Dr inż. Robert Wójcik

Zakład Podstaw Informatyki i Teleinformatyki
Politechnika Wrocławska

Wykład 2

2. Podstawowe elementy języka

2.1. Struktura programu

2.2. Zmienne i stałe o typach prostych (niepodzielnych)

2.3. Podstawowe operacje wejścia / wyjścia

Program składa się z:

- dyrektyw preprocesora,
- komentarzy,
- deklaracji,
- instrukcji.

2.1. Struktura programu

```
// To jest komentarz rozciągający się do końca linii

/* To jest komentarz,
   który może zajmować wiele linii
*/
deklaracje globalne: typów, stałych i zmiennych
deklaracje (prototypy) i/lub  definicje funkcji

main()
{
      deklaracje lokalne: typów, stałych i zmiennych
      wywołania funkcji
      return 0;
}

definicje innych funkcji
```

Przykład. 2.1.

```
// prosty program wyprowadzający komunikaty na ekran

#include <stdio.h>
#include <iostream.h>

main(void)
{
 printf("Witaj studencie \n");                    //  '\n' - znak przejścia do nowej linii
 cout << "Witaj programisto" << '\n';  // strumień wyjściowy
 cout << "Koniec \n";
 return 0;
}
```

Przedstawionym kod źródłowy programu zawiera następujące charakterystyczne elementy programu C++:

- Komentarz zaczynający się od //.
- Dyrektywy preprocesora postaci: #include<*Nazwa*> , gdzie *Nazwa* jest nazwą pliku; dyrektywa informuje kompilator o konieczności włączenia pliku nagłówkowego *Nazwa*; plik nagłówkowy zawiera deklaracje stałych, typów danych, zmiennych i zapowiadających deklaracji funkcji.
- #include<stdio.h> - dołącza plik nagłówkowy udostępniający operacje we/wy w stylu C (dekl. funkcji stand. *printf* – wyprowadzającej dane na ekran).
  #include<iostream.h> - dołącza plik nagłówkowy udostępniający operacje we/wy w stylu C++ (dekl. stand. strumienia wyjściowego *cout* i operatora pobierającego dane z programu <<). Języki C i C++ nie zawierają wbudowanej obsługi operacji we/wy będącej częścią rdzenia języka (odmiennie niż Pascal). Do obsługi operacji we/wy wymagane są specjalizowane biblioteki.
- Program C++ składa się z deklaracji globalnych i funkcji. Program musi zawierać funkcję o nazwie *main*, od której rozpoczyna się wykonywanie programu. Funkcja *main* jest najczęściej bezparametrowa co można wyrazić za pomocą słowa kluczowego *void*. Można ją umieścić w dowolnym miejscu kodu. Wszystkie funkcje mogą zawierać lokalne typy danych, stałe i zmienne. Funkcje mają dostęp do stałych, typów danych i zmiennych o zasięgu globalnym.

- C++ definiuje bloki instrukcji używając { i }.
- Każda instrukcja w programie C++ musi kończyć się średnikiem.
- Znaki zamykane są w pojedynczym cudzysłowie, np. 'A'. Łańcuchy zawarte są w podwójnym cudzysłowie, np. "A" jest łańcuchem jednoznakowym (zajmuje dwa bajty, gdyż na końcu dodawany jest znak końca \0).
- Program wyprowadza na ekran trzy łańcuchy. Po każdy wyprowadzeniu następuje przejście do początku nowego wiersza.
- Funkcja *main* musi zwracać wartość odzwierciedlającą stan błędu programu. Wartość 0 oznacza brak błędu. Jeżeli napiszemy *void main()*, to main jest funkcją bezparametrową, która nie zwraca wartości.

Inne elementy programu

- *Słowa kluczowe* – słowa charakterystyczne, pełniące w języku specjalne funkcje, np. void, asm, continue, float, for, inline, do, while, return, i inne. Są słowami zastrzeżonymi i nie mogą być wykorzystywane do oznaczania takich elementów programu jak zmienne, funkcje i typy.
- *Identyfikatory* – słowa, które mogą być użyte do oznaczania obiektów programu źródłowego (nazwy stałych, zmiennych, funkcji); ciągi liter i cyfr rozpoczynające się od litery lub znaku _ (podkreślenie), liczące nie więcej niż 31 znaków; litery małe i duże uważa się za różne (np. ALA i Ala są różnymi identyfikatorami); np. napisy main, printf, cout są identyfikatorami.
- *Literały* – napisy reprezentujące dane. Zapis literału pozwala określić wszystkie atrybuty danej, w tym jej wartość i typ. Literałami są liczby (np. 7, 14.5), znaki (np. 'A') i łańcuchy (np. "Witam").
- *Literały całkowite* – liczby całkowite; mogą być dziesiętne, ósemkowe (jeśli liczba zaczyna się od cyfry 0 i zawiera jedynie cyfry mniejsze niż 8, np. 012 równa się 10), szesnastkowe (jeśli liczba jest poprzedzona napisem 0x lub 0X i zapisywana za pomocą znaków ze zbioru {0,1...9,a,b,c,d,e,f},
  np. 0xA równa się 10).
- *Literały rzeczywiste* – reprezentują liczby rzeczywiste, np. 12e-2, 1.5.
- *Wyrażenia liczbowe* – liczby całkowite i rzeczywiste mogą być tylko nieujemne. Jeśli liczba jest poprzedzona znakiem plus lub minus, to taki napis jest wyrażeniem, np. –2500 wyrażenie typu całkowitego.
- *Literały znakowe* – reprezentują pojedyncze znaki. Literał ma postać znaku widocznego lub opisu znaku, który zawarty jest w pojedynczym cudzysłowie, np. 'A'. Opisem znaku jest symboliczny kod znaku (np. '\101' lub '\x41' – z zakresu ASCII) albo symboliczny opis znaku (np. '\n').
- *Literały łańcuchowe* – ciągi znaków zawarte w podwójnym apostrofie, np. "AAAAAAAAA". Każdy ciąg jest tablicą znaków zawierającą wszystkie znaki łańcucha oraz znak specjalny o kodzie 0 ('\0' null).
- *Jednostki leksykalne* – słowa kluczowe, identyfikatory, literały i ograniczniki (operatory i znaki interpunkcyjne). Podział programu odbywa się od lewej do prawej i od góry do dołu. Jednostka leksykalna, która nie mieści się w wierszu programu może być przeniesiona do następnego wiersza za pomocą znaku \, np. prin\ tf.

2.2. Zmienne i stałe o typach prostych

Wykonując określone zadania komputer przetwarza pewne dane wejściowe, reprezentowane przez liczby i znaki, na odpowiednie dane wyjściowe – wyniki.

W programach posługujemy się zmiennymi i stałymi, w których przechowujemy dane określonego typu. Każda zmienna i stała posiada unikalny identyfikator (unikalną nazwę) oraz typ (np. zmienna całkowita - int, rzeczywista - double).

Stałe – reprezentują dane, które nie ulegają zmianie podczas działania programu (są to liczby, znaki lub teksty).

Rodzaj stałej jest rozpoznawany przez kompilator języka po jej zapisie.
Np.
12 – stała typu całkowitego (typu int),
123.5 – stała typu rzeczywistego (typu double),
'A' – stała znakowa,
"To jest stała tekstowa" – stała łańcuchowa.

Zmienne – reprezentują dane, które mogą się zmieniać podczas działania programu (są dostępne w programie poprzez identyfikatory zmiennych).

Zmienne i stałe dzielimy na:

- *proste* (skalary) - można im przypisywać jedynie dane niepodzielne: całkowite, rzeczywiste, wskazujące, wyliczeniowe;

- *złożone* (agregaty) – można im przypisywać dane złożone: tablicowe, strukturalne i obiektowe.

Przed użyciem zmiennej należy ją **zdefiniować**, tzn. podać jej:
- nazwę,
- typ,
- wartość początkową (opcjonalnie dla zmiennych, obowiązkowo dla stałych).

```
Np.  const int a = 5;       // stała całkowita a typu int o wartości 5;

     double x = 2.3;        // zmienna rzeczywista x typu double o wartości 2.3;

     char z;                // zmienna znakowa z typu char
```

Jeśli nie podamy wartości początkowej zmiennej, to kompilator przyjmie wartość domyślną. Zmienne zdefiniowane poza wszystkimi funkcjami (*zmienne globalne*), są domyślnie inicjowane wartością 0. Zmienne zdefiniowane wewnątrz funkcji (*zmienne automatyczne* – lokalne) mają wartość nieokreśloną.

Np.

```
    const int a = 5;      // stała

     double x;            // zmienna globalna o wartości 0.0
     int c;               // zmienna globalna o wartości 0
     int d = 3;           // zmienna globalna o wartości 3

void main()
{
int w, y;                 // zmienne automatyczne o wartościach,
                          // które nie są określone (przypadkowych)
....
   w = 2;           // inicjacja zmiennej w programie
}
```

Deklaracje i definicje zmiennych

Istnieje możliwość określenia typu zmiennej poprzez deklarację identyfikatora typu zmiennej. Do deklarowania typów zmiennych wykorzystywany jest specyfikator typu **typedef**.

```
Np.  typedef unsigned char byte;     // definicja typu o nazwie byte
                                     // reprezentującego dane typu
                          // unsigned char – liczby 1 bajtowe bez znaku
```

Deklaracja nie tworzy zmiennej – nie rezerwuje dla niej pamięci, a jedynie określa własności (typ, struktura) deklarowanego identyfikatora.

```
Np.  typedef  int calkowite;   // zdefiniowanie identyfikatora typu calkowite
                               // jako int
```

Definicja zmiennej (w odróżnieniu od deklaracji) rezerwuje dla zmiennej pamięć i może być połączona z nadaniem jej wartości początkowej.

```
Np.  calkowite k = 4;   // zmienna całkowita k typu całkowite
```

Definicje i deklaracje służą do interpretacji identyfikatorów.

Ogólna postać definicji lub deklaracji:

<**specyfikator klasy zmiennej**> <**typ zmiennej**> <*lista identyfikatorów*>;

```
Np. int a, b, c;  // definicje zmiennych prostych a, b, c typu int;
      static  int d;        // definicja zmiennej statycznej d
      extern double w;      // deklaracja zmiennej zewnętrznej w
                            // (definicja zmiennej może być w innym pliku)
```

Specyfikator klasy określa sposób przechowywania zmiennej w pamięci lub definiuje nowy typ danej. Może on należeć do jednej z czterech klas pamięci: **auto**, **static**, **extern**, **register** lub być specyfikatorem **typedef**, który umożliwia definiowanie typów danych.

Jeśli specyfikator klasy nie występuje to domyślnie dla definicji zmiennych globalnych (zmienne znajdujące się poza funkcjami), przyjmowana jest klasa extern, natomiast dla definicji lokalnych (zmienne znajdujące się wewnątrz funkcji) przyjmowana jest klasa auto.

Zmienne klasy static mają zasięg blokowy (od { do } tak jak zmienne automatyczne), ale w odróżnieniu od zmiennych automatycznych istnieją i zachowują swoje wartości przez cały czas wykonywania programu (statyczny czas trwania).

Zmienne automatyczne mogą być umieszczane w szybkich rejestrach procesora. Powstaje w ten sposób klasa zmiennych rejestrowych (register), których wykorzystanie przyspiesza obliczenia (możliwość taka dotyczy zazwyczaj tylko niektórych typów danych, np. int).

Definicja typu wskaźnikowego

Jeśli w liście identyfikatorów nazwa jest poprzedzona operatorem wyłuskania * (gwiazdka), to deklarowana zmienna jest typu wskaźnikowego. Zmienne tego typu służą do przechowywania adresów innych zmiennych.

```
Np.
 double * wd;          // wd wskaźnik na zmienną typu double
 int      * wi;        // wi wskaźnik na zmienną typu int

typedef  double * twd;  // definicja typu twd – wskaźnika na obiekty
                        // typu double
twd     wsk;            // zmienna - wskaźnik wsk na zmienne typu twd

twd p1, p2;   // dwie zmienne (wskaźniki) typu twd
```

ostatnia definicja interpretowana jako

```
double *p1, *p2;
```

Zdefiniowana zmienna wd może być wykorzystana do przechowywania programowego adresu zmiennej typu double. Zmienna wd jest wskaźnikiem (adresem) typu double*. Podobnie wi jest wskaźnikiem na zmienne typu int. Zmienna wi jest typu int*.

Typ void

Typ void umożliwia definiowanie zmiennych o nie określonym typie lub informowanie o tym, że parametry nie występują.

```
Np.  void *pv;   // pv – wskaźnik na zmienną dowolnego typu
                 // może zawierać adres zmiennej dowolnego typu
```

Zasięg deklaracji i definicji, widoczność zmiennych

Deklaracje i definicje zmiennych znajdujące się poza wszystkimi blokami { ... } mają charakter globalny, tzn. są widoczne w całym pliku począwszy od miejsca wystąpienia.

Deklaracje i definicje zmiennych znajdujące się wewnątrz bloku { ... } mają charakter lokalny – są widoczne w obrębie bloku, w którym wystąpiły.

Identyfikator zmiennej lokalnej przesłania w danym bloku definicję lub deklarację identyfikatora zmiennej zewnętrznej o tej samej nazwie.

Nazwy zmiennych w tym samym bloku nie mogą się powtarzać.

```
int  i=10;                  // zmienna globalna i = 10;

void main()
{
cout << i << endl;   // i = 10;
  {
      int i=2;              // zmienna lokalna i = 2
       {
           int  i=7;        // zmienna lokalna i = 7;
                i++;
           cout << i << endl;          // i = 8
       }
    cout << i << endl;                 // i =2
    }
     cout << i << endl;                // i =10
}
```

Proste typy danych występujące w C++:

- int - liczby całkowite (typ domyślny),
- char - znaki, liczby całkowite o małych wartościach
- float - liczby rzeczywiste (zmiennoprzecinkowe) pojedynczej precyzji,
- double - liczby rzeczywiste podwójnej precyzji (typ domyślny).

Modyfikatory typów:

- signed – zmienna całkowita ze znakiem (domyślnie signed int; znak liczby jest określony przez jej najstarszy bit),
- unsigned – zmienna całkowita bez znaku,
- short – zmienna całkowita o zmniejszonej precyzji (int),
- long – zmienna całkowita o rozszerzonej precyzji.

Przykładowe definicje zmiennych:

```
char c;          // zmienna znakowa
int X;           // zmienna typu całkowitego
long a, b;       // dwie zmienne typu long int
float z;         // zmienna typu rzeczywistego
double w;        // zmienna podwójnej precyzji
```

Przykładowe definicje połączone z inicjacją wartością:
```
char c = 'A';
int i = 10;
double z = 12.5;
```

Podczas nadawania wartości można kontrolować typy literałów liczbowych. Służą do tego celu przyrostki:

- U lub u – literał całkowity bez znaku (unsigned),
- L lub l – literał całkowity jest typu long; literał rzeczywisty jest typu long double,
- F lub f – literał rzeczywisty jest typu float (zamiast double).

Jeżeli opuścimy specyfikatory typu, kompilator wykonana niezbędne konwersje typu. Umieszczenie specyfikatora eliminuje konwersje i daje możliwość kontroli interpretacji stałych liczbowych (np. przy mnożeniu liczb typu int wynik może przekroczyć zakres int: int seg; long adr =16L∗seg).

```
unsigned  a = 12500U;
long i = 1000L;
float z = 123.45F;
```

Tabela pokazuje predefiniowane typy danych w C++ (kompilator BC++ 3.1)

Typ danej	Zakres	Rozmiar [B]	Uwagi
char signed char	-128 ... 127	1	małe liczby całkowite, znaki ASCII
unsigned char	0 ... 255		
int signed int short short int signed short int	-32768 ... 32767	2	liczby całkowite ze znakiem
unsigned unsigned int unsigned short int	0 ... 65535	2	liczby całkowite bez znaku
long long int signed long int	-2147483648 ... 2147483647	4	bardzo duże liczby całkowite
unsigned long int unsigned long	0 ... 4294967295	4	
float	-3.4E-38 ... 3.5E38	4	7 cyfr znaczących
double (long float)	-1.7E-308 ... 1.7E308	8	15 cyfr znaczacych
long double	3.4E-4932 ... 1.1E4932	10	19 cyfr znaczących

Zawsze spełnione są następujące warunki:

- dla typów całkowitych

  sizeof(char) <= sizeof(short) <= sizeof(int) <=sizeof(long)
  (modyfikatory signed oraz unsigned zmieniają zakres wartości zmiennej danego typu, ale nie zmieniają wielkości przydzielanego obszaru pamięci);

- dla typów rzeczywistych (zmiennoprzecinkowych)
  sizeof(float) <= sizeof(double) <= sizeof(long double)

Operator
- *sizeof (x)* – podaje wielkość pamięci w bajtach zajmowanej przez obiekt x; wbudowany operator języka C.

Stałe

- oparte na definicji makro *#define*

  ```
  #define nazwa_stałej wartość          // bez średnika

  np.  #define N 10
       #define Kom "To jest komunikat"
  ```

  Dyrektywa kompilatora #define uruchamia preprocesor, który zastępuje stałe odpowiadającymi im wartościami.

- stałe formalne oparte na deklaracji *const*

  ```
  const typ_danej nazwa_stałej = wartość;

  np.  const char litera = 'C';          // średnik
       const double licz = 15.0;
  ```

  Typ danej jest opcjonalny. Domyślnie przyjmowany jest int.
  Stała formalna nie może być modyfikowana w programie.
  Kompilator może dokonać sprawdzenia zgodności typów w wyrażeniach.
  Stałe mogą być lokalne (zadeklarowane wewnątrz funkcji main) lub globalne.

Klasyfikacja stałych

Stałe dzielą się na: znakowe, całkowite, rzeczywiste, łańcuchowe.

Stała znakowa może być przedstawiona za pomocą:
- literału, np. 'A', 'b', '1';
- symbolicznego kodu, np. '\101' - 65, '\x41' - 65, '\142' – 98, '\61' – 49 (x – zapis szesnastkowy, pozostałe przypadki – zapis ósemkowy);
- symbolicznego opisu, np. '\n', '\t', '\\', '\%'.

Literał – pojedynczy znak ujęty w apostrofy. Wartość liczbowa stałej znakowej jest równa kodowi ASCII znaku. W C++ stała ta jest typu char (–128 do 127), np. const char z = 'A';

Symboliczny kod – ujęta w apostrofy i poprzedzona znakiem \ (back slash) liczba ósemkowa (np. '\101'; ogólnie '\0nn', '\nnn') lub szesnastkowa (np. '\x41'; ogólnie '\xhh'), np. const char z ='\101'; const char z = '\20'; const char z = '\x41'; Liczba podaje kod znaku i jest typu unsigned char; ogólna postać może być maksymalnie 3 znakowa ('\nnn' i '\xhh');

Symboliczny opis – definiuje stałą za pomocą literowego oznaczenia znaku, np. \n – przejście do nowej linii (kod 10), \t – tabulacja,
\r - nowy wiersz, \\, \%, \', \" – znaki \, %, ', ", np. const char z = '\%';

Stałe znakowe są liczbami całkowitymi i mogą być wykorzystywane w wyrażeniach (np. int i = 'A' + 1; i=66; i = 'a' – 'A'; i=32).

Stała całkowita może mieć postać:
- dziesiętną, np. 65, 98, 49
- ósemkową, np. 0101, 0142, 061;
- szesnastkową, np. 0x41, 0x62, 0x31.

Stała całkowita może być zakończona małymi lub dużymi literami u, l, ul (U, L, UL). Jest ona wówczas odpowiednio typu unsigned, long, unsigned long. Jeżeli nie podano specyfikatora typu to typ stałej zależy od jej wartości (gdy wartość stałej rośnie staje się ona kolejno typu int, unsigned, long, unsigned long, np. 32000 – stała typu int, 40000 – stała typu unsigned, 70000 – stała typu long).

Stała rzeczywista (zmiennoprzecinkowa) może mieć postać:
- dziesiętną, np. 65.0, 98.0, 49.0, .43,
- wykładniczą, np. 6.5e1, 980E-1, 0.49e+2, .27e-2.

Stała rzeczywista może być zakończona małymi lub dużymi literami f, l (F, L). Jest ona wówczas odpowiednio typu float i long double (np. 1.2F, 123.0L). Stała rzeczywista bez przyrostka jest typu double.

Wszystkie stałe są zawsze *bez znaku* (np. 12, 23.5). Jeżeli zostały poprzedzone znakiem – lub +, to są wyrażeniami złożonymi z operatora (– lub +) oraz liczby dodatniej odpowiedniego typu (12 – typu int; 23.5 – typu double). Np. –12, -23.5, -0.43, -.27, -.34e-2.

Stała łańcuchowa (tekstowa) – ciąg znaków objęty w cudzysłowy.

Długość stałej tekstowej może być praktycznie dowolna. Stała zawsze zawiera na końcu znak '\0', który jest automatycznie dodawany przez kompilator i stanowi ogranicznik tekstu, np. stała "Tekst" ma długość 5, ale zajmuje w pamięci 6 bajtów ("Tekst" + '\0').

Jeżeli definicja stałej nie mieści się w jednym wierszu, to należy wprowadzić znak \ i kontynuować w kolejnej linii programu.

Każda para sąsiadujących stałych łańcuchowych jest traktowana jako jeden łańcuch, np. napisy "Tek""st" oraz "Te""k\x73t" są równoważne.

Jednak napisy 'A' oraz "A" nie są identyczne, gdyż 'A' jest literałem o wartości 65, natomiast "A" jest stałą łańcuchową złożoną z dwóch znaków 'A' i '\0'.

Przykład 2.2. Wyprowadzanie stałych na ekran

```
#include <stdio.h>
#include <conio.h>
#include <iostream.h>

  // Stale globalne

const char znak = 'A';
const int kod = 65;

void main(void)
{
  // Stale lokalne

 const long a = 126l;
 const unsigned long b = 12345lu;
 const float s1 = 1.123456789f;
 const double s2 = 1.123456789;
 const long double s3 = 1.1234567890123456789l;

 clrscr();

 printf("Znak %c - kod znaku %d\n",znak, kod);
 printf("Liczba long = %ld \n", a);
 printf("Liczba unsigned long = %lu \n", b);
 printf("Liczba float = %f \n", s1);                    // zaokragla pozycje 6
 printf("Liczba double = %4.2lf \n", s2);               // format lf = f dla printf
 printf("Liczba long double = %20.19Lf \n", s3);        // pozycja 17

getch();          // czekaj na znak lub enter

}
```

W przypadku funkcji printf pierwszy łańcuch podaje format argumentów.

printf ("format", arg, arg, ..., arg);

Format %4.2lf oznacza, że wyświetlana liczba zajmie 4 kolumny (łącznie z kropką) i będzie miała 2 miejsca dziesiętne.

Wyliczenia

Typ wyliczeniowy:

- wprowadzany za pomocą słowa kluczowego **enum**;
- za jego pomocą można nadawać identyfikatory grupie stałych wartości całkowitych;
- wyliczenie obejmuje ciąg wartości stałych o logicznie powiązanym znaczeniu.

Np.
```
enum { CZERWONY, ZIELONY, NIEBIESKI };
```

Identyfikator CZERWONY jako pierwszy na liście będzie miał przypisaną wartość 0, następne na liście uzyskują wartości sukcesywnie zwiększające się o 1.

Domyślne przypisania możemy zmieniać przy deklaracji przez bezpośrednie przypisanie.

Np.
```
enum { CZERWONY = 4, ZIELONY = 6, NIEBIESKI = 8 };
```

- Dopuszczalne jest, by dwa różne identyfikatory miały taką samą wartość.
- W przypisaniach mogą wystąpić wyrażenia stałe, kompatybilne z typem int.

W programach możemy dokonywać podstawień

```
int x = ZIELONY;   // x = 6;
```

Typ wyliczeniowy może być nazwany, np.

```
enum  Kolory { R, G, B };
```

Wówczas, możemy napisać:
```
Kolory a;   // zmienna typu enum
 a = R;     // a = 0
```
Dzięki zmiennym wyliczeniowym kompilator może poinformować nas o użyciu wartości nie należącej do danego typu wyliczeniowego w miejscu, gdzie oczekiwana jest zmienna tego typu.

2.3. Podstawowe operacje wejścia / wyjścia

Funkcja printf

Do wyprowadzania danych na standardowe wyjście służy funkcja printf. Posiada ona bardzo duże możliwości formatowania danych.

printf ("format", arg, arg, ..., arg);

Instrukcja formatowania dla jednego argumentu ma w ogólnym przypadku postać:

%[znaczniki] [szerokość][.dokładność] [F | N | h | l | L] <znak typu>

- Znaczniki:
  0 – dodanie zer wypełniających minimalny rozmiar pola określony przez szerokość,
  - - wyrównanie do lewego brzegu w obrębie pola,
  + - wyświetlenie znaku plus lub minus liczby,
  spacja – wartość ujemna wyświetlany minus, dodatnia – spacja,
  # - liczby w układzie szesnastkowym wyświetlane z 0X lub 0x, liczby ósemkowe poprzedzone zerem, liczby rzeczywiste wyświetlane z kropką dziesiętną.

- Szerokość:
  określa minimalną liczbę wyświetlanych znaków; nadmiarowe pola uzupełniane są zerami (jeśli szerokość poprzedzona zerem) lub spacjami; jeśli liczba zajmuje więcej pozycji niż podana szerokość, to szerokość jest ignorowana, a liczba jest wyprowadzana w całości; jeśli zamiast wartości podamy *, to funkcja printf uzyska wartość szerokości pola z listy argumentów funkcji.

- Dokładność:
  określa liczbę cyfr wyświetlanych po przecinku; ostatnia cyfra jest zaokrąglana; jeśli nie jest podana, to wyprowadzanych jest 6 cyfr; podanie * oznacza, że dokładność pobrana zostanie z listy argumentów.

- Modyfikatory F, N, h, l, L:
  określają rozmiar wyświetlanych wartości i służą do zmiany domyślnego rozmiaru argumentu; F i N używane są w połączeniu ze wskaźnikami far i near; opcje h, l wskazują odpowiednio typ short int lub long; za pomocą opcji l można wyprowadzić liczbę long float = double; za pomocą L można wyprowadzić liczbę typu long double.

- Znaki typu:
  c – pojedyńczy znak,
  d lub i – liczba całkowita ze znakiem, dziesiętnie,
  x lub X – liczba całkowita ze znakiem, szesnastkowo,
  o - liczba całkowita ze znakiem, ósemkowo,
  p – wyświetlanie adresu wskaźnika (seg:ofs),
  f – liczba rzeczywista ze znakiem (float, double); format [-]ddd.ddd,
  e lub E – liczba rzeczywista ze znakiem; format wykładniczy
  [-]d.dddE[+|-]ddd,
  g lub G – wyświetlanie liczby rzeczywistej ze znakiem w formacie
  f lub e w zależności od wartości oraz podanej dokładności;
  s – wyświetlanie znaków aż do napotkania znaku zera kończącego
  łańcuch.

W szczególności:

printf("%<szerokość>d", zmienna_int);

printf("%<szerokość>ld", zmienna_long);

printf("%<szerokość>s", zmienna_string);

printf("%c%d %f", c, a, z);

W łańcuchu formatującym mogą wystąpić sekwencje znaków sterujących, np. '\n' znak przejścia do nowej linii. Do najważniejszych należą:

\a – alarm (sygnał dźwiękowy); kod 0x07;
\b – cofnięcie o jedno miejsce; kod 0x08 (bs);
\f – wysuw strony; kod 0x0c (ff);
\n – nowa linia; kod 0x0a (nl);
\r – powrót karetki; kod 0x0d (cr);
\t – tabulacja pozioma; kod 0x09 (ht);
\v – tabulacja pionowa; kod 0x0b (vt);
**** – znak łamania (\); kod 0x5c;
\' – znak apostrofu('); kod 0x2c;
\" – znak cudzysłowu (") ; kod 0x22;
\? – znak zapytania (?); kod 0x3f.

Funkcja scanf

Umożliwia wprowadzanie danych z konsoli.

scanf ("format", &arg, &arg, ..., &arg);

Format dla jednego argumentu ma następującą postać:

% [*] [szerokość][F | N | h | l | L]<znak typu>

Łańcuch formatujący informuje o ilości wprowadzanych danych, ich typie i formacie. Funkcja scanf korzysta z adresów zmiennych pod, które wprowadzane są dane. W przypadku, gdy wprowadzanych jest kilka danych istotna jest liczba i rodzaj znaków pomiędzy elementami łańcucha formatującego. Należy wpisać taką samą liczbę i te same znaki podczas wprowadzania danych. Symbol * oznacza pominięcie danej odpowiadającej polu wejściowemu w strumieniu (wskaźnik strumienia zostanie przesunięty, ale dana nie zostanie wczytana pod zmienną). Dla scanf: f – float, natomiast lf – double.

Np. scanf("%c :%c", &x, &y); znak char, spacja, dwukropek, znak char;
scanf("%d%*d%d", &i, &j, &k); dla 2 3 4 jest i=2, j=4, k=?
scanf("%5d%3d", &i, &j); dla 123456 jest i=12345, j=6.

Przykład 2.3. Operacje z wykorzystaniem printf i scanf.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

void main (void)
{
  char c = 'A';
  unsigned char u = 140;
  int xint = 40000;
  unsigned xword = 40000U;
  long xlong = -500000;
  unsigned long ulong = 2000000;
  float xfloat = 4.123456789;
  double xdouble = 4.123456789;
  long double xldouble = 4.123456789;
  clrscr();
  printf("Wartosci zmiennych\n");
  printf("Znak char  %%c %c\n", c);
  printf("Znak unsigned char %%c %c\n", u);
  printf("Liczba int 40 000 %%d %d\n", xint);
  printf("Liczba unsigned %%u %u\n", xword);
  printf("Liczba long %%ld %ld\n", xlong);
  printf("Liczba unsigned long %%lu %lu\n", ulong);
  printf("Liczba float %%f %f\n", xfloat);
  printf("Liczba double %%lf %lf\n", xdouble);
  printf("Liczba  long double %%Lf %Lf\n", xldouble);
// możliwości formatu zostaną wykorzystane jeżeli podamy
// liczbę miejsc po przecinku

  printf("Podaj nowe wartosci zmiennych\n");

  printf("Wpisz znak char %%c\n");   scanf("%c",&c);
  printf("Wprowadzono znak %c\n", c);
  fflush(stdin);  //czysc bufor

  printf("Wpisz znak char %%c\n");   scanf("%c",&u);
  printf("Wprowadzono znak %c\n", u);

  printf("Wpisz liczbe int %%d \n");   scanf("%d",&xint);
  printf("Wprowadzono liczbe int %%d %d\n", xint);

  printf("Wpisz liczbe unsigned %%u \n");   scanf("%u",&xword);
  printf("Wprowadzono liczbe unsigned %%u %u\n", xword);

  printf("Wpisz liczbe long %%ld \n");
  scanf("%ld",&xlong);
  printf("Wprowadzono liczbe long %%ld %ld\n", xlong);

  printf("Wpisz liczbe float %%f\n");
  scanf("%f",&xfloat);                                  // 1.23
  printf("Wprowadzono liczbe float %%f %f\n", xfloat);    // 1.230000
  printf("Wprowadzono liczbe float %%E %E\n", xfloat); // 1.230000E+00
  printf("Wprowadzono liczbe float %%g %g\n", xfloat);

  xdouble = (double) xfloat * xfloat + 1;

  printf("Liczba double %%lf %lf\n", xdouble);
  printf("Liczba double %%lE %lE\n", xdouble);
  printf("Liczba double %%lg %lg\n", xdouble);

  getch();
  }
```

Strumienie wejścia - wyjścia

W C++ można wykonywać operacje we/wy z wykorzystaniem strumieni zrealizowanych w postaci proceduralnej (język C) lub obiektowej (język C/C++). Do strumieni standardowych zaliczamy:

- **stdout** (wskaźnik do struktury), **cout** (obiekt) - strumień wyjściowy (zwykle ekran; istnieje możliwość przedefiniowania na plik),
- **stdin** (wskaźnik do struktury), **cin** (obiekt) - strumień wejściowy (zwykle klawiatura; istnieje możliwość przedefiniowania na plik),
- **stderr** (wskaźnik do struktury), **cerr** (obiekt) - standardowy strumień błędów (zwykle ekran; istnieje możliwość przedefiniowania na plik).

Funkcje standardowe scanf i printf używają domyślnie strumieni w wersji proceduralnej (stdin i stdout). Strumienie we/wy w wersji obiektowej (cin/cout) umożliwiają wykonywanie operacji na danych za pośrednictwem wewnętrznych funkcji interfejsu oraz odpowiednio zdefiniowanych operatorów.

Operator **<<**
- wstawiania danych do strumienia cout.

Operator **>>**
- pobierania danych ze strumienia wejściowego cin.

Np.

```
cout << 3 << '  '<< 2.3 << '  '<< 'X' << "Ala \n";

cin >> pierwsza >> druga >> trzecia;
```

Za pomocą odpowiednich funkcji strumieni można sterować sposobem wykonywania operacji we/wy. Pobieranie danych ze strumienia cin wymaga potwierdzenia za pomocą Enter.

Przykładowe metody:

cout.width(arg) lub setw(arg) – liczba pozycji zajmowanych przez wartość,

cout.fill(arg) – znak wprowadzany na niewykorzystane pozycje,

cout.precision(arg) – liczba miejsc po przecinku.

cout.put(arg) – wyprowadzanie jednego znaku,

cin.get(arg) - pobieranie jednego znaku z klawiatury,

cin.getline(adres, ile) – pobieranie łańcucha znaków,

Np.

```
cout << "Moja liczba to" << setw(4) << 1000 << endl;

cout << "Moja liczba to" << setw(5) << 1000 << endl;
```

Przykład 2.4. Wykorzystanie strumieni we/wy.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

void main (void)
{
  char c = 'A';
  int x = 10;
  float z1 = 1.1257;

  clrscr();

  cout.fill('.');  cout.width(10);

  // dana wyswietlana na 10 pozycjach;
  // wolne pozycje wypelniane kropkami

  cout << c << endl;

  cout.fill('*');  cout.width(10);
  cout << x << endl;

  cout.precision(4);          // cztery miejsca po przecinku
  cout << z1 << endl;

  cout.precision(2);          // dwa miejsca po przecinku
  cout << z1 << endl;

  getch();
  }
```

Funkcje obsługujące znakowe operacje we/wy

- getche() - funkcja zwraca znak wprowadzony z konsoli i wyświetla go na ekranie; nie czeka na enter;
- getchar() – funkcja zwraca znak wprowadzony z konsoli i wyświetla go na ekranie; wymaga wprowadzenia enter; działa nie tylko z konsolą, ale również z ogólnymi urządzeniami we/wy;
- getch() – funkcja zwraca znak wprowadzony z konsoli, ale nie wyświetla go na ekranie;
- putch() – funkcja wysyła pojedyńczy znak na ekran;
- putchar() – funkcja wyprowadza znak na ekran; działa nie tylko z konsolą, ale również z ogólnymi urządzeniami we/wy.

Przykład 2.5. Wykorzystanie znakowych funkcji we/wy

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

void main (void)
{
  char c1, c2;

  clrscr();
  printf("\nWpisz pierwszy znak: ");
  c1 = getche();
  printf("\nWpisz drugi znak: ");
  c2 = getchar();
  printf("\n\nWprowadziles znaki: ");
  putch(c1);
  putchar(c2);

  getch();
  }
```

Wyświetlanie danych w trybie tekstowym

Dostępne funkcje obsługi ekranu umożliwiają:

- zmianę trybu tekstowego ekranu,
- sterowanie atrybutami znaków wysyłanych na ekran,
- realizację operacji wejścia-wyjścia (klawiatura-ekran),
- obsługę pamięci ekranu (operacje na blokach pamięci),
- sterowanie położeniem i kształtem kursora.

Ekran w trybie tekstowym jest podzielony na jednakowe komórki, w każdej może znaleźć się jeden znak, posiadający pewien atrybut (kolor i tło). Liczba komórek zależy od aktualnego trybu tekstowego.

Na ekranie tekstowym zawsze jest określone pewne „okno" tekstowe, do którego wysyłane są informacje podczas wykonywania operacji wyjścia na ekran. Standardowo oknem jest cały obszar ekranu, ale można to zmienić.

Dostępne funkcje

Organizacja ekranu:

- window – określanie obszaru ekranu zajmowanego przez okno tekstowe;
- textmode – przełączanie między różnymi trybami pracy ekranu;
- gettextinfo – informacja o stanie ekranu w trybie tekstowym.

Sterowanie atrybutami znaków wysyłanych na ekran:

- textattr – ustawianie atrybutu tekstu;
- textbackground – ustawianie koloru tła;
- textcolor – ustawianie koloru znaków.

Sterowanie pozycją i wyglądem kursora:

- gotoxy – ustawianie kursora na zadanej pozycji;
- wherex, wherey – pobieranie współrzędnych kursora;
- setcursortype – zmiana wyglądu kursora lub jego brak.

Czyszczenie ekranu:

- clreol – usuwa znaki od pozycji kursora do końca linii;
- clrscr – kasowanie ekranu;
- delline – usuwa wiersz na pozycji kursora i przesuwa pozostałe;
- insline – wstawia wiersz na pozycji kursora i przesuwa pozostałe.

Operacje we/wy (konsola-ekran):

- kbhit – funkcja sprawdzająca czy naciśnięcie klawisza jest dostępne;
- ungetch – wprowadzanie znaku na początek bufora klawiatury, powodując, że będzie on następnym odczytanym znakiem;
- cprintf – działa jak printf, ale znaki wysyłane są do aktualnego okna tekstowego i z odpowiednim atrybutem.

Operacje na blokach pamięci obrazu

- gettext – zapamiętanie prostokątnego fragmentu obrazu w buforze pamięci;
- movetext – kopiowanie prostokątnego fragmentu obrazu na zadaną pozycję;
- puttext – kopiuje zawartość bufora pamięci do ustalonego prostokątnego fragmentu obrazu.

Przykład 2.6. Wyprowadzanie tekstu na ekran.

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

void main (void)
{
  clrscr();    gotoxy(2,2);    textcolor(1);
  cprintf("Napis w kolorze niebieskim\n\r");
  gotoxy(4,4);   textcolor(2);
  cprintf("Napis w kolorze zielonym\n\r");
  gotoxy(6,6);   textcolor(3);
  cprintf("Napis w kolorze jasno niebieskim\n\r");
  gotoxy(8,8);   textcolor(4);
  cprintf("Napis w kolorze czerwonym\n\r");
  getch();
  }
```