n");
strupr(lancuch2);
printf("Lancuch2 wyglada teraz tak: %s \n",lancuch2);
printf("\nLaczymy dwa lancuchy: \n" );
lancuch3=strcat(lancuch1,lancuch2);
printf("Lancuch3 wyglada teraz tak: %s \n",lancuch3);
printf("\nOdwracamy kolejnosc znakow w lancuchu: \n" );
strrev(lancuch3);
printf("Lancuch3 wyglada teraz tak: %s \n",lancuch3);
printf("\nWypelniamy lancuch znakiem 'x':\n" );
strset(lancuch3,znakwyp);
printf("Lancuch3 wyglada teraz tak: %s \n",lancuch3);
getch();
return 0 ;
}
```

Podsumowanie 3 –znaki, łańcuchy znaków

1. Zmienne liczbowe mogą zawierać się w pewnych zakresach, których nie można przekraczać.
2. Deklaracja zmiennej znakowej: ***char znak;*** a zmiennej łańcuchowej: ***char *slowo;***
3. Wartość zmiennej znakowej można przypisać w programie poprzez umieszczenie znaku w apostrofach lub przez napisanie jego kodu.
4. Wartość zmiennej łańcuchowej można przypisać w programie poprzez umieszczenie napisu w cudzysłowie.
5. Zmienne znakowe i łańcuchowe można wczytywać z klawiatury używając funkcji ***scanf()*** i odpowiednich wzorców konwersji**:** ***%s*** dla ciągu znaków i ***%c*** dla pojedynczego znaku.
6. Każdy znak posiada swój kod ASCII.
7. Kod ASCII mają również znaki nie przedstawione na klawiaturze komputera. np. ß, ö.
8. Łańcuch, który wygląda jak liczba nie jest liczbą.
   Istnieją funkcje, które potrafią przekonwertować łańcuch liczbowy do postaci liczby.
9. Mając dany łańcuch, możemy odczytać dowolny jego znak używając nawiasów kwadratowych. Pierwszy wpisany znak ma numer 0, a nie 1.
10. Każdy ciąg kończy znak **'\0'.**
11. Długość łańcucha można ograniczyć przy deklaracji, np.: ***char slowo[10];***
12. Łańcuchy można ze sobą porównywać, łączyć, odwracać w nich kolejność liter, zmieniać małe litery na duże i odwrotnie, a także przeszukiwać, kopiować na siebie itp. Nazwy funkcji, które to wykonują zawsze zaczynają się na '**str**' (z angielskiego: string).

Wejście i wyjście programu

- **Wejście i wyjście programu**
- Do podstawowych funkcji języka C, umożliwiających komunikację z otoczeniem należą:
  dla operacji **wejścia**: **getchar, gets, scanf;**
  dla operacji **wyjścia: putchar, puts, printf**
- W **DOS**, wyniki wysyłane na ekran mogą być przy pomocy znaku potoku wysłane do pliku lub na drukarkę.
  Np.
  p11 > Wynik.txt (do pliku)
  p11 > PRN (na drukarkę)
- **Funkcja putchar:** int putchar(int)
  Funkcja wysyła pojedynczy znak na zewnątrz (do standardowego strumienia wyjściowego **stdout**, standardowo na ekran) .
  Funkcja (makro) zwraca wartość wypisanego znaku lub EOF (jako sygnał błędu).

Przykład programu z funkcją **putchar**

```
#include <stdio.h>
#define SPACJA 32
main()
{
  char napis[] = "ABCDEFGHIJKLMNOPQRS...";
  int znak = 73;
  /* wyprowadzanie pojedynczych znakow
     przy pomocy funkcji  putchar   */
  putchar('\n');     putchar(znak);     putchar(SPACJA);
  putchar(55);       putchar(SPACJA);   putchar('Z');
  putchar(SPACJA);   putchar('\066');   putchar(SPACJA);
  putchar('\x3A');   putchar('\n');     putchar(napis[0]);
  putchar(SPACJA);   putchar(napis[4]); putchar(SPACJA);
  putchar(znak+'\066'-52);              putchar('\n');
}
```

Wynik:

I 7 Z 6 :

A E K

Funkcja puts

Funkcja puts: int puts(const char *s);

Wysyła łańcuch s do standardowego strumienia wyjściowego (stdout) i dołącza znak końca wiersza. W przypadku powodzenia operacji wartość jest nieujemna, w przeciwnym wypadku EOF.

```
/* Program puts1.c */
#include <stdio.h>
#include <conio.h>
#define NOWA_LINIA putchar('\n')
main()
{
 char napis[] = "ABCDEFGHIJKLMNOPQRS...";
 /* wyprowadzanie pojedynczych linii tekstu  przy
    pomocy funkcji  puts  */
 puts("Program z funkcja puts");
 puts("\n\n");
 NOWA_LINIA; puts(napis); NOWA_LINIA;
 puts("To jest praktyczna funkcja");
 puts("\066\067\x2B\x2A itd.");
 getch();
}
```

Wynik

Program z funkcja puts

ABCDEFGHIJKLMNOPQRS...

To jest praktyczna funkcja
67+* itd.

Funkcja printf()

- Funkcja printf() wyprowadza wynik przetwarzania w różnych formatach.
  printf (łańcuch, lista argumentów); lub inaczej
  printf(ciag_formatujący, lista parametrów);

Ciąg formatujący jest zwykłym ciągiem znaków do wyświetlenia na ekranie. Jednak niektóre znaki mają funkcję specjalną i nie zostaną one po prostu wyświetlone.
Takim właśnie znakiem jest znak **%** **.**
Gdy funkcja **printf**() go napotka to wie, że po nim wystąpi określenie rodzaju argumentu i formatu jego wyświetlenia na ekranie.

- Ogólnie **ciąg formatujący** ma zapis :
  % [flagi] [szerokość] [precyzja] [modyfiktor wielkości] typ_parametru
- Tylko "typ_parametru" musi wystąpić po znaku **%** ,
  natomiast parametry podane w nawiasach kwadratowym są opcjonalne i może ich w ogóle nie być (tak jest w przedstawionym przykładzie).

```
/*  Przykłąd 1 z printf() - Program  printf1.c        */
#include <stdio.h>
main()
{
  int  k = 21101;  /* liczba dziesietna */
  printf("\nk(_10) = %i   k(_8) = %o    k(_16) = %X", k, k, k);
}
```

Wynik: k(_10) = 21101 k(_8) = 51155 k(_16) = 526D

Formaty realizowane przez funkcję printf()
(znaki typu w łańcuchach formatujących)

Typ danych - znak typu w formacie	Argument wejściowy	Format wyjściowy
Liczba		
%d, %i	**int**	liczba całkowita ze znakiem
%u	unsigned int	l. całowita. ze bez znaku
%o	int	ósemkowa
%x	int	szesnastkowa bez znaku, małe litery a..f
%X	int	j.w. będą duże litery A..F
%f	**float**/double	l. zmiennoprzecinkowa: [-]nnnn.mmmmm
%e	float/double	j.w. w postaci [-].nnnne[+-]mmm
%E	-"-	j.w. ze znakiem E
%G	-"-	jak %f lub %E, mniejsza liczba znaków
ZNAK lub ŁAŃCUCH		
%c	**char** (znak)	pojedynczy znak
%s	char * - wskaźnik **łańcucha**	łańcuch znaków, aż do napotkania bajtu zerowego \0
WSKAŹNIK		
%n	int *	Liczba dotychczas wysłanych znaków
%p	pointer - wskaźnik	argument w postaci wskaźnika, co zależy od modelu pamięci (segm:offs lub offs)- liczba 16-wa

Przykład 2 programu z printf()

```
/* Program printf2.c &/
#include <stdio.h>
void main(void)
{
  float       f = 0.521;
  int         i = -123;
  unsigned int u = 24;
  char        c = 'A';
  printf("Zmienna f = %f, a zmienna i jest rowna %d.\n", f, i);
  printf("Zmienna c = %c, a zmienna u jest rowna %u.\n", c, u);
  printf("Zmienna u w zapisie szesnastkowym jest rowna %x, \n", u);
  printf("natomiast w zapisie osemkowym jest rowna %o.", u);
}
```

Wyniki:

Zmienna f = 0.521000, a zmienna i jest rowna -123.
Zmienna c = A, a zmienna u jest rowna 24.
Zmienna u w zapisie szesnastkowym jest rowna 18,
natomiast w zapisie osemkowym jest rowna 30.

Przykład 3 z printf()

```
/* program printf2.c */
#include <stdio.h>
void main(void)
{
  float f = 0.521;
  printf("Zmienna f = %6.3f\n", f);
  printf("Zmienna f = %-6.3f\n", f);
  printf("Zmienna f = %06.3f\n", f);
  printf("Zmienna f = %+6.3f\n", f);
  getch();
}
```

Wynik

Zmienna f = 0.521
Zmienna f = 0.521
Zmienna f = 00.521
Zmienna f = +0.521

Przykład 3 z printf()

```
#include <stdio.h>
#define PI         3.1415926
#define PROMIEN    3.3
#define WYSOKOSC   44.4
main()
{
  double promien, wysokosc, objetosc;
  promien  = PROMIEN;
  wysokosc = WYSOKOSC;
  objetosc = PI * promien * promien * wysokosc;
  printf("\nObjetosc walca = %f", objetosc);
  printf("\nObjetosc walca = %E", objetosc);
  printf("\nObjetosc walca = %g", objetosc);
  printf("\nObjetosc walca = %15.10f", objetosc);
  printf("\nObjetosc walca = %25.20f", objetosc);
}
```

Wyniki:

Objetosc walca = 1519.010288
Objetosc walca = 1.519010E+03
Objetosc walca = 1519.01
Objetosc walca = 1519.0102875816
Objetosc walca = 1519.01028758159986900000

Funkcja getchar()

- **int getchar(void)**
- Funkcja odczytuje znak ze standardowego strumienia wejściowego (stdin) i zwraca go po dokonaniu konwersji bez rozszerzenia znakowego. Funkcja przy każdym wywołaniu podaje następny znak z wejścia lub EOF, gdy napotkała koniec pliku. W przypadku błędu lub końca zbioru wartością funkcji jest EOF. Stała symboliczna EOF jest zdefiniowana w nagłówku <stdio.h>.
  Jej wartoscią jest na ogół -1, ale e testach należy używać EOF.
  Zadaniem funkcji jest wprowadzenie pojedynczego znaku do pamięci komputera.
- Syntaktycznie postać funkcji jest następująca:
  c = getchar(); gdzie c jest typu char.
- Po każdym wywołaniu funkcja getchar pobiera ze strumienia znaków następny znak i wraca z jego zawartością. Zawartością zmiennej jest następny znak z wejścia.

Przykład programu

```
#include <stdio.h> /* przepisz wejscie na wyjście, wersja 1 */
main()
{  int c;  c = getchar(); /* przeczytaj znak */
 while (c != EOF) /* dopóki znak nie jest znakiem konca pliku (Ctrl Z) */
  { putchar(c);  /* wypisz przeczytany znak */
   c=getchar(); /* przeczytaj nastepny znak */
  }
}
```

Przykładowy wynik

Linia 1
Linia 2
By zakonczyc nacisnij Ctrl Z

Zmienne

- **Zmienne** służą do tego, aby zapamiętać sobie tymczasową wartość (którą potem można zmienić w każdej chwili według potrzeby).
  Dane w programie umieszczane są w postaci tzw. **zmiennych.**
- **Zmienna** jest to pewien fragment pamięci o ustalonym rozmiarze, który posiada własny identyfikator (nazwę) oraz może przechowywać pewną wartość, zależną od typu zmiennej.
- **Deklaracja zmiennych typ nazwa_zmiennej;**
  Oto deklaracja zmiennej o nazwie “wiek” typu “int” czyli liczby całkowitej:
  int wiek;
- Zmiennej w momencie zadeklarowania można od razu przypisać wartość (inicjalizować): **int wiek = 16;**
- W języku C zmienne deklaruje się na samym początku bloku (czyli przed pierwszą instrukcją).
- Język C nie inicjalizuje zmiennych lokalnych.
  Oznacza to, że w nowo zadeklarowanej zmiennej znajdują się śmieci - to, co wcześniej zawierał przydzielony zmiennej pamięci.
  Aby uniknąć ciężkich do wykrycia błędów, dobrze jest inicjalizować (przypisywać
- *wartość) wszystkie zmienne w momencie zadeklarowania.*

Zasięg zmiennej

- **Zmienne globalne -** obejmujące zasięgiem cały program – mogą być dostępne dla wszystkich funkcji programu
- Deklaruje się je przed wszystkimi funkcjami programu:
- Zmienne globalne, jeśli programista nie przypisze im innej wartości podczas definiowania, są inicjalizowane wartością 0.
- **Zmienne lokalne –** o zasięgu obejmującym pewien blok.
- Zmienne, które funkcja deklaruje do "własnych potrzeb" nazywamy **zmiennymi lokalnymi**.
- *Nasuwa się pytanie: "czy będzie błędem nazwanie tą samą nazwą zmiennej globalnej i lokalnej?". Otóż odpowiedź może być zaskakująca: nie. Natomiast w danej funkcji da się używać tylko jej zmiennej lokalnej.*
  Tej konstrukcji należy, z wiadomych względów, unikać.

```
int a=1; /* zmienna globalna */
int main()
{
int a=2; /* to już zmienna lokalna */
printf("%d", a); /* wypisze 2 */
}
```

Stałe

- Stała pozostanie taka sam w trakcie przebiegu programu. Nadaje się jej wartość w czasie pisania programu a nie w czasie działania.
- Stała, różni się od zmiennej tym, że nie można jej przypisać innej wartości w trakcie działania programu.
- Stałą deklaruje się z użyciem słowa kluczowego **const**:
- **const typ nazwa_stałej=wartość;**
- Dobrze jest używać stałych w programie, ponieważ unikniemy wtedy przypadkowych pomyłek a kompilator może często zoptymalizować ich użycie (np. od razu podstawiając ich wartość do kodu).

Przykład:

```
const  int  WARTOSC_POCZATKOWA=5;
int  i=WARTOSC_POCZATKOWA;
WARTOSC_POCZATKOWA=4; /* tu kompilator zaprotestuje */
int  j=WARTOSC_POCZATKOWA;
```

Stałe symboliczne

Stała symboliczna jest nazwą zastępującą ciąg znaków

#define NAZWA tekst

Np.

```
#define PI   3.1415926
#define MIEJSCOWOSC   Sosnowiec
#define WYNIK   printf(("Pole=%d\f %”,pole1)
#define WZOR1   (a*b)
```

Typy zmiennych, rzutowanie:

- Określając typ zmiennej przekazuje się kompilatorowi informację, ile pamięci trzeba zarezerwować dla zmiennej, a także w jaki sposób wykonywać na nim operacje.
- Jeśli potrzebujemy w pewnym miejscu programu innego typu danych to stosujemy **rzutowanie**.
  W takim wypadku stosujemy **konwersję** (**rzutowanie**) jednej zmiennej na inną zmienną

```
float a = 7.0 / 2; /*  float a = 7 / 2 da wynik 3; */
float b = (float)1000 * 1000 * 1000 * 1000 * 1000 * 1000;
    // rzutowanie
printf("%f\n", a);
```

- Istnieją **wbudowane** i **zdefiniowane przez użytkownika** typy danych.

Podstawowe typy zmiennych

- W języku C wyróżniamy następujące podstawowe typy zmiennych.
  - **char** – typ znakowy - jednobajtowe liczby całkowite, służy do przechowywania znaków;
  - **int**- liczby całkowite - typ całkowity, o długości domyślnej dla danej architektury komputera;
  - **float** – liczby rzeczywiste - typ zmiennopozycyjny (zwany również zmiennoprzecinkowym), reprezentujący liczby rzeczywiste (4 bajty);
  - **double** – liczby rzeczywiste - typ zmiennopozycyjny podwójnej precyzji (8 bajtów);
  - ***short*** - liczby całkowite krótkie
  - ***long*** - liczby całkowite długie
  - ***long double*** - liczby zmiennoprzecinkowe podwójnej precyzji długie
- Typ **int** przeznaczony jest do liczb **całkowitych**.

Liczby całkowite można zapisać na kilka sposobów

- System **dziesiętny**: np. **21; 13; 45; 156**

- System **ósemkowy** (oktalny): np. **010; 027**.
  (Cyfry 0 ..7)
  Cyfra 8 nie jest dozwolona.
  Jeżeli chcemy użyć takiego zapisu musimy zacząć liczbę od **0**.

- System **szesnastkowy** (heksadecymalny), np. **0x10, 0xff.**
  Aby użyć takiego systemu musimy poprzedzić liczbę ciągiem **0x**.
  Wielkość znaków w takich literałach nie ma znaczenia.

Typ float dla liczb zmiennoprzecinkowych

- Ten typ oznacza liczby zmiennoprzecinkowe czyli ułamki. Istnieją dwa sposoby zapisu:
- System **dziesiętny**, np. **10.256**, 3.14 (z kropką dziesiętna)
- System "**naukowy**" – wykładniczy, np. **6e2** (czyli 6 * 102)
  3.4e-3 (czyli 3.4 * 10(-3_
- Typ **double** dla liczb zmiennoprzecinkowych
- **Double** - czyli "podwójny" - oznacza liczby zmiennoprzecinkowe podwójnej precyzji.
  Oznacza to, że liczba taka zajmuje zazwyczaj w pamięci dwa razy więcej miejsca niż **float** (np. 64 bity wobec 32 dla float), ale ma też dwa razy lepszą dokładność.
- Domyślnie ułamki wpisane w kodzie są typu **double**.
  Możemy to zmienić dodając na końcu literę "f":
  1.4f (znaczy float), 1.4 (znaczy double)

Typ **char** dla znaków

- Jest to typ **znakowy**, umożliwiający zapis znaków ASCII. Może też być traktowany jako liczba z zakresu 0..255.
  Znaki zapisujemy w pojedynczych cudzysłowach (czasami nazywanymi apostrofami), by odróżnić je od łańcuchów tekstowych (pisanych w podwójnych cudzysłowach). Np. **‘A’, ‘a’, ‘!’, ‘$’**
- Pojedynczy cudzysłów ' zapisujemy **'\''**
  a null (czyli zero, które między innymi kończy napisy) tak: '**0**'

Inne **znaki specjalne**

- '\a' - alarm (sygnał akustyczny terminala)
- '\b' - backspace (usuwa poprzedzający znak)
- **'\f'** - wysuniecie strony (np. w drukarce)
- **'\r'** - powrót kursora (karetki) do początku wiersza
- **'\n'** - znak nowego wiersza
- '\"' - cudzysłów
- '\'' - apostrof
- '\\' - ukośnik wsteczny (backslash)
- **'\t'** - tabulacja pozioma
- '\v' - tabulacja pionowa
- '\?' - znak zapytania (pytajnik)
- **'\ooo'** - liczba zapisana w systemie oktalnym (ósemkowym), gdzie 'ooo' należy zastąpić trzycyfrową liczbą w tym systemie
- **'\xhh'** - liczba zapisana w systemie heksadecymalnym (szesnastkowym), gdzie 'hh' należy zastąpić dwucyfrową liczbą w tym systemie, np. ‘\xAFF’

Specyfikatory – signed, unsigned, short, long

- **Specyfikatory** - słowa kluczowe, które postawione przy typie danych zmieniają jego znaczenie.
- **signed** – liczba ze znakiem i **unsigned** – nieujemna (bez znaku)
- **short i long** są wskazówkami dla kompilatora, by zarezerwował dla danego typu mniej lub więcej pamięci. Mogą być zastosowane do dwóch typów: **int** i **double** (tylko long), mając różne znaczenie.
- Jeśli przy short lub long nie napiszemy, o jaki typ nam chodzi, kompilator przyjmie wartość domyślną czyli **int**.

Przykłady:

```
signed char a;    /* zmienna a przyjmuje wartości od -128 do 127 */
unsigned char b;  /* zmienna b przyjmuje wartości od 0 do 255    */
unsigned short c;
unsigned long int d;
```

Wyrażenia i operatory

- Wyrażenia są zapisami operacji, jakie mają być wykonane. **Wyrażenie** składa się ze stałych, zmiennych i operatorów. **Kolejność** wykonywania działań określają **nawiasy** i **priorytet operatorów**.
- Najważniejsze **operatory**:

Operator	**Działanie**	**Przykład**	**Wynik**
*	Mnożenie	a=2*5; /*1 0 */	
/	dzielenie	a=a/10.0;	/* 1.0 */
+	dodawanie	a=a+14;	/* 15 */

- Odejmowanie a=a-5; // 10

% reszta z dzielenia c=10 % 3; // wynik = 1

Operatory

- Do wykonania obliczeń zmieniających dane w informacje, niezbędne są **operatory**.
- **Operator** to symbol mówiący komputerowi, jak ma przetwarzać dane.
- **Operatory**, z punktu widzenia liczby argumentów dzielimy na dwuargumentowe i jednoargumentowe.

a op b

- op jest operatorem **2-argumentowym**, np.

a+b; a*b; a/b; a%b;

op a

- op jest operatorem **jednoargumentowym**, np.

-a; -(a+b); !a

Operatory można pogrupować wg cech funkcjonalnych na grupy:

- operatory arytmetyczne: **+, -, *, /, %** *(dzielenie modulo - reszta)*
- operatory porównania – relacyjne: **==, !=, <, >, <=. >+**
- operatory logiczne: **&&** (and), **||** (or), **!** (not)
- operatory bitowe
- operatory przypisania: **=**, **+=** (a=+b → a=a+b), **-=**, ***=**, **/+**, **%=**…
- operatory unarne inkrementacji i dekrementacji: ++, -- (np. x++, x--, ++x, --x)
- operatory rozmiaru: sizeof(obiekt)
- operatory konwersji: **(nazwa typu) wyrażenie**
- operator warunkowy: **Op1 ? Op2: Op3;**
- operator przecinkowy: **Op1, Op2, ..., Opn**
- operatory wskazywania ->

Przykład programu do wydania reszty - ilość banknotów 20-, 10-, 5- i 1- dolar.

```
/* change.c */
#include <stdio.h>
main()
{
int amount,twenties, tens, fives, ones, r20, r10;
printf("Wprowadz kwote do wydania: "); /* Wprowadzenie kwoty do wydania */
scanf("%d", &amount);
twenties= amount/20;                    /* banknoty 20-dolarowe */
r20=amount % 20;  /* r20 reszta pozostała po 20-ch*/
tens= r20/10;         /* ilość banknotów 10-dolarowych */
r10=r20 % 10;         /* r10 reprezentuje resztę po 10-ch */
fives= r10 / 5;       /* banknoty 5-dolarowe */
ones = r10 % 5;       /* reszta - dolarówki */
putchar('\n');
printf("By przekazac %d wydaj banknoty: \n", amount);
printf("%d dwudziestki\n", twenties);
printf("%d dziesiatki\n", tens);
printf("%d piatki \n", fives);
printf("%d pojedyncze (s)\n", ones);
}
```

Operatory relacyjne (relacji i porównania)

- W języku C i C++ występują następujące **operatory porównania**:

Operator	Znaczenie
<	Mniejszy (relacja)
<=	Mniejszy lub równy
>	Większy
>=	Większy lub równy (relacja)
==	Równy (porównanie)
!=	Nierówny (porównanie)

- Wykonują one odpowiednie porównanie swoich argumentów i zwracają jedynkę jeżeli warunek jest spełniony lub zero jeżeli nie jest.
- Operatory **relacji**: **>** **>=** **<** **<=** (mają ten sam priorytet)
- Operatory **przyrównania**: **==** (równe) **!=** (różne) (priorytet niższy)

Przykłady wyrażeń z operatorami **porównania**

k=1; m=2; n=3;

Wyrażenie	Opis	Wartość
m<n	Prawda	1
(k+m)>=n	Prawda	1
(m-k)>n	Fałsz	0
n>=3	Prawda	1
k != m	Prawda	1
k == 2	Fałsz	0
(m+2)<=k	Fałsz	0

```
#include <iostream.h>
int main() {
int a, b, wyn, k=1,  m=2, n=3;
a=10; b=3; wyn=a>b;
cout << (m<n) << endl;
cout << ( (k+m)>=n ) << endl;
cout << ( (m-k)>n  )  << endl;
cout << ( n>=3 ) << endl;
cout << ( k != m ) << endl;
cout << ( k == 2 ) << endl;
cout << ( (m+2)<=k ) << endl;
cout << "a = " << a << "  b = " <<b <<  "
wyn = a>b = " << wyn;
return 0;
}
```

Wyniki:

1
1
0
1
1
0
0
a = 10 b = 3 wyn = a>b = 1

Podejmowanie decyzji – operacje logiczne

- **Operacje logiczne**:

  ! - **Negacja – not logiczne nie**

  && - **Koniunkcja - and – logiczne i**

  || - **Alternatywa - or – logiczne lub**

- Przykład:

```
int main() {
int a, b, c, d; a=1; b=0; c = a&&b; // →0;
d = a||b;  // →1
cout  << c << endl <<  d;  return 0; }
```

Wynik:

0

1

Operator przypisania i wieloznakowe operatory przypisania

- Instrukcją przypisania jest trójka elementów zapisana jako:
  identyfikator operator_przypisnia wyrażenie
  Identyfikator *to np. nazwa zmiennej.* ***Wyrażenie*** *– dowolne wyrażenia*

Operator	Zapis w C	Znaczenie
=	a=b;	a=b // przypisanie zwykłe
+=	a=+b;	a-a+b; // wieloznakowe operatory przypisania
-=	a=-b;	a=a-b;
***=**	a=*b;	a=a*b;
/=	a=/b;	a=a/b;
%=	a=%b;	a=a%b;
&=	a=&b;	a=a&b;
!=	a=!b;	a=a!b;
~=	a=~b;	a=a~b;

Wieloznakowe operatory przypisania

- **Zapis skrócony postaci**

  a #= b;

  gdzie # jest jednym z operatorów:

  +, -, *, /, &, |, ^, << , >>.

  Ogólnie zapis

  a #= b;

  jest równoważny zapisowi

  a = a # b;

```
//  Program wielop1.cpp
#include <iostream.h>
#include <conio.h>
int main()  {
int a = 1, b;
cout << "Operatory przypisania wieloznakowe\n";
cout << "a= " << a << endl;
a += 5;     /* to samo, co a = a + 5; czyli 6 ;       */
cout << "a+=5 czyli  a=a+5 = " << a << endl;
a /= a - 3; /* to samo, co a = a / (a - 3);  czyli 6/(6-3)=3
       */
cout << "a/=a-3 czyli a=a/(a-3) =  " << a << endl;
a +=1;    // 4
cout << "a+=1 czyli a=a+1 = " << a << endl;
a %= 2;     /* to samo, co a = a % 2;  4%2     */
cout << "a%2 czyli a=a%2 = " << a << endl << endl;;
a=10;
cout << "Nowe a = " << a << endl;
b= (a++, a+=10); // 21
cout << "b= (a++, a+=10) = " << b; // 21
cout << " bo a=a+10 = 20 i b=a++ = a+1 =21" << endl;
getch();
}
```

Wyniki

Operatory przypisania wieloznakowe
a= 1
a+=5 czyli a=a+5 = 6
a/=a-3 czyli a=a/(a-3) = 2
a+=1 czyli a=a+1 = 3
a%2 czyli a=a%2 = 1

Nowe a = 10
b= (a++, a+=10) = 21 bo a=a+10 = 20
i b=a++ = a+1 =21

Operatory unarne (jednoargumentowe)

- **Operatory unarne**
  - **inkrementacji ++** // zwiększenie o 1, typu
    x=a++; // postinkrementacja. Najpierw przypisanie do x wartości a, potem zwiększenie a o 1

    x= ++a; // preinkrementacja
    Najpierw zwiększenie a o 1 , potem przypisanie a do x
  - **dekrementacji --** // zmniejszenie o 1, typu
    x=i- - // postdekrementacja (przypisanie i zmniejszenie a)
    x=--i // predekrementacja (zmniejszenie a i przypisanie)
  - **minus jednoargumentowy** np. a=-b;

Przykładowy program w C++

```
#include <iostream.h>
#include <conio.h>
main() {
int a=1, x, b;
cout <<"Operatory zwiekszania i zmniejszania" << endl;
cout << "a= " << a << endl;
x=a++;
cout << "x=a++ " << endl;
cout << "x= " << x << " a= " << a << endl;
x=++a;
cout << "x=++a " << endl;
cout << "x= " << x << " a= " << a << endl;
b=10;
cout << "\nb=" << b <<  endl;
cout << "Wydruk b++ = " << b++ << endl;
cout << "Wartosc b po wydruku =" << b << endl;
cout << "++b = " << ++b << endl;
getch();
}
```

Wyniki

Operatory zwiekszania i zmniejszania
a= 1
x=a++
x= 1 a= 2
x=++a
x= 3 a= 3

b=10
Wydruk b++ = 10
Wartosc b po wydruku =11
++b = 12

Program w C – operator jednoargumentowy

```
#include <stdio.h>
#include <conio.h>
main()
{
  int   i=5 ;
  printf("\n----------------");
  printf("\n  i = %i ",   i  );
  printf("\n  i = %i ", ++i  );
  printf("\n  i = %i ",   i  );
  printf("\n  i = %i ",   i++);
  printf("\n  i = %i ",   i  );
  printf("\n  i = %i ", --i  );
  printf("\n  i = %i ",   i  );
  printf("\n  i = %i ",   i--);
  printf("\n  i = %i ",   i  );
  printf("\n  i = %i ",  -i  );
  printf("\n----------------");
  getch();
}
```

Wyniki

```
----------------
  i = 5
  i = 6
  i = 6
  i = 6
  i = 7
  i = 6
  i = 6
  i = 6
  i = 5
  i = -5
----------------
```

Instrukcja **printf("\n i = %i", ++i);**
jest równoważna sekwencji instrukcji:
i=i+1; printf("\n i = %i",i);
Instrukcja **printf("\n = %%i", i++);**
jest równoważna sekwencji instrukcji:
printf("\n i =%i",i); i=i+1;.

Rzutowanie, operator konwersji

Zadaniem **rzutowania** jest konwersja danej jednego typu na daną innego typu.

Konwersja może być niejawna (domyślna konwersja przyjęta przez kompilator) lub jawna (podana explicite przez programistę).

Przykłady konwersji niejawnej:

```
int i = 42.7;          /* konwersja z double do int */
float f = i;           /* konwersja z int do float */
double d = f;          /* konwersja z float do double */
unsigned u = i;        /* konwersja z int do unsigned int */
f = 4.2;               /* konwersja z double do float */
i = d;                 /* konwersja z double do int */
```

Operator konwersji (rzutowania) pozwala na jawne przekształcenie typów danych

Zapis operatora konwersji

(nazwa typu) wyrażenie

Np. **int m, n; n=10; m=n+(int) 1.2345;**

Wyrażenie jest przekształcane wg reguł dla typu określonego przez nazwę typu.

Operator rzutowania służy do jawnego wymuszenia konwersji

```
/* Program konwn.c */
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define NL printf("\n");
int main()
{
int i; float f; double d; unsigned u;
i = 42.7; printf("%i",i); NL;
f = i; printf("%f",f); NL;
d=f; printf("%f",d); NL;
u=1; printf("%u",u); NL;
f=4.2; printf("%f",f); NL;
i=d; printf("%d",i); NL;
getch();
}
```

Wyniki

```
42
42.000000
42.000000
42
4.200000
42
```

Konwersja niejawna

```
int i = 42.7;       /* konwersja z double do int  42 */
float f = i;        /* konwersja z int do float  42.000*/
double d = f;       /* konwersja z float do double 42.00 */
unsigned u = i;     /* konw. z int do unsigned int  42*/
f = 4.2;            /* konwersja z double do float  4.2*/
i = d;              /* konwersja z double do int   42*/
```

Konwersja jawna

/* Przykład z poprzedniej strony */

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define NL printf("\n");

int main()
{
int m, n;  n=10;
m=n+ (int)1.2345;
printf("%d",m); /* wynik 11 */
getch();
}
```

```c
/*  Program p35.c   ( operator konwersji )    */
#include <stdio.h>
#include <math.h>
#include <conio.h>
int main()  {
  char  c = 'k';
  int    i = 70;
  float  f = 345.6789;
  printf("\n Wartosci poczatkowe:  ");
  printf("\n c = \'%c\';  i = %i;  f = %8.4f;", c ,i, f);
  printf("\n-------------------------------------");
  printf("\n Konwersja          Wynik          ");
  printf("\n-------------------------------------");
  printf("\n (int)    c         %i ", (int)    c     );
  printf("\n (float)  c         %f ", (float)  c     );
  printf("\n (double) i         %f ", (double) i     );
  printf("\n (char)   i         %c ", (char)   i     );
  printf("\n (double) i+2000     %f ", (double) i+2000 );
  printf("\n (int)    f         %i ", (int)    f     );
  printf("\n sqrt((double) c)    %f ", sqrt((double) c));
  printf("\n-------------------------------------");
  getch();
  return 1;
}
```

Wyniki

```
 Wartosci poczatkowe:
 c = 'k';  i = 70;  f = 345.6789;
-------------------------------------
 Konwersja          Wynik
-------------------------------------
 (int)    c          107
 (float)  c          107.000000
 (double) i          70.000000
 (char)   i          F
 (double) i+2000     2070.000000
 (int)    f          345
 sqrt((double) c)    10.344080
-------------------------------------
```

Operator warunkowy ?

- **Operator warunkowy ?**
- Wyrażenie z użyciem operatora warunkowego:
  Op1 ? Op2: Op3;
  wyrażenie_warunkowe ? wyrażenie_na_tak: wyrażenie_na_nie;
- Operator warunkowy (?:) pozwala zapisać w bardziej zwartej formie pewne konstrukcje programowe wymagające użycia instrukcji if.
- Np.
  z = (a>b)? a:b; /* z=max(a,b) */

Program z zastosowaniem operatora warunkowego - funkcja max

```
/* C++ - Operator warunkowy - funkacja max */
#include <iostream.h>
void main()
{
 int max(int, int); // deklaracja - zapowiedz funkcji max
 int a, b, m;   char c;
 cout << "Program wyznacza max 2 liczb\n";
 cout << "Podaj 2 liczby calkowite oddzielone spacja: ";   cin >> a >> b;
 m=max(a,b);  // wywolanie funkcji max
 cout << "Max liczb " << a << " i " << b << " wynosi " << m << endl;
 cout << "\nWprowadz jakis znak i nacisnij Enter  ";   cin >> c;
}
// Definicja funkcji  max
int max(int i, int j)
{
 return i > j? i: j;
}
```

Operator przecinkowy/wyliczeniowy, (koma)

Wyrażenie wyliczeniowe ma postać:

Op1, Op2, ..., Opn

gdzie Opi, (i=1..n) są operandami dowolnych typów.

Opracowanie tego wyrażenia polega na obliczeniu wartości kolejnych operandów, począwszy od Op1, a skończywszy na Opn.

Typ i wartość całego wyrażenia określa operand Opn.

W C++ wynik wyrażenia wyliczeniowego jest **l-wartością**.

Poprawne jest więc wyrażenie: (a+=b, c-=d, c*=a)++.

Przykład:

int n=10, i=2, k=4, wynik; wynik=(n-=i, k+=2, i*=5):

W wyniku obliczenia powyższego wyrażenia zmienne uzyskają następujące wartości:

n = n-i = 10-2=8 k = k+2 = 4+2=6 i = i*5 = 2*5=10 (ostatni operand)

wynik = wartość ostatniego operandu, czyli **10**

```
// Program operprz1.cpp
#include <iostream.h>
#include <conio.h>
main()
{
int n=10, i=2, k=4, wynik;
wynik=(n-=i, k+=2, i*=5);
cout << wynik; // 10
getch();
}
```

Operator przecinkowy – przykładowy program

- Operator przecinkowy stosowany jest m.in. w instrukcjach for, np. do sterowania równoległego 2 indeksami, a także w wywołaniach funkcji.
  Stosując wyrażenia wyliczeniowe w wywołaniach funkcji należy pamiętać o użyciu nawiasów, np.:
  func(i, (i+=k, k++), j); - funkcja otrzymuje tylko 3 parametry.

```
/*Program  opprz2.c  operator przecinkowy )   */
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define RS 180.0/M_PI
int main()
{
  int  i, j;
  double kat_rad;
  printf("Wartosc M_PI = %f  RO[st]= %f\n",M_PI, RS);
  printf("\n    i   j[st]  sqrt(i)    sin(j)");
  printf("\n---------------------------------- ");
  for (i=0, j=90; j>=0; i++, j--)
    {
     kat_rad=(double)j*(M_PI/180.0);
     printf("\n   %2i    %2i    %8.5f    %6.5f",i,j,
     sqrt((double) i), sin(kat_rad));   /* argument w radianach */
    }
  printf("\n---------------------------------- ");
  getch();
 return 0;
}
```

WYNIKI

Wartosc M_PI = 3.141593 RO[st]= 57.295780

i	j[st]	sqrt(i)	sin(j)
0	90	0.00000	1.00000
1	89	1.00000	0.99985
2	88	1.41421	0.99939
3	87	1.73205	0.99863
4	86	2.00000	0.99756
5	85	2.23607	0.99619
6	84	2.44949	0.99452
7	83	2.64575	0.99255
8	82	2.82843	0.99027
9	81	3.00000	0.98769
10	80	3.16228	0.98481
11	79	3.31662	0.98163
12	78	3.46410	0.97815
13	77	3.60555	0.97437
14	76	3.74166	0.97030
15	75	3.87298	0.96593
16	74	4.00000	0.96126
17	73	4.12311	0.95630
18	72	4.24264	0.95106
19	71	4.35890	0.94552
20	70	4.47214	0.93969
21	69	4.58258	0.93358
22	68	4.69042	0.92718
.....			
89	1	9.43398	0.01745
90	0	9.48683	0.00000

Biblioteki, procedura, funkcja

- **Biblioteki (moduły)** - zapisane są w nich różne funkcje i procedury do użycia w programie.
- **Funkcja** - jest to podobnie jak procedura, pewien wyraz, który nie dość że coś wykona to również zwróci nam rezultat tego co zrobił, *przykładowymi funkcjami są:* pierwiastkowanie, sinus, cosinus. Zwracaną wartością niekoniecznie musi być liczba może to być również inny rodzaj zmiennych.

Typy całkowite

czyli zbiory, których elementami są wyłącznie liczby **całkowite**.

Nazwa typu	Zakres	Ilość bajtów
unsigned char	**0..255**	1
short int	**-128..127**	1
unsigned short	**0..65535**	2
int	**-32768..32767**	2
long	**-2147483648..2146483647**	4
unsigned long	**0..4294967295**	4

Typy **rzeczywiste**

Nazwa typu	Zakres	Znaczące cyfry - liczba cyfr	Format - rozmiar
float	**-3.4E38..-3.4E-38, 3.4E-38 .. 3.3E38**	11-12	**4** Bajty
double	**1.7E-308.. 1.7E308**	15-16	**8 B**
long double	**3.4E-4932..1.1E4932**	19-20	80-bitowy **10 B**

Typy ***bool***, **char**, ***string***

- TYP **LOGICZNY** - ***bool***- typ ten może przyjmować jedynie dwie wartości:
  true (**prawda**) – **1** lub false (**fałsz**) **- 0**
  #include <stdbool.h>
  int main() { bool b = false; b = true; }
- TYP **ZNAKOWY** - ***CHAR*** - typ ten przyjmuje dowolny pojedynczy znak o kodach ASCII (0..255)
  np. znak **'A'** czy **'!'**
- W C łańcuchy reprezentuje się jako tablice znaków
  char slowo[dlug] a operacje na nich wykonuje z użyciem wskaźników.
- W **C++** oprócz tradycyjnych ciągów znaków w stylu C istnieje w bibliotece standardowej **klasa std::string.**
  Ukrywa ona niewygodne aspekty używania napisów w stylu C:
  zarządzanie pamięcią, określanie długości, łączenie napisów, wstawianie, usuwanie i inne manipulacje na napisie.
  Dodatkowo *pozbyto się problemu znaku kończącego* - znak o kodzie **\0**.

Najważniejsze typy dostępne w C i C++

Nazwa	Znaczenie	Zakres wartości	Przykład
int (short int)	liczba **całkowita** ze **znakiem**	-3276S..+32767	-14574
float	liczba **rzeczywista**	-1.1*10E38..3.1E1038	1.23245el7
char	**znak**	znaki o kodach 0.255	'a'
string	**napis** (łańcuch, ciąg znaków)	ciąg do 255 znaków	‘’Napis’’
bool	**wartość logiczna**	prawda (true) lub fałsz (false)	false
unsigned int	**słowo**	0..65535	56412
char	**bajt**	-128..127	127
unsigned char	**bajt**	0.255	127
long	długa liczba całkowita ze znakiem	-214743364S..+2147433647	-1936734234
double	długa liczba rzeczywista	-1.7E308 ..1.7*E308	-1.8e+234
long double	bardzo długa liczba rzeczywista	-3.4E-4932..1.1E4932	4.5e2345

Wyrażenia i operatory

- Wyrażenia są zapisami operacji, jakie mają być wykonane. **Wyrażenie** składa się ze stałych, zmiennych i operatorów. **Kolejność** wykonywania działań określają **nawiasy** i **priorytet operatorów**.
- Najważniejsze **operatory**:

Operator	**Działanie**	**Przykład**	**Wynik**
*	Mnożenie	a=2*5; /*1 0 */	
/	dzielenie	a=a/10.0;	/* 1.0 */
+	dodawanie	a=a+14;	/* 15 */

- Odejmowanie a=a-5; // 10

% reszta z dzielenia c=10 % 3; // wynik = 1

Instrukcja przypisania

- **Instrukcja przypisania** - służy do przypisania zmiennym wartości. Jest postaci: zmienna:=wyrażenie;
- Przykład:

```
/* Program przypis1.c */
#include <stdio.h>
#include <conio.h>
/* zmienne globalne */
int a; float   b, c;
char napis[]="Hello World";  /*  deklaracja zmiennej napis typu lancuch znakow */
int main() {
      clrscr();
      a=5;  b=2.15;  c=a+b;
      puts (napis);
      printf("a=%d  b=%f   c=%f",a , b,c);
      getch();
      return 0;
}
```

Instrukcje wejścia

- Do **wczytywania danych** stosuje się instrukcje **getchar(), gets(), scanf()** w C i C++ oraz **cin >>** w C++
- Funkcja **gets()** służy do wczytania pojedynczej linii.
  Np. char linia[80]; gets(linia);
- **getchar** umożliwia wprowadzenie pojedynczego znaku
  Np. char znak; znak=getchar(); putchar(znak);
- **scanf()** to uniwersalna funkcja do wprowadzania wszelkiego typu informacji. Składa się z łańcucha sterującego i listy danych.
  scanf(„lancuch_sterujacy”,lista_danych);
  Np. float ilosc; scanf(„%f”,&ilosc);
  Łańcuch sterujący zawiera specyfikatory formatowania – jak będą interpretowane dane wejściowe.
- Specyfikatory formatowania:
  %d – wprowadza liczbę całkowitą, **%u** – liczba bez znaku, **%f** – liczba float, **%e** – liczba w systemie wykładniczym, **%g** – liczba dziesiętna w najkrótszym zapisie, **%c** – dana znakowa char, **%s** – łańcuch znaków, **%o** – liczba ósemkowa, **%x** – liczba szesnastkowa

Przykład programu z instrukcjami **we/wy**

```
/* Program  wczytuj1.c */
#include <stdio.h>
#include <conio.h>
int main() {
 int a, b, c;  char nazw[15];
      clrscr();
       puts("Wprowadzanie danych w jezyku C");
      puts("Podaj wartosci 3 zmiennych calkowitych oddzielone spacja:  ");
      scanf("%d  %d  %d", &a,&b,&c);
      printf("Wprowadzone liczby: a= %d   b= %d   c= %d",a,b,c);
      printf("\nPodaj swoje nazwisko: ");
      scanf("%s",nazw);
      printf("\nNazywasz sie %s ",nazw);
      getch();
      return 0;
}
```

Wersja druga programu wczytywania danych z **gets**

```
/* Program  wczytuj2.c */
#include <stdio.h>
#include <conio.h>
int main() {
 int a, b, c;  char nazw[15];
       clrscr();
        puts("Wprowadzanie danych w jezyku C");
        puts("Podaj swoje nazwisko   ");
        gets(nazw);
       puts("Podaj wartosci 3 zmiennych calkowitych oddzielone spacja:  ");
       scanf("%d  %d  %d", &a,&b,&c);
       printf("Wprowadzone liczby: a= %d   b= %d   c= %d",a,b,c);
       printf("\nNazywasz sie %s ",nazw);
       getch();
       return 0;
}
```

Program na pole i obwód koła

```
/* program kolo.c  oblicza obwod
      i pole koła o promieniu podanym */
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define  PI 3.1415926
#define  NL printf("\n");
int main()
{
       /* deklaracja zmiennych */
       const double pi=3.1415926;
       float r , o, p;
      /* promień koła, obwod, pole */
      clrscr;
      puts("Program oblicza obwod i pole kola");
       printf("PI=%f  pi=%f  M_PI=%f",PI, pi,
      M_PI);
      printf("\nPodaj promien kola: ");
      scanf("%f",&r);/* wczytaj r z klawiatury */
      NL;  /* nowa linia */
      o=2.0*PI*r; /* obwod */
      p=PI*r*r;     /* pole */
      printf("Obwod kola o promieniu %.3f = %8.3f",r,o );
      NL;
      printf("Pole  tego kola = %8.3f", p);
      /* wypisz pole koła */
       NL;
      getch();      /* czeka na naciśniecie klawisza */
      return 0;
      }
```

Instrukcja złożona - grupująca

Instrukcję złożoną można porównać z nawiasami w arytmetyce.

Umożliwia zgrupowanie wielu instrukcji i traktowanie jako jednej.

Czasem jest niezbędna przy korzystaniu z rozkazów strukturalnych takich jak: **if … then else** czy do **…while**

Ma postać:

```
{
 {ciąg_instrukcji_oddzielonych_średnikami}
}
```

Np.

```
{
 a=3;
b=10;
}
```

Trzy powyższe instrukcje są traktowane jako jedna.

Preprocesor, dyrektywy, funkcje biblioteczne C

Preprocesor przetwarza wiersze programu rozpoczynające się znakiem **#**.

Taki wiersz nazywamy **dyrektywą** preprocesora.

Podstawową **dyrektywą** jest #include, umożliwiająca dołączenie do programu pliku o podanej nazwie.

Ma 2 postacie**: #include <nazwa>** i **#include „nazwa”**

Nazwa w nawiasach kwadratowych oznacza **plik nagłówkowy** dostępny w systemie C, natomiast w cudzysłowie plik zdefiniowany przez użytkownika.

W C istnieje duża biblioteka funkcji standardowych, czyli dostępnych bezpośrednio w systemie. Zawarte są w tzw. modułach.

Moduły biblioteczne dołącza się przez instrukcję **#include <nazwa>**.

Standardowo dołączamy s**tdio.h** w C a **iostream.h** w C++

Procedura **clrscr()** zawarta w module conio.h umożliwia wyczyszczenie ekranu.

W **math.h** zdefiniowane jest np. **M_PI** (czyli PI)

Dyrektywa **#define** , np. #define NMAX 20 definiuje stałą NMAX o wartosci 20.

Operatory relacyjne

- Do ustalania warunków przydają się także inne operatory, tzw.
  Operatory relacyjne
  = = równy
  < mniejszy
  > większy
  <= mniejszy lub równy
  >= większy lub równy
  != nierówny

Instrukcja warunkowa if... then... [else...]

- Pozwala na wykonanie lub zaniechanie wykonania pewnych czynności, *w zależności od konkretnego warunku logicznego.*
- Instrukcja ma następującą **składnię**:
  if (warunek) instrukcja;
  np. if (x>30) y=20;
  If (warunek)
  {instrukcja_1; instrukcja_2; … instrukcja_N;}
  np. if (min <x) {min=x; nr=i;}
- Instrukcja ta sprawdza czy jest spełniony warunek postawiony po „if", jeżeli tak to wykonywana jest instrukcja lub ciąg instrukcji w nawiasach { }
- Można jeszcze też użyć słowa **else** oznaczającego "w przeciwnym wypadku"
  If (warunek) instrukcja_1; else instrukcja_2;
  np. if (a>0) c=b+2; else c=b-3;
  If (warunek)
  {instrukcja_1; instrukcja_2; … instrukcja_N;}
  else {instrukcja_A; instrukcja_B; … instrukcja_Z;}

Przykłady programów z if

```
/* Program If1. c */;
#include <iostream.h>
#include <conio.h>
int main()
{
        int a;
        cout << "Podaj liczbe calkowita >0  ";  cin >> a;
        if  (a<0) return 1;  //wyjscie  awaryjne z systemu
        cout << "OK"; getch();  return 0; // wyjscie normalne
}
```

```
/* Program If2. c */;
#include <iostream.h>
#include <conio.h>
int main()
{
int a;
cout << "Podaj liczbe calkowita > 0 ==> ";  cin >> a;
if  (a<0) cout << "a < 0";
else cout << "a >=0 ";
getch();
return 0;
}
```

Operatory logiczne: **&&** (and), || (or), ! (**not)**

- Mogą one przyjmować wartości
  true czyli 1 lub 0 czyli **false**
  Za ich pomocą możemy skonkretyzować nasz warunek.
  Np.
  if (warunek1) && **(**warunek2) instrukcja;
- Operator &&(**koniunkcja**) sprawia, że instrukcja zostanie wykonana tylko w przypadku gdy spełnione są oba warunki.
  Gdybyśmy zastąpili go operatorem || (**alternatywa**), to wystarczyłby tylko jeden spełniony.
- Drugi przypadek
  if (! Warunek) instrukcja;
  W tym przypadku instrukcja będzie wykonana jeśli warunek nie jest spełniony (ma wartość **false**).
  Do tego właśnie służy operator ! (czyli **negacja**).

```
/* program rowkwadr.cpp    */
#include <iostream.h>
#include <conio.h>
#include <math.h>
int main()
{
float a, b, c, delta;
clrscr();
cout << "Program oblicza pierwiastki rownania kwadratowego." << endl;
cout << "Podaj wspolczynniki: " << endl;
cout <<  "a= " ; cin >>a;
cout << "b= ";  cin >> b;
cout << "c= ";  cin >> c;
delta=b*b-4*a*c;
if (a !=0)
{
  if (delta<0)   cout << "Delta ujemna - brak rozwiazan.\n" ;
  else
  if (delta==0)  cout << "x= " << (-b+sqrt(delta))/(2*a);
  else
  if (delta>0)
    {
     cout << "x1= " << (-b+sqrt(delta))/(2*a) << endl;
     cout << "x2= " << (-b-sqrt(delta))/(2*a) << endl;
   }
}
else
{
   if ((a==0) && (b==0)) cout << "Rownanie ma nieskonczenie wiele rozwiazan.";
   else
   if ((a==0) && (b!=0))
     {
        cout << "Rwnanie o podaneych wspolczynnikach jest liniowe.\n";
        cout << "jego pierwiastek jest rowny: " << -c/b << endl;
     }
 }
 getch();
 return 0;
}
```

switch - instrukcja wyboru

- Instrukcja **switch** jest wykorzystywana kiedy zachodzi konieczność podjęcia kilku decyzji, gdy wykonanie różnych części programu jest uzależnione od stanu pewnej zmiennej.
- Podstawą podjęcia decyzji jest wyrażenie typu całkowitego, znakowego lub logicznego
- Instrukcja **switch** ma postać:

```
switch (zmienna lub wyrażenie) {
    case wartosc_1: Instrukcje_1; break;
    // break nie musi być
    .....................
    case wartosc_N: Instrukcje_N; break;
    default  Instrukcje_domyslne;
}
```

Instrukcja wyboru switch .. case – przykład programu

```
// Program Miesiace.cpp  - podaje nazwe miesiaca na podstawie jego numeru
#include <iostream.h>
#include <conio.h>
int main()
{
int miesiac; // numer miesiąca w roku
cout <<  "Podaj numer miesiaca w roku: ";
cin>>  miesiac;
switch (miesiac)
{ // switch
case 1: cout << "Styczen"; break;
case 2: cout << "Luty"; break;
case 3: cout << "Marzec"; break;
case 4: cout << "Kwiecien"; break;
case 5: cout << "Maj"; break;
case 6: cout << "Czerwiec"; break;
case 7: cout << "Lipiec"; break;
case 8: cout << "Sierpien"; break;
case 9: cout << "Wrzesien"; break;
case 10: cout << "Pazdziernik"; break;
case 11: cout << "Listopad"; break;
case 12: cout << "Grudzien"; break;
default: cout << "Numer nie poprawny";
} // switch
 getch();
 return 0;
}
```

Instrukcje iteracyjne Organizacja obliczeń cyklicznych

Instrukcje iteracyjne służą do wielokrotnego wykonywania pewnych sekwencji instrukcji i zazwyczaj są one nazywane po prostu **pętlami.**

Instrukcja while - dopóki Warunek wykonuj Instrukcję;

Składnia: while (wyrazenie) instrukcja;

Instrukcja do while – powtarzaj Instrukcję dopóki Warunek

Składnia: do instrukcja while (wyrazenie);

Instrukcja for: np. dla i od 1 do N - dla i=W1 do W2 wykonuj Instrukcję

For (i=W1; i<W2; i++) Instrukcja;

Składnia:

for (wyrażenie1; wyrażenie2; wyrażenie3) instrukcja;

for (wyrażenie1; wyrażenie2; wyrażenie3) { lista instrukcji}

Instrukcję for stosuje się w przypadkach, gdy z gór można określić liczbę wykonań pętli.

Instrukcja while

- Instrukcja ma postać:
  while (warunek) instrukcja**;**
- Powoduje wykonywanie instrukcji tak długo, dopóki spełniony jest warunek
- **Warunek** jest najczęściej dowolnym wyrażeniem porównania, które powinno w wyniku dać wartość logiczną (**true** lub **false**).
  Można posłużyć się tutaj **operatorami relacyjnymi**: ==, !=, <, >, <=, >=
- Instrukcja wykonywana jest tak długo, dopóki warunek ma wartość **true**.

Przykład programu z while – suma 6 podanych liczb

```
//  Program Suma6.cpp; Oblicza sumę 6 podanych liczb
#include <iostream.h>
#include <conio.h>
int main()
{
 int i;   float suma, x;
 i=1; // Nadanie wartości początkowej i - licznik
 suma=0; // Nadanie wartości początkowej suma
 cout << "Program oblicza sume podanych 6 liczb " << endl;
 while (i<=6)
  { // while
      cout << i << "   Podaj liczbe x=";       cin >> x;       suma=suma+x;
      cout << "\Aktualna suma=" << suma << endl;
      i++;
  } // while
cout << "\nSuma koncowa =" << suma << "\nNacisnij Enter" << endl;
getch();
return 0;
}
```

Program z do … while

```
// Program ilosc.cpp wyznacza długość ciągu liczb zakończonych 0
#include <iostream.h>
#include <conio.h>
int main()
{
int  dlug, liczba, suma=0;
clrscr();
cout << "Wprowadzaj liczby, 0 koniec \n";
dlug=0;
do
 {
 cin >> liczba;
 dlug = ++dlug;
 suma=suma+liczba;
 }
while (liczba!=0);
cout << "Dlug. ciagu wynosi " <<  --dlug << endl; // 0 nie wliczamy – odejmujemy 1
cout << "Suma ciagu wynosi " <<  suma << endl;
getch();
}
```

Instrukcja for

Instrukcja jest wygodna, gdy z góry można określić liczbę powtórzeń.

Postać instrukcji:

Dla zmienna=wart_pocz **do** wart_konc **do** instrukcja;

W zapisie w C najczęściej

for (wart_pocz; zmienna<wart_konc; zmienna++) instrukcja;

Zapis instrukcji w postaci ogólnej w C / C++:

for (wyrażenie1; wyrażenie2; wyrażenie3) { lista instrukcji}

Np. *for (int i=1; i<11; i++) cout << i << endl;*

Zmienna nazywana jest zmienną sterującą instrukcji **for**.

Musi być typu całkowitego, znakowego lub logicznego.

Instrukcja po słowie kluczowym do wykonywana jest tyle razy ile wartości znajduje się w zakresie

wart_pocz .. wart_konc.

Programy z for

```
/* Program for1.cpp Wydruk cyfr od 1 do 5 */
#include <iostream.h>
#include <conio.h>
int main()  {
int i;
for (i=1; i<= 5; i++)  cout << i << endl;
getch();
return 0;
}
 /* Program for2.cpp Wydruk cyfr od 5 do 1 */
#include <iostream.h>
#include <conio.h>
int main() {
int i;
for (i=5; i>= 1; i--)  cout << i << endl;
getch();
return 0; }
```