

# **Podstawy informatyki**

Elektrotechnika I rok

## **Język C++** **Typy danych, zmienne, funkcje** Instrukcja do ćwiczenia



## Tematyka i cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z prostymi i złożonymi typami danych oraz własnościami zmiennych statycznych, automatycznych i rejestrowych. Omówione są także podstawowe własności funkcji oraz możliwości strukturalnego tworzenia programów z wykorzystaniem funkcji w języku C++.

## Wprowadzenie

### Typy danych

W języku C++ typy danych możemy podzielić na proste i złożone.

### Typy proste

W języku C++ są dostępne następujące proste typy danych:

#### Typ pusty

Typ **void** jest najczęściej używany do deklaracji funkcji nie zwracającej żadnej wartości, do deklaracji pustej listy argumentów funkcji i do deklaracji wskaźników beztypowych.

#### Typy całkowite

Typ **bool** jest typem całkowitym przeznaczonym do przechowywania wartości boolowskich: 0 lub 1 (albo false lub true). Najczęściej zmienne typu bool mają rozmiar 1 bajtu.

Typ **char** jest typem znakowym służącym do przechowywania dowolnego znaku ze zbioru znaków dostępnego w danym systemie komputerowym. Każdy znak jest przechowywany jako liczba całkowita (przeważnie bez znaku – zależy to od implementacji). W większości implementacji języka C/C++ zmienne typu `char` mają rozmiar 1 bajtu, co powoduje, że mogą przechowywać liczby całkowite z zakresu  $0 \div 255$  lub  $-128 \div +127$ , jeśli są interpretowane ze znakiem. Na obiektach typu `char` można wykonywać operacje arytmetyczne. Najczęściej stosuje się kodowanie znaków w standardzie ASCII.

Typ **int** jest typem całkowitym pozwalającym przechowywać liczby całkowite ze znakiem. Rozmiar obiektu typu `int` odpowiada najczęściej długości słowa danego komputera (lecz nie mniej niż 16 bitów): w systemach 16-bitowych obiekty typu `int` mają rozmiar 2 bajtów, w systemach 32-bitowych – 4 bajtów.

Przy deklaracji obiektów typu `char` lub `int` można stosować następujące modyfikatory typu:

- **unsigned** w celu jawnego wyspecyfikowania, że wartości mają być traktowane bez znaku (nieujemne).
- **signed** w celu jawnego wyspecyfikowania, że wartości mają być traktowane ze znakiem.

Przy deklaracji obiektów typu `int` można stosować następujące modyfikatory (oprócz wymienionych powyżej):

- **short** – implementacje języka C/C++ gwarantują, że rozmiar obiektów typu `short int` jest nie większy niż obiektów typu `int` (mogą być równe).
- **long** – implementacje języka C/C++ gwarantują, że rozmiar obiektów typu `long int` jest nie mniejszy niż obiektów typu `int` (mogą być równe) oraz nie mniej niż 32 bity.
- **long long** – implementacje języka C/C++ gwarantują, że rozmiar obiektów typu `long long int` wynosi 64 bity. W systemach 16-bitowych typy `long long int` i `unsigned long long int` nie są implementowane

W tabeli 1 zestawiono rozmiary obiektów typu `int` i pochodnych, otrzymanych przez dodanie modyfikatorów typu oraz zakresy liczb, które te obiekty mogą przechowywać.

# Język C++. Typy danych, zmienne, funkcje

Tabela 1

Typ	Systemy 16-bitowe		Systemy 32-bitowe	
	Rozmiar (bajty)	Zakres	Rozmiar (bajty)	Zakres
short int	2	$-2^{15} \div (2^{15}-1)$	2	$-2^{15} \div (2^{15}-1)$
unsigned short int	2	$0 \div (2^{16}-1)$	2	$0 \div (2^{16}-1)$
int	2	$-2^{15} \div (2^{15}-1)$	4	$-2^{31} \div (2^{31}-1)$
unsigned int	2	$0 \div (2^{16}-1)$	4	$0 \div (2^{32}-1)$
long int	4	$-2^{31} \div (2^{31}-1)$	4	$-2^{31} \div (2^{31}-1)$
unsigned long int	4	$0 \div (2^{32}-1)$	4	$0 \div (2^{32}-1)$
long long int	---	----	8	$-2^{63} \div (2^{63}-1)$
unsigned long long int	---	----	8	$0 \div (2^{64}-1)$

Przykłady deklaracji zmiennych całkowitych:

```
char c;  
signed char sc, scl;  
int x1, x2;  
long int l1;  
unsigned int uu, ww;  
unsigned long int ul1, ul1;  
long long int j23;  
unsigned long long int fl6;
```

## Typy zmiennoprzecinkowe (zmiennopozycyjne).

Obiekty zmiennoprzecinkowe służą do przechowywania i wykonywania obliczeń na liczbach rzeczywistych. W implementacjach języka C/C++ można wyróżnić następujące typy zmiennoprzecinkowe:

- **float** – typ pojedynczej precyzji,
- **double** – typ podwójnej precyzji,
- **long double** – typ rozszerzonej precyzji (w wielu implementacjach utożsamiany z typem double).

W tabeli 2 zestawiono przybliżoną dokładność obliczeń, rozdzielczość i dopuszczalny zakres wartości obiektów typu float i double.

Tabela 2

Typ	Dokładność (cyfr dziesiętnych)	Rozdzielczość	Najmniejsza liczba (moduł)	Największa liczba (moduł)
float	ok. 6	$10^{-5}$	ok. $10^{-38}$	ok. $10^{37}$
double	ok. 15	$2 \cdot 10^{-16}$	ok. $2 \cdot 10^{-308}$	ok. $2 \cdot 10^{308}$

Przykłady deklaracji zmiennych zmiennopozycyjnych:

```
float fl1, fl2;  
double db, zm;
```

## Typy złożone

Język C++ pozwala na budowanie złożonych typów danych na bazie typów prostych lub wcześniej zdefiniowanych typów złożonych. W języku C podstawowymi typami złożonymi są: **tablice**, **struktury**, **unie** i **klasy**. Poniżej zostaną pokrótce omówione 3 pierwsze typy złożone. Klasy będą przedmiotem innych ćwiczeń.

### Tablice

**Tablice** są obiektami zawierającymi określoną liczbę obiektów składowych tego samego typu (prostego lub złożonego). Dostęp do danego elementu tablicy odbywa się przez indeks będący numerem porządkowym danego elementu. Indeks danego elementu musi być wyrażeniem całkowitym podanym w nawiasach kwadratowych [ ] będących operatorem indeksowania tablicy. W języku C++

# Język C++. Typy danych, zmienne, funkcje

pierwszy element tablicy ma indeks 0. Tablice mogą być jedno- lub wielowymiarowe, które są traktowane jako tablice tablic. Przy deklaracji tablic dwuwymiarowych jako pierwsze podaje się liczbę wierszy, a następnie liczbę kolumn. Przy odwoływaniu się do elementu tablicy dwuwymiarowej jako pierwszy podaje się numer wiersza, a następnie kolumny.

Przykłady deklaracji tablic:

```
int x[6];
long int k[100][10];          /* 100 wierszy, 10 kolumn */
double d[10];
char cc[15];
```

Przykłady odwołań do elementów tablic zadeklarowanych powyżej:

```
x[0] = x[2];
k[1][6] = 23;
cc[2] = 'k';
```

## Struktury

**Struktury** są obiektami zawierającymi elementy składowe różnego typu (prostego lub złożonego). Każdy element składowy struktury ma swoją nazwę, według której jest identyfikowany. Dostęp do elementu struktury umożliwiają operatory dostępu: „.” (kropka) oraz „->” (minus i znak większości).

Przykłady deklaracji struktur:

```
struct Alfa {
    int x;
    double dd1, dd2;
    int tab[10];
} s1;

struct Beta {
    char c1, c2[5];
    int j;
} s2;
```

Przykłady odwołań do elementów struktur zadeklarowanych powyżej:

```
s1.x = 45;
s1.tab[5] = s2.j
s2.c1 = 0x30;
s1.dd1 = 3.141592;
```

## Unie

**Unie** są obiektami zawierającymi elementy składowe różnego typu (prostego lub złożonego) przy czym w danej chwili użyty może być tylko jeden z elementów składowych. Unię można sobie wyobrazić jako strukturę, w której wszystkie elementy są umieszczone na tym samym adresie. Każdy element składowy unii ma swoją nazwę, według której jest identyfikowany. Dostęp do elementu unii umożliwiają operatory dostępu: „.” (kropka) oraz „->” (minus i znak większości).

Przykłady deklaracji unii:

```
union Gamma {
    double dd1, dd2;
    int x;
    int tab[10];
} U1;

union Delta {
    char c1, c2[5];
    float j;
    struct {
        long int kk1, kk2;
    };
};
```

# Język C++. Typy danych, zmienne, funkcje

```
    } st;  
} U2;
```

Przykłady dostępu do elementów unii zadeklarowanych powyżej:

```
U1.dd2 = 1.34e-4;  
U2.st.kk2 = U2.st.kk1;  
U2.c2[4] = 'Q';
```

## Zmienne

W języku C++ zmienne dowolnego typu można podzielić na:

- globalne,
- lokalne automatyczne lub rejestrowe (`register`),
- lokalne statyczne (`static`).

Zmienne **globalne** to te, których deklaracja znajduje się na zewnątrz wszystkich funkcji (również funkcji `main()`). Ich czas życia jest równy czasowi wykonywania programu. Zmienne te są automatycznie inicjowane zerami, jeśli w deklaracji zmiennej nie jest jej przypisana jawnie inna wartość.

Zmienne zdefiniowane wewnątrz funkcji, to zmienne **lokalne**. Zmienne lokalne mogą być **automatyczne** i **rejestrowe**. Są one tworzone przy każdym wywoływaniu funkcji – a co za tym idzie, za każdym razem mają one inną wartość początkową (nie są inicjowane). Zmienną taką można w momencie deklaracji jawnie zainicjować żadaną wartością. Następnie po wykonaniu funkcji zmienne lokalne przestają istnieć. Zatem czas ich życia wynosi tyle, ile czas wykonywania funkcji, w której są zadeklarowane. Jeżeli deklaracja zmiennej zostanie poprzedzona słowem kluczowym `register` oznacza to sugestię dla kompilatora, by zmienną tę zaalokował w rejestrze procesora. Możliwość spełnienia tego warunku zależy głównie od architektury procesora.

Jeśli przed definicją zmiennej lokalnej znajduje się słowo kluczowe `static`, to zmienna jest **lokalna statyczna**. Zmienne lokalne **statyczne** są tworzone i inicjalizowane przy wejściu do programu (a nie przy wejściu do funkcji) i ich wartość jest zachowywana pomiędzy wywołaniami danej funkcji. Zmienne lokalne statyczne w odróżnieniu od zmiennych globalnych są dostępne jedynie wewnątrz funkcji, w której są zadeklarowane. Ich czas życia wynosi tyle, ile czas wykonywania programu.

W celu zilustrowania różnicy między zmienną lokalną automatyczną i statyczną rozważmy następujący przykład:

```
#include <iostream>  
  
void f_auto(void)  
{  
    int k = 0;  
  
    k++;  
    cout << "Zmienna automatyczna = " << k << endl;  
}  
/* ----- */  
  
void f_static(void)  
{  
    static int l = 0;  
  
    l++;  
    cout << "Zmienna statyczna = " << l << endl;  
}  
/* ----- */  
  
int main()  
{  
    int i;
```

# Język C++. Typy danych, zmienne, funkcje

```
for(i=0; i<5; i++) f_auto();
for(i=0; i<5; i++) f_static();

return 0;
}
```

Pięciokrotne wywołanie funkcji `f_auto()` spowoduje wypisanie na konsoli pięć razy liczby 1, natomiast kolejne wywołania funkcji `f_static()` spowodują wypisanie liczb 1, 2, 3 itd.

Można zdefiniować zmienną lokalną o nazwie identycznej jak istniejąca zmienna globalna. Nowo zdefiniowana zmienna zasłania wtedy w danym lokalnym zakresie zmienną globalną. Jeśli w tym lokalnym zakresie odwołamy się do danej nazwy, to kompilator uzna to za odniesienie do zmiennej lokalnej. Zmienna globalna jest wtedy dostępna poprzez operatora dostępu `::`. Ilustruje to poniższy przykład:

```
int x = 10; // deklaracja globalna

int fun( void )
{
    int z1, z2, x = 5; // deklaracja lokalna

    z1 = x; // x lokalne (z1=5)
    z2 = ::x; // x globalne (z2=10)

    return 0;
}
```

## Funkcje

Pod pojęciem „**funkcja**” w języku C++ należy rozumieć podprogram, niezależnie od tego, czy zwraca on jakąś wartość, czy nie. W przeciwieństwie do języka PASCAL, gdzie podprogram nie zwracający wartości nazywa się procedurą, w języku C++ nie istnieje rozróżnienie na funkcje i procedury. Aby zaznaczyć, że funkcja nie zwraca żadnej konkretnej wartości, używamy typu `void` w deklaracji funkcji:

```
void funkcja()
```

Dzięki użyciu funkcji możemy niejako dodawać do języka własne instrukcje, realizujące specyficzne potrzeby, jakie stawia prawie każdy problem. Jak wiadomo, sam język C posiada niewiele instrukcji (kilkanaście). To właśnie dzięki funkcjom i klasom bibliotecznym możliwe jest wypisywanie na ekran, wczytywanie danych z klawiatury (funkcja `printf()`, `scanf()` pochodzą z biblioteki `stdio` albo strumienie `cin` i `cout` z biblioteki `iostream`), rysowanie, zapis i odczyt danych z dysku itd. Stosowanie funkcji niezwykle ułatwia programowanie, narzucając strukturalizację problemu oraz umożliwiając testowanie wybranych fragmentów programu.

## Deklaracja i definicja funkcji

Przed użyciem danej funkcji należy ją **zadeklarować**. Deklaracja funkcji (prototyp funkcji) jest to określenie typu funkcji (to jest typu wartości zwracanej przez funkcję), jej nazwy oraz liczby i typów argumentów przyjmowanych przez funkcję.

Przykładowo, deklaracja:

```
int NarysujTekst( char *tekst, float x, float y, int kolor );
```

oznajmia kompilatorowi, że funkcja `NarysujTekst()` zwraca wartość typu `int` i ma być wywołana z czterema argumentami, których typy to kolejno: `char*`, `float`, `float`, `int`. W deklaracji nie ważne są nazwy zmiennych (`tekst`, `x`, `y`, `kolor`) - można je pominąć, istotne są jedynie typy argumentów. Od momentu zadeklarowania funkcji, kompilator analizując tekst programu może sprawdzić, czy wywołanie funkcji jest poprawne pod względem ilości argumentów, ich typów oraz typu wartości zwracanej przez funkcję.

## Język C++. Typy danych, zmienne, funkcje

Jeśli funkcja jest zadeklarowana przez nas, to należy ją **zdefiniować**, inaczej mówiąc napisać co ma ona wykonywać. Definicja funkcji musi być oczywiście zgodna z jej deklaracją.

Przykładowo:

```
int NarysujTekst( char *tekst, float x, float y, int kolor )
{
    ...
    gotoxy( (int)x, (int)y );
    ...
    cprintf( "%s", tekst );
    ...
    return n;
}
```

W przypadku, gdy najpierw pojawia się definicja funkcji, deklaracja (prototyp) nie jest już potrzebna.

### Zwracanie rezultatu przez funkcję

Do zwracania wartości przez funkcję służy instrukcja `return`. Rozpatrzmy następujący przykład:

```
int silnia( int );           // deklaracja (prototyp) funkcji silnia

int main( )
{
    int n;

    n = silnia( 5 );         // wywołanie funkcji silnia
    cout << "5! wynosi << n << endl;
    silnia( 10 );           // wywołanie funkcji silnia
    return 0;
}

int silnia( int k )         // definicja funkcji silnia
{
    if( k > 1 ) return k * silnia( k - 1 );
    else return 1;
}
```

W pierwszej linii znajduje się deklaracja funkcji `silnia()`. W czasie wykonywania programu, gdy komputer napotka wywołanie funkcji `silnia()`, przekazuje do niej sterowanie. Następnie funkcja `silnia()` liczy  $k!$  i zwraca wyliczoną wartość do funkcji `main()`, a tam wartość zwrócona przez funkcję `silnia()` jest przypisywana do zmiennej `n` i wypisywana na ekran. Drugie wywołanie funkcji `silnia()` oblicza wartość  $10!$ . Wynik obliczeń jest tutaj ignorowany.

Gdy funkcja jest typu `void`, błędne jest użycie instrukcji

```
return wyrażenie;
```

a jedynym poprawnym użyciem instrukcji `return` jest po prostu

```
return;
```

co powoduje powrót z funkcji. W przypadku funkcji typu `void` nie jest konieczne umieszczanie na końcu funkcji instrukcji `return`, można natomiast w ten sposób w dowolnym miejscu funkcji wyjść z niej.

Jeżeli typ funkcji jest różny od typu `void` to wartość zwracaną przez funkcję można przypisać zmiennej lub użyć w wyrażeniu. Wartość zwracaną przez funkcję można zignorować wywołując funkcję bez przypisywania zwracanej wartości zmiennej (wtedy wywołanie funkcji nie może nastąpić po prawej stronie operatora przypisania `=`).

W przypadku funkcji typu `void` błędem jest wywołanie takiej funkcji po prawej stronie operatora przypisania lub w wyrażeniu.



## Przekazywanie argumentów do funkcji

W języku C++ argumenty do funkcji mogą być przekazywane przez wartość albo przez referencję.

### Przekazywanie argumentów przez wartość

W języku C++ przekazywanie argumentów do funkcji przez wartość jest najczęściej wykorzystywane. Ten sposób przekazywania argumentów jest jedynym sposobem w języku C.

Przekazanie argumentów przez wartość oznacza to, że w chwili wywołania funkcji tworzone są kopie poszczególnych zmiennych skojarzonych z danymi argumentami i kopiom tym jest przypisywana wartość zmiennych będących argumentami aktualnymi. Wewnątrz funkcji wszystkie operacje są wykonywane na kopiach zmiennych. Powoduje to, że algorytm funkcji nie ma możliwości modyfikacji wartości zmiennej przekazanej do funkcji jako argument aktualny. Jeżeli funkcja ma mieć możliwość modyfikacji wartości zmiennej przekazanej do funkcji jako argument to musi być przekazany wskaźnik (adres) do tej zmiennej. Należy tutaj pamiętać, że sam wskaźnik jest przekazany przez wartość. Funkcja znając adres danej zmiennej w pamięci może modyfikować jej wartość.

Gdy argumentem funkcji jest tablica to zawsze jest przekazywany wskaźnik do zerowego elementu tej tablicy (nie jest tworzona kopia tablicy). Należy o tym pamiętać, aby funkcja przypadkowo nie zmieniała wartości poszczególnych elementów tablicy.

### Przekazywanie argumentów przez referencje

W języku C++, oprócz przekazywania argumentów przez wartość, istnieje możliwość przekazywania przez referencję. Referencja jest inną nazwą (synonimem) zmiennej. Żaden operator w wyrażeniu nie działa na referencji, lecz na obiekcie, który jest przez referencję reprezentowany. W związku z tym, jeżeli argumentem funkcji jest referencja do zmiennej, to wszystkie operacje wewnątrz funkcji wykorzystujące referencję są wykonywane na zmiennej będącej w danej chwili argumentem aktualnym.

Rozważmy program, w którym są zadeklarowane 3 funkcje mające zamieniać między sobą wartości dwóch zmiennych przekazywanych jako argumenty. Argumentami funkcji `swap1()` są wartości zmiennych (przekazanie zmiennych przez wartość), funkcji `swap2()` – wskaźniki do zmiennych (przekazanie przez wartość wskaźników (adresów) zmiennych), a funkcji `swap3()` – referencje do zmiennych (przekazanie referencji do zmiennych).

```
void swap1( int x, int y )           /* przekazanie przez wartość */
{
    int tmp = x;
    x = y;
    y = tmp;
}
/* ----- */

void swap2( int *x, int *y )         /* przekazanie wskaźników */
/* przez wartość */
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
/* ----- */

void swap3( int &x, int &y )          /* przekazanie przez */
/* referencję */
{
    int tmp = x;
    x = y;
    y = tmp;
}
/* ----- */

int main( )
{
    int a = 5, b = 10;
```

## Język C++. Typy danych, zmienne, funkcje

```
swap1( a, b );      /* formalnie poprawnie, ale nie działa */
swap2( &a, &b );    /* OK */
swap3( a, b );      /* OK */

return 0;
}
```

Wywołanie funkcji `swap1()` nie powoduje zamiany wartości zmiennych `a` i `b`, gdyż funkcja ta dokonuje zamiany na kopiach argumentów aktualnych. W pozostałych dwóch przypadkach funkcje działają zgodnie z oczekiwaniami. Proszę zwrócić uwagę na nagłówki funkcji w deklaracjach i sposób ich wywołania w funkcji `main()`.

### Biblioteki i funkcje biblioteczne

W języku C++ dostępnych jest kilka standardowych bibliotek. Są to m. in. biblioteki standardowego wejścia/wyjścia, matematyczna i inne. Dostępne są również biblioteki, ułatwiające programowanie konkretnych zadań (np. biblioteki zawierające funkcje dźwiękowe, sieciowe itp.). Samemu również możemy tworzyć biblioteki.

Z reguły bibliotece funkcji towarzyszy plik nagłówkowy (`*.h`) zawierający deklaracje (prototypy) funkcji znajdujących się w bibliotece. Pliki nagłówkowe dołącza się do tekstu programu za pomocą dyrektywy preprocesora `#include`. W ten sposób kompilator może sprawdzić, czy funkcje biblioteczne zostały prawidłowo użyte. Po skompilowaniu tekstu programu linker na etapie konsolidacji dołącza kod funkcji bibliotecznych do naszego programu.

W tabeli 3 wymieniono kilka często używanych plików nagłówkowych związanych z biblioteką standardową języka C++ oraz ich odpowiedniki z biblioteki języka C.

Tabela 3.

Nagłówek C++	Nagłówek C	Opis
conio.h	conio.h	zawiera deklaracje funkcji obsługujących konsolę (np. <code>getch()</code> , <code>getche()</code> , <code>kbhit()</code> )
cctype	ctype.h	zawiera deklaracje funkcji do operowania na znakach (np. <code>isupper()</code> , <code>islower()</code> , <code>isdigit()</code> , <code>isalpha()</code> ),
cfloat	float.h	zawiera deklaracje stałych związanych z arytmetyką zmiennopozycyją
climits	limits.h	zawiera deklaracje stałych i zakresów typów całkowitych ( <code>char</code> , <code>int</code> , <code>long</code> )
	malloc.h	zawiera deklaracje funkcji do alokacji i zwalniania obszarów pamięci (np. <code>malloc()</code> , <code>realloc()</code> , <code>free()</code> ),
new		zawiera deklaracje operatorów <code>new()</code> i <code>delete()</code>
cmath	math.h	zawiera deklaracje funkcji matematycznych (np. <code>sqrt()</code> , <code>log10()</code> , <code>exp()</code> , <code>acos()</code> , <code>asin()</code> ),
cstdio	stdio.h	zawiera deklaracje typów i funkcji do obsługi standardowego wejścia i wyjścia programu w konwencji języka C (np. <code>scanf()</code> , <code>printf()</code> , <code>fopen()</code> , <code>fread()</code> , <code>fgets()</code> , <code>fwrite()</code> , <code>fclose()</code> , <code>remove()</code> , <code>rename()</code> ),
iostream iomanip		zawiera deklaracje klas do obsługi standardowego wejścia i wyjścia za pomocą strumieni programu w konwencji języka C++
cstdlib	stdlib.h	zawiera deklaracje wielu użytecznych funkcji (np. <code>rand()</code> , <code>srand()</code> , <code>abs()</code> , <code>strtol()</code> , <code>strtoul()</code> , <code>strtod()</code> ),
cstring	string.h	zawiera deklaracje funkcji do obsługi łańcuchów znaków w konwencji języka C (np. <code>strlen()</code> , <code>strcpy()</code> , <code>strcat()</code> , <code>strcmp()</code> ),
ctime	time.h	zawiera deklaracje funkcji do obsługi czasu (np. <code>time()</code> ).

# Język C++. Typy danych, zmienne, funkcje

## Generator liczb pseudolosowych

W standardzie ANSI języka C++ są zdefiniowane dwie funkcje służące do obsługi generatora liczb pseudolosowych:

- `int rand(void)` – bezargumentowa funkcja, która po każdym wywołaniu zwraca całkowitą liczbę pseudolosową z przedziału od 0 do `RAND_MAX`. Stała `RAND_MAX` jest zdefiniowana w zbiorze nagłówkowym `cstdlib` (`stdlib.h`).
- `void srand(unsigned int seed)` – funkcja służąca do inicjowania tzw. zarodka generatora liczb pseudolosowych, od którego zależy sekwencja generowanych liczb.

Ciąg liczb pseudolosowych zwracanych przez funkcję `rand()` zależy od pewnej zmiennej statycznej (zarodka, ang.: *seed*), która jest wykorzystywana przez tę funkcję do obliczania kolejnej liczby pseudolosowej. Ta sama wartość zarodka powoduje, że kolejne wywołania funkcji `rand()` będą generować taki sam ciąg wartości. Sytuacja taka ma miejsce po każdym uruchomieniu programu, gdyż domyślnie zarodek generatora jest inicjowany wartością 1. Aby po każdym uruchomieniu programu generator zwracał różne ciągi liczb, należy zainicjować zarodek dowolną liczbą całkowitą. W celu uzyskania przypadkowej liczby najczęściej wykorzystuje się funkcję `time()`, która zwraca czas systemowy liczony w sekundach od 1 stycznia 1970 roku. Uzyskuje się to pisząc instrukcję:

```
srand((unsigned) time(NULL));
```

# Język C++. Typy danych, zmienne, funkcje

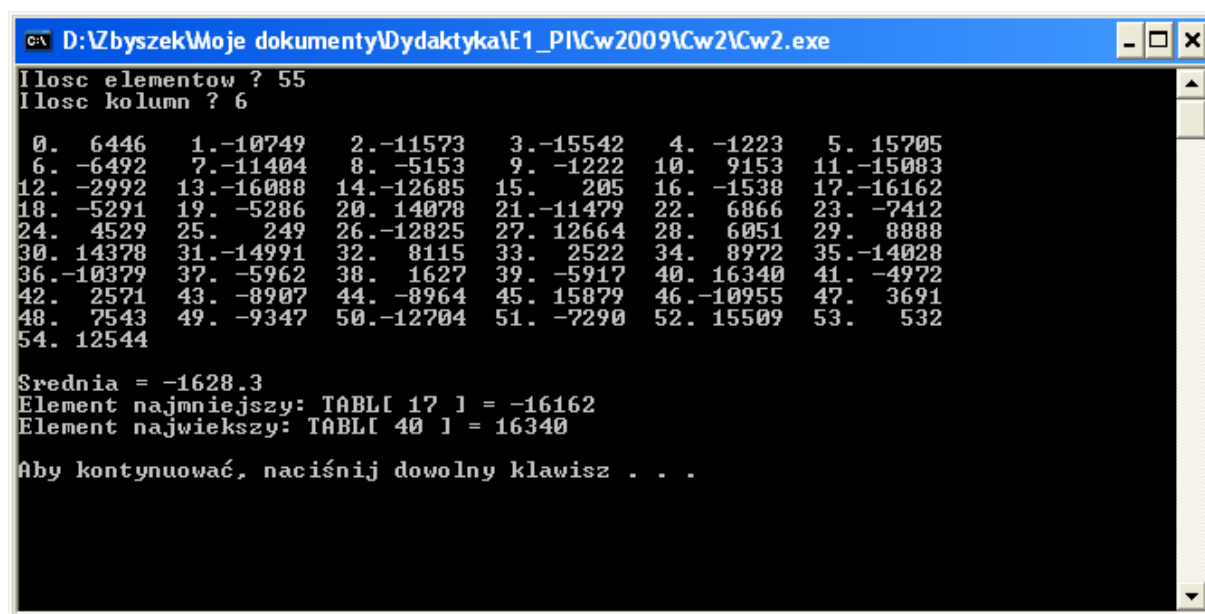
## Program ćwiczenia

Do instrukcji w pliku zip jest dołączony plik PI\_Cpp02.cpp, którego listing jest przedstawiony w załączniku 1 instrukcji. Po otwarciu nowego projektu w środowisku Microsoft Visual C++ 2005 należy go dołączyć do projektu i na nim wykonywać ćwiczenie.

Program, którego zarys („szablon”) jest zawarty w pliku PI\_Cpp02.cpp, w funkcji `main()` ma zadeklarowaną tablicę `t` typu `int`, na której będą przeprowadzane różne operacje. Po uzupełnieniu program powinien zachowywać się następująco:

1. Zachęcić użytkownika do podania liczby elementów tablicy, odczytać i sprawdzić liczbę wprowadzoną przez użytkownika (funkcja `CzytajInt()`).
2. Zachęcić użytkownika do podania liczby kolumn, w których zawartość tablicy będzie wyświetlana na ekranie. Odczytać i sprawdzić liczbę wprowadzoną przez użytkownika (funkcja `CzytajInt()`).
3. Wypełnić zadaną przez użytkownika liczbę elementów tablicy liczbami losowymi (funkcja `WypelnTabl()`).
4. Wyświetlić zawartość tablicy w zadanej przez użytkownika liczbie kolumn (funkcja `PiszTabl()`).
5. Obliczyć średnią wartość elementów tablicy, znaleźć indeks elementu najmniejszego i największego (funkcje `Srednia()`, `MinIndex()` i `MaxIndex()`).
6. Wypisać na ekranie średnią elementów tablicy oraz indeks i wartość najmniejszego i największego elementu.

Na rysunku 1 pokazano wygląd ekranu, na którym widać wyniki działania programu.



```
D:\Zbyszek\Moje dokumenty\Dydaktyka\I1_PIVc2009\Cw2\Cw2.exe
Ilosc elementow ? 55
Ilosc kolumn ? 6

0. 6446 1.-10749 2.-11573 3.-15542 4. -1223 5. 15705
6. -6492 7.-11404 8. -5153 9. -1222 10. 9153 11.-15083
12. -2992 13.-16088 14.-12685 15. 205 16. -1538 17.-16162
18. -5291 19. -5286 20. 14078 21.-11479 22. 6866 23. -7412
24. 4529 25. 249 26.-12825 27. 12664 28. 6051 29. 8888
30. 14378 31.-14991 32. 8115 33. 2522 34. 8972 35.-14028
36.-10379 37. -5962 38. 1627 39. -5917 40. 16340 41. -4972
42. 2571 43. -8907 44. -8964 45. 15879 46.-10955 47. 3691
48. 7543 49. -9347 50.-12704 51. -7290 52. 15509 53. 532
54. 12544

Srednia = -1628.3
Element najmniejszy: TABL[ 17 ] = -16162
Element największy: TABL[ 40 ] = 16340

Aby kontynuować, naciśnij dowolny klawisz . . .
```

Rys. 1. Przykładowy wygląd ekranu po uruchomieniu programu

W ramach ćwiczenia należy uzupełnić kod zawarty w pliku PI\_Cpp02.cpp i uruchomić program, według wytycznych i wymagań przedstawionych poniżej. Nie należy modyfikować istniejących w programie deklaracji i instrukcji. Numer wymagania odpowiada numerowi komentarza 'Uzupełnić' w tekście źródłowym programu:

1. Wpisać instrukcję wypisującą tekst zachęty do podania liczby elementów tablicy przez użytkownika. Przykładowy tekst zachęty jest pokazany na rys. 1.
2. Wpisać instrukcję wypisującą tekst zachęty do podania liczby kolumn, w jakiej ma być wypisana zawartość tablicy. Przykładowy tekst zachęty jest pokazany na rys. 1.
3. Wpisać instrukcję wypisującą wartość średnią liczb w tablicy z dokładności do 1 miejsca po kropce dziesiętnej oraz indeks i wartość najmniejszego elementu tablicy w sposób pokazany na rys. 1.

## Język C++. Typy danych, zmienne, funkcje

4. Napisać definicję funkcji `MinIndex()`, aby wyszukiwała i zwracała indeks najmniejszego elementu w tablicy `tabl` o rozmiarze `rozmTabl`, które są przekazane jako argumentu. Wewnątrz funkcji nie wolno odwoływać się do żadnych zmiennych globalnych i stosować instrukcji do wyprowadzania i wprowadzania danych (np. `cout <<`, `cin >>`). Jako zmiennych pomocniczych można używać tylko zmiennych lokalnych.
5. Napisać definicję funkcji `MaxIndex()`, aby wyszukiwała i zwracała indeks największego elementu w tablicy `tabl` o rozmiarze `rozmTabl`, które są przekazane jako argumentu. Wewnątrz funkcji nie wolno odwoływać się do żadnych zmiennych globalnych i stosować instrukcji do wyprowadzania i wprowadzania danych (np. `cout <<`, `cin >>`). Jako zmiennych pomocniczych można używać tylko zmiennych lokalnych.
6. Napisać definicję funkcji `Srednia()`, aby obliczała zwracała średnią arytmetyczną elementów w tablicy `tabl` o rozmiarze `rozmTabl`, które są przekazane jako argumentu. Wewnątrz funkcji nie wolno odwoływać się do żadnych zmiennych globalnych i stosować instrukcji do wyprowadzania i wprowadzania danych (np. `cout <<`, `cin >>`). Jako zmiennych pomocniczych można używać tylko zmiennych lokalnych.
7. Napisać definicję funkcji `CzytajInt()`, którą pobierze liczbę wpisaną z klawiatury za pomocą instrukcji `cin >>` i sprawdzić, czy liczba wprowadzona liczba nie przekracza ograniczeń `min` i `max` przekazanych jako argumenty. Jeżeli:
  - a.  $min \leq liczba \leq max$ , to funkcja powinna zwrócić wprowadzoną liczbę,
  - b.  $liczba > max$ , funkcja ma zwrócić wartość maksymalną `max`,
  - c.  $liczba < min$ , funkcja ma zwrócić wartość minimalną `min`.Wewnątrz funkcji nie wolno odwoływać się do żadnych zmiennych globalnych i stosować instrukcji do wyprowadzania danych (np. `cout <<`). Jako zmiennych pomocniczych można używać tylko zmiennych lokalnych.
8. Napisać definicję funkcji `PiszTabl()`, która wypisuje `rozmTabl` elementów tablicy `tabl[]` w zadanej liczbie kolumn (argument `nKol`). Sposób wyświetlania zawartości tablicy jest pokazany na rys. 1. Na wypisanie indeksu należy przeznaczyć pole o szerokości 3 znaki, na wartość elementu tablicy – pole 6 znaków. Wewnątrz funkcji nie wolno odwoływać się do żadnych zmiennych globalnych i stosować instrukcji do wprowadzania danych (np. `cin >>`). Jako zmiennych pomocniczych można używać tylko zmiennych lokalnych.
9. Zadanie dodatkowe na ocenę bardzo dobrej (5.0).  
Napisać funkcję, która będzie sortować tablicę rosnąco. Używając tej funkcji uzupełnić funkcję `main()` programu o odpowiednie wywołanie tej funkcji i powtórne wypisanie posortowanej tablicy za pomocą funkcji `PiszTabl()`.

# Język C++. Typy danych, zmienne, funkcje

## Załącznik 1. Kod źródłowy programu do uzupełnienia (plik PI\_Cpp02.cpp)

```
#include <ctime>
#include <cstdlib>
#include <iostream>
#include <iomanip>

using namespace std;

#define ROZM_TABL_MAX 1000

// Deklaracje (prototypy) własnych funkcji użytych w programie

int WypelnTabl( int tabl[], int rozmTabl );

int MinIndex( int tabl[], int rozmTabl );
int MaxIndex( int tabl[], int rozmTabl );
double Srednia( int tabl[], int rozmTabl );

int CzytajInt( int min, int max );
void PiszTabl( int tabl[], int rozmTabl, int nKol );

//-----

int main(int argc, char *argv[])
{
    int t[ROZM_TABL_MAX];
    int ilLiczba, ilKolumn;
    int idxMin, idxMax;
    double sr;

    // 1. Uzupełnic
    ilLiczba = CzytajInt( 1, ROZM_TABL_MAX );
    WypelnTabl( t, ilLiczba );

    // 2. Uzupełnic
    ilKolumn = CzytajInt( 1, 8 );

    cout << endl;
    PiszTabl( t, ilLiczba, ilKolumn );

    sr = Srednia( t, ilLiczba );
    idxMin = MinIndex( t, ilLiczba );
    idxMax = MaxIndex( t, ilLiczba );

    // 3. Uzupełnic

    system("PAUSE");

    return 0;
}

//-----
```

## Język C++. Typy danych, zmienne, funkcje

```
int WypelnTabl( int tabl[], int rozmTabl )
{
    int    ofs = RAND_MAX / 2;
    int    i;

    srand( (unsigned int)time( NULL ) );
    for( i = 0; i < rozmTabl; i++ )
        tabl[ i ] = rand( ) - ofs;

    return rozmTabl;
}

//-----

int MinIndex( int tabl[], int rozmTabl )
// 4. Uzupełnic

//-----

int MaxIndex( int tabl[], int rozmTabl )
// 5. Uzupełnic

//-----

double Srednia( int tabl[], int rozmTabl )
// 6. Uzupełnic

//-----

int CzytajInt( int min, int max )
// 7. Uzupełnic

//-----

void PiszTabl( int tabl[], int rozmTabl, int nKol )
// 8. Uzupełnic

//-----
```