



# Podstawy C++

Skrypt dla klasy III Mat. – Inf.

Wersja 1.0

**Marek Czerwiec**

**2008-11-19**

## Rozdział I, Wprowadzenie

### krótka historia C/C++

Język C powstał w latach 70. XX wieku w Bell Laboratories AT&T, opracowany przez Dennisa Ritchiego jako następcę języka B. Pod koniec lat 80. powstaje standard ANSI C.

Autorem C++ jest Bjarne Stroustrup, który sformułował założenia języka około roku 1979, natomiast sama nazwa pojawia się pod koniec 1983 roku. Obecnie standaryzacją języka C++ zajmują się odpowiednie komitety ANSI i ISO. Standard ISO C++ powstał w 1998 roku (rok po ANSI C++).

Rozważając paradygmaty (wzorce) programowania, czyli programowanie proceduralne i programowanie zorientowane obiektowo, możemy powiedzieć, że język C jest głównie zaprojektowany jako język programowania proceduralnego, natomiast C++ jest jego nadzbiorem przygotowanym do programowania zorientowanego obiektowo.

Język C++ może zmieniać się nieznacznie w zależności od używanego kompilatora, lecz podstawowe funkcje języka są niezmiennie.

### darmowe kompilatory

W Linuksie najpopularniejszym kompilatorem jest gcc (g++), pracujący z środowisku konsolowym, by skompilować zapisany w pliku tekstowym kod źródłowy programu, należy użyć polecenia:

**gcc nazwa.cpp -Wall -o nazwa** (szczegółowe informacje uzyskamy używając polecenia **man gcc**)

W Windows jednym z najpopularniejszych jest **DevC++** który wykorzystuje wersję kompilatora gcc i można pobrać go z sieci internet: <http://www.bloodshed.net/devcpp.html>)

### hello world

"Hello World" to uznany standard przy tworzeniu pierwszego programu. Zapisz kod programu w dowolnym edytorze tekstu, a następnie go skompiluj i uruchom.

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello World!";
    return 0;
}
```

Program po uruchomieniu w systemie Windows wykona się tak szybko, że nie zobaczymy efektów jego działania. By zatrzymać na chwilę wykonanie programu, możemy dodać do niego linię (przed linią **return 0;**)

```
system ("pause");
```

Co jest wywołaniem systemowego polecenia DOS. Istnieje wiele innych rodzajów zatrzymań programu, lecz to jest najprostsze, niestety związane jest ono jedynie z systemem Windows.

## struktura programu

Kod źródłowy programu to zwykły plik tekstowy, który w procesie kompilacji zamieniany jest na plik wykonywalny (w systemie Windows posiada on rozszerzenie \*.exe).

```
#include <nazwa_pliku_naglowkowego>; /* blok dyrektyw preprocesora */  
  
using namespace std; /* określenie przestrzeni nazw */  
  
typ nazwa_funkcji(parametry) /* blok własnych funkcji */  
{  
    /* kod funkcji */  
}  
  
main() /* blok funkcji głównej */  
{  
    /* kod funkcji głównej */  
}
```

Nie jest wymagane ściśle stosowanie powyższej struktury, ponieważ możemy ją dość znacznie modyfikować.

## funkcja główna

Fragmenty kodu do wielokrotnego wykorzystania możemy grupować w funkcje (omówione będą one później). Jedną z nich wyróżniamy, nazywając ją funkcją główną i od której zaczyna się wykonanie całego programu (do niej przekazywane jest sterowanie z/do systemu operacyjnego).

Najprostszy program w C++ może wyglądać w ten sposób:

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

W nawiasach klamrowych umieszczamy właściwą treść funkcji, czyli treść naszego programu, natomiast linia `return 0;` oznacza zwrócenie wartości do systemu operacyjnego określając, że nasz program działał poprawnie.

Poprzedzenie słowem `int` nazwy funkcji oznacza, że będzie ona zwracać wartość całkowitą i należy do dobrych obyczajów programowania. Ponadto funkcja główna może zawierać parametry wywołania, ale omówimy je później.

## jednostki leksykalne

Programy (kody źródłowe) w języku C++ pisane są przy pomocy edytorów tekstowych i składają się z jednostek leksykalnych:

- identyfikatory obiektów (np. zmiennych) – unikalny ciąg liter i cyfr rozpoczynający się od litery, rozróżniana jest wielkość liter, nie może rozpoczynać się od cyfry,
- słowa kluczowe – charakterystyczne dla danego języka programowania, zastrzeżone, zatem nie mogą być

używane w znaczeniu identyfikatorów obiektów,

- literały – wartość wpisana bezpośrednio w kod programu, np.: tekst = "ala ma kota"; gdzie tekst jest identyfikatorem obiektu, a "ala ma kota" jest literałem. Literał napisowy może zawierać znaki specjalne, np.: `\n`
- separatory

Ponadto mamy komentarze, spacje, znaki tabulacji i znaki specjalne, które nie są jednostkami leksykalnymi.

## słowa kluczowe

and	const_cast	float	operator	static_cast	using
and_eq	continue	for	or	struct	virtual
asm	default	friend	or_eq	switch	void
auto	delete	goto	private	template	volatile
bitand	do	if	protected	this	wchar_t
bitor	double	inline	public	throw	while
bool	dynamic_cast	int	register	true	xor
break	else	long	reinterpret_cast	try	xor_eq
case	enum	mutable	return	typedef	
catch	explicit	namespace	short	typeid	
char	export	new	signed	typename	
class	extern	not	sizeof	union	
const	false	not_eq	static	unsigned	

## zasady pisania kodu, komentarze

- I. Instrukcję kończymy średnikiem (niekoniecznie każdą linijkę!),
- II. Nie używamy polskich liter.
- III. Stosujemy komentarze! Każdy blok kodu powinien być opisany - do czego służy, a szczególnie zawiłe linie kodu warto by były opisane linia po linii. Kompilator ignoruje komentarz czyli znaki pozostawione w jednej linii po postawieniu `// treść_komentarza` a jeśli chcemy wykonać dłuższy komentarz, stosujemy ramy `/* komentarz */` który może zawierać więcej niż jedną linię.
- IV. Stosujemy wcięcia i odstępy. Język C++ jest językiem, w którym tak naprawdę możemy pisać jak nam się to podoba, porównaj kod:

```
#include<iostream> using namespace std;int main(){cout <<"Hello world"<<'\\n'
; getchar();return 0;}
```

z kodem:

```
#include<iostream>
using namespace std;

int main()
{
    cout <<"Hello world"<<'\\n';
    getchar(); return 0;
}
```

Zapis kodu dla kompilatora nie ma znaczenia, ponieważ 'białe znaki' (spacje, dodatkowe linie, wcięcia) są ignorowane. Kompilator 'czyta' instrukcję od średnika do średnika.

W kodzie napisanym według pewnych niepisanych zasad, zdecydowanie łatwiej znaleźć interesujące nas fragmenty i zdecydowanie łatwiej zrozumieć, co taki kod robi, dlatego stosujemy trzy proste metody: wstawianie pustych linii, stosowanie wcięć oraz pisanie każdej instrukcji w oddzielnej linii.

## Rozdział II, Typy

### typy liczbowe

#### typy całkowite:

identyfikator typu	rozmiar (byte)	uwagi
char	1	-128 ÷ 127
int	2 (4 w trybie 32 bit)	-32768 ÷ 32767 (-2147483648 ÷ 2147483647)
short	2	-32768 ÷ 32767
long	4	-2147483648 ÷ 2147483647
unsigned char	1	0 ÷ 255
unsigned	2 (4 w trybie 32 bit)	0 ÷ 65535 (0 ÷ 4294967295)
unsigned short	2	0 ÷ 65535
unsigned long	4	0 ÷ 4294967295
enum	2	-32768 ÷ 32767

#### typy rzeczywiste:

identyfikator typu	rozmiar	uwagi
float	4	1,2E-38 ÷ 3,4E+38 dokładność 7 cyfr
double	8	2,2E-308 ÷ 1,8E+308 dokładność 15 cyfr
long double	10	3,4E-4932 ÷ 1,2E+4932 dokładność 19 cyfr

w szczególności mamy **cztery typy podstawowe**:

- int - zmienna przechowująca liczby całkowite
- char - zmienna przechowująca znaki (litery, itp.), lub liczbę z zakresu -128 ÷ 127
- float - zmienna przechowująca liczby rzeczywiste
- double - zmienna przechowująca liczby rzeczywiste

modyfikowane poprzez kwalifikatory (**modyfikatory**):

- unsigned - zmienna przechowuje tylko wartości dodatnie
- signed - zmienna przechowuje wartości dodatniej ujemne
- short - zmienna ma długość przynajmniej 2 bajtów
- long - zmienna ma długość przynajmniej 4 bajtów
- const - wartości zmiennej nie można modyfikować, jest to stała

### typ wyliczeniowy

Typ enum pozwala wybrać wartości jakie mogą być przechowywane w zmiennej. Przykład:

```
enum pory_roku
```

```
{
    WIOSNA, LATO,
    JESIEN, ZIMA
};

pory_roku pora = WIOSNA; //Przykład przypisania:
```

nie jest możliwe przypisanie zmiennej wartości z poza podanego zakresu

## typ logiczny

typ `bool`, który może przyjąć wartości `false` lub `true`, przykładowo:

```
void main()
{
    bool w=true,p;

    while(w)
    {
        if(!p) w=0;
    }
}
```

Wszystkie powyższe typy są typami wbudowanymi (built-in) lub ścisłymi (strict)

## typ string (klasa string)

Biblioteka standardowa posiada zaimplementowaną klasę `string`, która daje jednolity, niezależny od systemu i bezpieczny interfejs do manipulowania napisami. Aby móc korzystać z klasy `string` należy dołączyć plik nagłówkowy:

```
#include <string>
```

tworzyć nowe obiekty tego typu możemy następująco:

```
string napis1;

napis1 = "text";

string napis2( "text" );

string napis3 = "text";

cout << napis1 << endl
     << napis2 << endl
     << napis3 << endl;

string napis4(10, 'X');

cout << napis4;
```

powyższy kod da w wyniku:

```
text
text
text
XXXXXXXXXX
```

Klasą string zajmiemy się osobno, ponieważ dodatkowo zawiera ona także metody operacji na łańcuchach znaków.

## deklaracja zmiennych (stałych) i ich zasięg

Deklarując zmienną musimy podać jej **modyfikator typ nazwę** (etykietę), ewentualnie wartość początkową. Kompilator ma swoje nazwy zastrzeżone (min. słowa kluczowe), oraz rozróżnia małe i duże litery. W nazwach zmiennych nie wolno używać spacji ani polskich liter, nazwa nie może rozpoczynać się cyfrą.

Zwyczajowo zmiennej nadaje się nazwę opisową, która pomoże określić przeznaczenie zmiennej, natomiast dla zmiennych sterujących licznikami pętli stosuje się nazwy: **i,j,k** itp.

Gdy nazwa zmiennej składać się będzie z wielu słów stosujemy oddzielanie słów podkreślnikiem (np. liczba\_a) lub (co jest nowszym podejściem) pierwszą literę nowego słowa piszemy dużą (np.: FunkcjaKwadratowa).  
Składnia deklaracji zmiennej:

```
modyfikator typ NazwaZmiennej;
```

Przy deklaracji kilku zmiennych tego samego typu, możemy podać ich nazwy oddzielone przecinkami. Dodatkowo zmienna może być zainicjalizowana wartością (co jest zalecane), w formie:

```
typ nazwa = wartosc; // notacja języka C
// lub
typ nazwa (wartosc); // notacja języka C++
```

Przykładowo:

```
int i = 0; // notacja języka C (aktualna także w C++)
int i(0); // notacja wprowadzona z językiem C++
```

Zmienne nie są automatycznie inicjalizowane, zatem zmienna nie zainicjalizowana posiada wartość taką, jaka się jej trafiła w przeznaczonym dla niej kawałku pamięci. Wartość taką nazywamy wartością osobliwą (singular), czyli taką o której nic nie wiemy, a ponadto program nie ma nad nią żadnej kontroli – dlatego należy inicjalizować zmienne.

## modyfikatory

Obiekty (zmienne lub stałe) mogą być w zależności od potrzeby umieszczane w pamięci w różny sposób (często również determinuje to ich zachowanie w programie), co sterowane jest przez odpowiednie modyfikatory:

- **register** - oznacza, że zmienna powinna być (w miarę miejsca, o czym decyduje kompilator) trzymana w rejestrze procesora, a nie w pamięci.
- **const** - oznacza, że obiekt jest stały. W konsekwencji istnieje obowiązek zainicjalizowania go i nie wolno później zmieniać jego wartości
- **volatile** - oznacza, że nie ma się wyłączości do podanego obiektu, zatem kompilatorowi nie wolno używać w celach optymalizacyjnych wartości odczytanej we wcześniejszej operacji; musi on za każdym odczytem jej wartości dokonać jej fizycznego odczytu
- **static** - oznacza, że obiekt taki istnieje przez cały czas, niezależnie od zasięgu, który go używa. Dla funkcji oraz zmiennych globalnych oznacza z kolei zniesienie zewnętrznego symbolu obiektu lub funkcji (tzn. poza bieżącą jednostką kompilacji, czyli plikiem, nic nie może z tego korzystać)
- **extern** - oznacza, że następująca deklaracja nie jest fizyczną deklaracją, lecz zaimportowaniem owej deklaracji z innej jednostki kompilacji (modyfikatory `'static'` i `'extern'` wzajemnie się wykluczają)

Jeżeli zmienna w funkcji jest często wykorzystywana, można ją zadeklarować ze słowem **register**. Wtedy jeżeli tylko to będzie możliwe kompilator umieści zmienną w rejestrze procesora. Warto takie zmienne wykorzystywać zwłaszcza do zmiennych sterującymi pętlami. Np:

```
for(register int i=0;i<=32476;++i)...
```

Można również argument funkcji zadeklarować jako zmienną rejestrową.

```
int funkcja (register int zmienna1)...
```

Przeważnie przy odpowiednim wykorzystaniu może znacznie przyspieszyć wykonywanie funkcji, a tym samym szybsze działanie programu.

## stałe

Stała jest zmienną, której wartości nie możemy zmienić. Deklaracja może być postaci:

```
#define nazwa wartosc // wykorzystanie dyrektywy preprocesora
```

lub

```
const typ nazwa = wartosc // wykorzystanie modyfikatora
```

Stałe powinny być zainicjowane z pewną wartością początkową. Z punktu widzenia komputera stała nie różni się od zmiennej, bo miejsce w pamięci trzeba zarezerwować, umieścić w tablicy i zapamiętać jej identyfikator i adres. Przykład:

```
#define maksimum_zakresu 22;
#define nazwisko 'Kowalski';

const int maksimum_zakresu=22;
const float PI = 3.14;
```

Konstrukcja z `const` jest zalecana ponieważ od razu zadeklarowany jest typ stałej. Jest jednocześnie deklaracja, definicja i zainicjowanie stałej.

## typowanie literałów

Rodzaje literałów, sposoby ich rozpoznawania i odpowiadające im typy (np.: podczas deklarowania stałych):

- 3.14 - double
- 3.14F - float
- 3.14L - long double
- 3 - int
- 3L - long int
- 3U - unsigned int
- 3UL - unsigned long int

## aliasy (referencje)

Alias stosowane są w przypadku gdy chcemy nadać nową nazwę dla typu już istniejącego (np.: w celu skrócenia jego nazwy):

```
typedef unsigned int size_t;
```

typ `size_t` istnieje w bibliotece i taka jest jego definicja. Według standardu, jest to typ danej zwracanej przez `sizeof`.

## zasięg zmiennej

Każda zmienna ma określony zasięg, czyli nie wszędzie jest ona dostępna. Największy zasięg ma zmienna która jest zdefiniowana na samym początku pliku przed definicją funkcji.

Kod programu składa się z bloków oznaczanych nawiasami { }, i jeżeli zmienna została zdefiniowana w bloku, to nie można jej używać w instrukcjach które nie należą do tego bloku.

W języku C++ można też mieszać kod z deklaracjami zmiennych, deklarując zmienne w miarę potrzeby. Pamiętać należy jednak o tym, że zadeklarowanie zmiennej wewnątrz jakiegoś bloku jest jednocześnie ukryciem tej zmiennej dla 'zewnętrznych' bloków programu.

## Rozdział III, Operatory

### operatory bitowe

<<	przesuwanie bitów w lewo, składnia: zmienna << liczba bitów
>>	przesuwanie bitów w prawo.
~	negacja bitowa, składnia: ~zmienna
&	koniunkcja bitowa, składnia: zmienna1 & zmienna2
^	różnica symetryczna, składnia: zmienna1 ^ zmienna2
	alternatywa bitowa, składnia: zmienna1   zmienna2

### operatory logiczne

!	negacja logiczna, składnia: !argument
&&	ioczyn logiczny, składnia: argument1 && argument2
	suma (alternatywa) logiczna, składnia: argument1    argument2
==	równość, składnia: argument1 == argument2
!=	nierówność, składnia: argument1 != argument2

### operatory porównania (relacji)

<	TRUE gdy lewy argument1 jest mniejszy od prawego argumentu2, składnia: argument1 < argument2
>	TRUE gdy lewy argument1 jest większy od prawego argumentu2, składnia: argument1 > argument2
<=	TRUE gdy lewy argument1 jest mniejszy lub równy od prawego argumentu2, składnia: argument1 <= argument2
>=	TRUE gdy lewy argument1 jest większy lub równy od prawego argumentu2, składnia: argument1 >= argument2

### operatory przypisania

<b>a = b</b>	do zmiennej a podstaw wartość zmiennej b, lub <b>a = wyrażenie</b> (wtedy zostanie podstawiony wynik obliczeń)
<b>a *= b</b>	skrótowy zapis a = a * b
<b>a /= b</b>	skrótowy zapis a = a / b
<b>a += b</b>	skrótowy zapis a = a + b
<b>a -= b</b>	skrótowy zapis a = a - b
<b>a %= b</b>	skrótowy zapis a = a % b

### operatory arytmetyczne

+	dodawanie
-	odejmowanie / zmiana znaku
*	mnożenie
/	dzielenie
%	dzielenie modulo (zwraca resztę z dzielenia)

### operatory zwiększania i zmniejszania (inkrementacji i dekrementacji)

<b>i++</b>	skrót wyrażenia i = i + 1 lub i +=1; (postinkrementacja)
<b>i--</b>	skrót wyrażenia i = i - 1 lub i -=1; (postdekrementacja)
<b>++i</b>	preinkrementacja – czyli zmienna zostanie powiększona przez jej użyciem
<b>--i</b>	predekrementacja – czyli zmienna zostanie zmniejszona przed jej użyciem

Operatory ++x, --x zmieniają zmienną przed wywołaniem instrukcji, a x++, x-- po zakończeniu danej instrukcji.

### sizeof

---

Przekazuje liczbę bajtów będącą rozmiarem wyrażenia lub specyfikatora typu. Składnia:

```
sizeof(typ);          // lub  
sizeof wyrażenie;
```

## Rozdział IV, Instrukcje wejścia / wyjścia

Możemy wyróżnić dwa podejścia do obsługi wejścia/wyjścia: praca ze strumieniami i biblioteką <iostream> lub wykorzystanie biblioteki <stdio>. Wykorzystanie biblioteki <stdio> jest traktowane jako archaizm języka C.

### instrukcje wejścia / wyjścia biblioteki <iostream>

Plik nagłówkowy <iostream.h> (z języka C) lub <iostream> (z języka C++) zawiera definicję obsługi strumieni (io- input/output, stream-strumień). Podstawowe instrukcje to **cout** (see out) i **cin** (see in) oraz operatory << lub >>, zwane operatorami wstawiania do/pobierania z strumienia.

```
#include <iostream>           //dołączenie plików nagłówkowych
#include <string>
using namespace std;        // określenie przestrzeni nazw
int main()
{
    cout <<" Podaj imie: ";
    string imie; cin >>imie;
    cin.ignore();           // reset stanu klawisza ENTER
    cout <<"Witaj "<<imie<<endl;    // endl to przejście do następnej linii
    cout <<"w swoim pierwszym programie"<<endl;
    cout <<"Nacisnij ENTER aby zakonczyc"<<endl;
    cin.get();              // wstrzymanie programu
    return 0;
}
```

Przykładowe programy z deklaracją zmiennych i ich wykorzystaniem:

```
#include <iostream>
using namespace std;
int main()
{
    float Liczba;           // deklaracja zmiennej zmiennopozycyjnej
    cout <<"Podaj wartosc liczby: "; // wyswietlenie komunikatu na ekranie
    cin >>Liczba;           // pobranie z klawiatury
    cin.ignore();           // ignorowanie stanu klawisza ENTER
    cout <<endl<<"Wartosc liczby podanej przez Ciebie to "<<Liczba<<endl;
    cout <<"Typ float zajmuje "<<sizeof(float)<<" bajt(y/ow)."<<endl;
    cout <<"Twoja zmienna zajmuje "<<sizeof(Liczba)<<" bajt(y/ow)."<<endl;
    cout <<"Nacisnij ENTER aby zakonczyc"<<endl;
    cin.get();
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    double moj_wzrost;
    int wiek;

    cout <<"Podaj swoj wzrost (w metrach): ";
    cin >>moj_wzrost;
    cout <<"Podaj swoj wiek: ";
    cin >>wiek;
    cout <<endl<<"Masz "<<moj_wzrost<<" metrow wzrostu.";
```

```

cout <<endl<<"Typ double zajmuje "<<sizeof(double)<<" bajt(y/ow).";
cout <<endl<<"Zmienna zajmuje "<<sizeof(moj_wzrost)<<" bajt(y/ow).";
cout <<endl<<"Masz "<<wiek<<" lat.";
cout <<endl<<"Typ int zajmuje "<<sizeof(int)<<" bajt(y/ow).";
cout <<endl<<"Zmienna zajmuje "<<sizeof(wiek)<<" bajt(y/ow).";
}
#include <iostream>

using namespace std;

int main()
{
    cout << "Wprowadz dwie liczby:";

    int a, b; cin >> a >> b;

    cout << "\nSuma      : " << a + b << " zajmuje pamieci : " << sizeof a+b <<" bajty";
    cout << "\nRoznica  : " << a - b << " zajmuje pamieci : " << sizeof a-b <<" bajty";
    cout << "\nIloczyn   : " << a * b << " zajmuje pamieci : " << sizeof a*b <<" bajty";
    cout << "\nIloraz    : " << a / b << " zajmuje pamieci : " << sizeof a/b <<" bajty";
    cout << "\nReszta    : " << a % b << " zajmuje pamieci : " << sizeof a%b <<" bajty";

    system("pause");
}

```

Możemy także nie deklarować przestrzeni nazw:

```

#include <iostream>

int main( )
{
    std::cout << "Podaj dwie liczby a i b" << std::endl;
    int a, b;

    std::cout << "a:";
    std::cin >> a;
    std::cout << "b:";
    std::cin >> b;
    std::cout << "a+b=" << a+b << std::endl;
}

```

Przeanalizuj poniższy przykład, zwracając uwagę na wartość zmiennej **x** przy każdym wywołaniu:

```

#include <iostream>

using namespace std;

int funkcja(int x);

int main(void)
{
    cout<<" x = 5 przy kazdym z wywolan funkcji"<<endl<<endl;

    int x=5;
    cout<<" x = "<<x<< ", funkcja(++x) = "<<funkcja(++x)<< ", x = "<<x<<endl;

    x=5;
    cout<<" x = "<<x<< ", funkcja(x++) = "<<funkcja(x++)<< ", x = "<<x<<endl;

    x=5;
    cout<<" x = "<<x<< ", funkcja(--x) = "<<funkcja(--x)<< ", x = "<<x<<endl;
}

```

```
x=5;
cout<<" x = "<<x<<", funkcja(x--) = "<<funkcja(x--)<<", x = "<<x<<endl<<endl;

system("pause");
}

int funkcja(int x)      /* funkcja jedynie zwraca podana jej wartosc */
{
    return x;
}
```

## instrukcje wejścia / wyjścia biblioteki <stdio>

Język C posiadał bibliotekę <stdio.h> służącą do obsługi ekranu, która zawierała dwa polecenia `printf` i `scanf`, podobne nieco w swoim działaniu do poleceń `write` i `read` z języka Pascal.

```
#include <stdio.h>

int main( )
{
    printf("Hello world!\n");
    getchar();
}
```

**funkcja `printf()`** jest dostępna jedynie po zadeklarowaniu pliku nagłówkowego <stdio.h> i jest funkcją odpowiedzialną za wyświetlenie napisu określonego w cudzysłowach na ekranie

```
#include <stdio.h>      /* dodajemy plik naglowkowy */
int main(void)
{
    printf("Hello World");
    printf("Hello World");
}
```

Poprzedni kod spowoduje wyświetlenie napisu Hello WorldHello World, ponieważ za przejście do następnej linii odpowiedzialny jest tag (znacznik) specjalny `\n` którego użyć możemy w dowolnej części tekstu.

```
#include <stdio.h>      /* dodajemy plik naglowkowy */
int main(void)
{
    printf("Hello World \n");
    printf("Hello World");
}
```

### Znaki specjalne

<code>\n</code> - przejście do nowej linii	<code>\a</code> - sygnał dźwiękowy
<code>\t</code> - tabulacja pozioma	<code>\\</code> - backslash
<code>\v</code> - tabulacja pionowa	<code>\?</code> - znak zapytania
<code>\b</code> - skasowanie znaku w lewo	<code>\'</code> - apostrof
<code>\r</code> - powrót karetki	<code>\"</code> - cudzysłów
<code>\f</code> - wysunięcie strony	

Znaki specjalne to relik z czasów gdy komputer wyniki swojej pracy drukował na drukarce, zamiast wyświetlać na ekranie.

**funkcja `scanf()`** jest funkcją formatowanego wejścia ze standardowego strumienia wejściowego (`stdin`), czyli funkcja ta jest odpowiedzialna za przypisanie wartości podanej z klawiatury do określonej zmiennej.

Interpretacja pobranych przez funkcję `scanf` znaków nastąpi zgodnie z życzeniem programisty określonym przez zadany funkcji format, który decyduje, czy pobrane znaki zostaną zinterpretowane np. jako liczba całkowita, znak, łańcuch znaków (napis), czy też w inny sposób.

Podając nazwy (identyfikatory) zmiennych należy poprzedzić je operatorem adresowym `&`.

```
#include <stdio.h>
int main(void)
{
    int liczba1, liczba2;
    printf("Podaj liczbę: \n");
    scanf( "%d" , &liczba1 );
    printf( " wpisałeś:   %d\n\n " , liczba1 );
    printf("Podaj liczbę: \n");
    scanf( "%d" , &liczba2 );
    printf( " wpisałeś:   %d\n\n " , liczba2 );
    printf("Liczba 1 wynosi %d a liczba 2 wynosi %d\n",liczba1,liczba2);
}
```

Używając funkcji `scanf`, najpierw w cudzysłowie podajemy typ (wzorec konwersji) pobieranej zmiennej, a po przecinku i ze znakiem `&` podajemy nazwę zmiennej.

By wypisać zawartość zmiennej na ekranie, używamy wywołania jej typu (jeszcze w cudzysłowie) a poza cudzysłowem, po przecinku jej nazwę. Jeśli chcemy wypisać kilka zmiennych jednocześnie - przepis postępowania mamy w ostatniej linijce - wystarczy podać ich typy oraz nazwy tych zmiennych oddzielonych przecinkiem, a zostaną one wypisywane w kolejności w jakiej zostały podane ich nazwy.

### Wzorce konwersji

Wzorec formatu:    % [przełączniki] [szerokość\_pola] [.precyzja] [rozmiar] typ

- **%s** – jako łańcuch znaków (s - string – łańcuch), format domyślny funkcji `printf()`. Składnia `printf("%s","napis");` jest identyczna z `printf("Jakis napis");`
- **%c** – jako znak (c - character – znak), np.: `printf("%c",'X');`
- **%d** – jako liczbę całkowitą dziesiętnie (d - decimal – dziesiętny), np.: `printf("%d", 1994);`
- **%f** – jako liczbę rzeczywistą dziesiętnie (f - floating point - zmienny przecinek) – np.: `printf("%f", 3.1416);` lub z formatowaniem pola wyjściowego `printf("%f3.2", 3.14159);`
- **%o** – jako liczbę całkowitą ósemkową (o - octal – ósemkowa), np.: `printf("%o", 255);`
- **%x** – jako liczbę całkowitą szesnastkową (x - hexadecimal – szesnastkowa), gdzie `%x` lub `%X` - cyfry szesnastkowe a,b,c,d,e,f lub A,B,C,D,E,F
- **%ld** – liczba całkowita "długa" - long int.
- **%Lf** – liczba rzeczywista poczwórnej precyzji typu long double float
- **%e** – liczba w formacie wykładniczym typu 1.23e-05 (0.0000123)
- **%g** – automatyczny wybór formatu `%f` albo `%e`

Przykład (pobieranie, obliczanie, podstawianie i wyświetlanie danych)

```
# include <stdio.h>
#include <stdlib.h>
int main()
{
    float x,y,wynik;
    printf("Zamieniam ułamki zwykłe na dziesiętne\n");
    printf("\nPodaj licznik ułamka: ");
    scanf("%f", &x);                   /* pobiera liczbę z klawiatury */
```

```
printf("\nPodaj mianownik ulamka: ");
scanf( "%f", &y);
wynik = x / y;          /* tu wykonuje sie dzielenie */
printf("\n %f : %f = %f", x, y, wynik);
system("pause");      // wywołanie DOSowego pause z biblioteki stdlib
return 0;
}
```

Posługując się różnymi sposobami formatowania liczb możemy zażądać wydrukowania liczb w najwygodniejszej dla nas formie, np.: zamieniając liczby dziesiętne na szesnastkowe:

```
#include <stdio.h>
int liczba;
int main(void)
{
    printf("Podaj liczbe dziesiętna całkowita ? \n");
    scanf("%d", &liczba);
    printf("\nSzesnastkowo to wynosi:  ");
    printf("%x",liczba);
    printf("\nPodaj liczbe SZESNASTKOWA-np. AF - DUZE LITERY: \n");
    scanf("%X", &liczba);
    printf("%s","\nDziesiętnie to wynosi:  ");
    printf("%d\n",liczba);
}
```

## Rozdział V, Instrukcje sterujące

### instrukcja prosta, blok instrukcji, zasięg zmiennej

**Instrukcja prosta** jest pojedynczą instrukcją (wyrażeniem) zakończoną średnikiem. **Blok instrukcji** (instrukcja złożona) jest to grupa instrukcji prostych, ujętych w klamry { }. Blok taki posiada osobny, lokalny zasięg, toteż identyfikatory w nim definiowane mają zasięg tylko wewnątrz tego bloku i mogą **przysłaniać** (hide) identyfikatory znajdujące się w wyższym zasięgu.

```
#include <iostream>
using namespace std;
int x = 1; // zmienna globalna
int main()
{
    int x = 2; // zmienna lokalna
    { // lokalny blok instrukcji
        int x = 3; // zmienna lokalna lokalnego bloku
        cout << "Wartosc zmiennej x : " << x << endl;
    }
    cout << "Wartosc zmiennej x (lokalnej) : " << x << endl;
    cout << "Wartosc zmiennej x (globalnej) : " << ::x << endl;
}
```

Operator :: jest operatorem zasięgu, który pozwala uzyskać identyfikator z najwyższego zasięgu.

### if ... else

Instrukcja sterująca **if...else** jest instrukcją decyzyjną (warunkową) postaci:

```
if(warunek)
{
    // instrukcje do wykonania gdy warunek jest prawdziwy
}
else
{
    // instrukcje do wykonania gdy warunek jest fałszywy
}
```

Jeśli **warunek** jest prawdziwy (true) to wykonane są instrukcje bezpośredni po **if**, w przeciwnym wypadku, gdy warunek jest fałszywy (false) instrukcje po **if** są pomijane, natomiast wykonywane są instrukcje po **else**.

Możliwe jest także sprawdzanie wielu warunków:

```
if(warunek1)
{
    // instrukcje 1
}
else if (warunek2)
{
    // instrukcje 2
}
else if (warunek3)
{
```

```

    // instrukcje 3
}
else
{
    // instrukcje 4
}

```

Instrukcja ta może mieć także postać:

```

if(warunek)
{
    // instrukcje
}

```

Przykład:

```

#include <iostream>
using namespace std;
int main()
{
    int a=0,b=0;
    cout <<endl << " Podaj a : "; cin >> a;
    cout <<endl << " Podaj b : "; cin >> b;
    if(a==b) cout <<endl << " a jest rowne b ";
    else     cout <<endl << " a jest rozne od b";
    return 0;
}

```

Przykład:

```

#include <iostream>
using namespace std;
int main()
{
    cout << "podaj x : "; int x; cin >>x;

    if(x%2==0)
    {
        cout << " liczba "<<x<<" jest parzysta" <<endl;
    }
    else if (x%3==0)
    {
        cout << " liczba "<<x<<" jest podzielna przez 3" <<endl;
    }
    else if (x%5==0)
    {
        cout << " liczba "<<x<<" jest podzielna przez 5" <<endl;
    }
    else
    {
        cout << x <<" nie jest podzielna przez 2, 3 lub 5" <<endl;
    }

    system("pause");
    return 0;
}

```

**Ćwiczenie, napisz programy:**

1. Program wczytuje dwie liczby, oblicza pole i obwód prostokąta, sprawdza, czy prostokąt jest kwadratem.
2. Program wczytuje 2 liczby i określa czy pierwsza jest wielokrotnością drugiej.

```

#include <iostream>
using namespace std;

```

```

int main()
{
    int a=0,b=0;
    cout<<"\n Podaj a : "; cin >> a;
    cout<<"\n Podaj b : "; cin >> b;
    cout<<"\n Pole prostokata : " << a*b;
    cout<<"\n Obwod          : " << 2*a+2*b;
    if(a==b) { cout<<"\n Prostokat jest kwadratem "; }
    else     { cout<<"\n Prostokat nie jest kwadratem "; }

    return 0;
}
#include <iostream>
using namespace std;
int main()
{
    int a=0,b=0;
    cout<<"\n Podaj a : "; cin >> a;
    cout<<"\n Podaj b : "; cin >> b;
    if(a%b==0) { cout<<"\n Liczba a jest podzielna przez b "; }
    else if (b%a==0) { cout<<"\n Liczba b jest podzielna przez a "; }
    else if (b%a!=0 && a%b!=0) { cout<<"\n Liczby nie sa podzielne przez siebie "; }
    return 0;
}

```

**Ćwiczenie:** napisz program: rozwiązywanie równania kwadratowego

```

#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    float a=0, b=0, c=0, delta=0;
    cout <<"Podaj a= "; cin >> a;
    cout <<"Podaj b= "; cin >> b;
    cout <<"Podaj c= "; cin >> c;
    if (a==0)
    {
        if (b==0)
        {
            if (c==0)
            {
                cout << endl << "Nieskonczenie wiele rozwiazan " << endl;
            }
            else
            {
                cout << endl << "Brak rozwiazan.\n ";
            }
        }
        else
        {
            if (c==0)
            {
                cout<< endl <<"x = 0 "<<endl;
            }
            else
            {
                cout << endl << " Funkcja liniowa, jeden pierwiastek : "<< -c/b << endl;
            }
        }
    }
    else
    {

```

```

    delta=(b*b)-4*a*c;
    cout << endl << "Delta tego rownania wynosi: " << delta << endl;
    if(delta < 0)
    {
        cout << "Brak pierwiastkow rownania, gdz delta<0 " << endl;
    }
    if(delta == 0)
    {
        cout << "Jeden pierwiastek rownania: " << -b/2*a << endl;
    }
    if(delta > 0)
    {
        cout << "Dwa pierwiastki : " << "x1= " <<(-b-sqrt(delta))/(2*a) << " i ";
        cout << " x2= " << (-b+sqrt(delta))/(2*a) << endl;
    }
}
cout << endl << "\a " << endl;
return 0;
}

```

## switch...case

Instrukcja **switch** pozwala na wybór z wielu opcji. Jeśli zmienna, gdy mamy wybrać jedną z wielu opcji, pozornie lepszą instrukcją niż `if...else` będzie instrukcja `switch...case`. Jest niewiele sytuacji wymagających użycia tej instrukcji.

Po słowie kluczowym **switch** podajemy zmienną przełącznikową, a następnie **case** podajemy możliwe opcje wyboru. Jeżeli żaden z warunków nie zostanie spełniony, wówczas wykonana będzie instrukcja `default`. Przełącznik musi być typu porządkowego (typ całkowity). Instrukcje nie muszą być ujęte w klamry.

```

switch(warunek)
{
    case wartość1:
        // jeżeli warunek == wartość1 to wykonuje instrukcje_1
        break;
    case wartość2:
        // jeżeli warunek == wartość2 to wykonuje instrukcje_2
        break;
    default:
        // instrukcja_domyślna (wykonywana gdy inne nie były wykonane);
        break;
};

```

Słowo **break** powoduje przerwanie działania instrukcji w danym miejscu - jeżeli nie użylibyśmy tego słowa, wówczas wszystkie instrukcje poniżej zgodnego warunku także zostałyby wykonane.

Instrukcja **break** powoduje, że wykonanie programu jest kontynuowane od pierwszej instrukcji po strukturze `switch`.

Przykład:

```
#include <iostream>

using namespace std;

int main()
{
    cout << " Podaj i = ";

    int i; cin >> i;

    switch(i)
    {
        case 1:
            cout << "wpisales: jeden";
            break;
        case 2:
            cout << "wpisales: dwa";
            break;
        case 3:
            cout << "wpisales: trzy";
            break;
        default:
            cout << "wpisales liczbe ktora nie jest 1,2 ani 3";
            break;
    }
}
```

Przykład (rola instrukcji break):

```
# include <iostream>

using namespace std;

int main()
{
    int Numer_Dnia;

    cout<<"\nPodaj numer dnia tygodnia\n";
    cin>>Numer_Dnia;

    switch(Numer_Dnia)
    {
        case 1: cout<<"PONIEDZIALEK.";
        case 2: cout<<"WTOREK";
        case 3: cout<<"SRODA.";
        case 4: cout<<"CZWARTEK.";
        case 5: cout<<"PIATEK.";
        case 6: cout<<"SOBOTA.";
        case 7: cout<<"NIEDZIELA.";
        default: cout<<"\n ** nie ma takiego dnia tygodnia **";
    }

    return 0;
}
```

## struktura powtórzenia – pętla licznikowa for

W celu wykonania kodu określoną ilość razy używamy petli licznikowej postaci:

```
for (Warunek_inicjujący; Warunek_logiczny; Warunek_kroku)
```

```
{  
    // Instrukcja_do_wykonania;  
}
```

Przykład:

```
#include <iostream>  
  
using namespace std;  
  
int main()  
{  
    for(int i=0;i<6;i++)  
        cout << "To jest mady tekst\n";  
}
```

```
#include <iostream>  
  
using namespace std;  
  
int main()  
{  
    for(int i=10;i>6;i--)  
        cout << "To jest mady tekst\n";  
}
```

Jeżeli powtarzany kod ma więcej niż jedną linię - obejmujemy go klamrami { } (nie objęcie kodu wieloliniowego spowoduje wykonanie jedynie pierwszej linii).

Warunek jest testowany przed wykonaniem instrukcji, zatem jeśli warunek nie zostanie spełniony, instrukcja może wcale się nie wykonać.

Ćwiczenie: wyświetl tabliczkę mnożenia 10x10

```
#include <iostream>  
  
using namespace std;  
  
int main()  
{  
    cout << "Tabliczka mnozenia dla liczb od 1 do 10" << endl <<endl << "      ";  
  
    for (int i=1; i<=10; i++)  
    {  
        cout<<i<<"  ";  
    }  
  
    cout<<"\n" << "  ";  
  
    for (int i=1; i<=10; i++)  
    {  
        cout<<"-----";  
    }  
  
    cout<<"\n\n";  
  
    for (int i=1; i<=10; i++)  
    {  
        if (i<10)  
        {  
            cout <<i<<" | ";  
        }  
    }  
}
```

```

    }
    else
    {
        cout <<i<<" | ";
    }

    for (int j=1; j<=10; j++)
    {
        if ((i*j)<10)
        {
            cout <<i*j<<" ";
        }
        else
        {
            cout <<i*j<<" ";
        }
    }
    cout << "\n";
}

cout << endl << "\a\n\n";
system ("pause");
return 0;
}

```

**Ćwiczenie:** zmodyfikuj powyższy program by wyświetlał tabliczkę mnożenia 15x15

Należy zmodyfikować pętle liczące iloczyn i dopisać formatowanie wyświetlania w przedziałach 1-9,10-99,100-999, np.:

```

    else if ((i*j)<100)
    {
        cout <<i*j<<" ";
    }
    else if ((i*j)<999)
    {
        cout <<i*j<<" "; // jedna spacja mniej niż w przypadku 10-99 (powyższym)
    }

```

```

#include <iostream> // autorzy: Słoma Dawid, Wardyński Stefan INF IV, gr A,10.2007
using namespace std;
int main()
{
    cout << "Tabliczka mnożenia dla liczb od 1 do 15" << endl <<endl << "      ";
    //-----
    for (int i=1; i<=15; i++) // nagłówek
    {
        if (i<10)
        {
            cout <<i<<" | ";
        }
        else
        {
            cout <<i<<" | ";
        }
    }
    cout<<"\n" << " ";
    //-----
    for (int i=1; i<=15; i++)
    {
        cout<<"-----";
    }
}

```

```

}
cout<<"\n\n";

for (int i=1; i<=15; i++)
{
    if (i<10) // lewa kolumna
    {
        cout <<i<<" | ";
    }
    else
    {
        cout <<i<<" | ";
    }
}
//-----
for (int j=1; j<=15; j++)
{
    if ((i*j)<10)
    {
        cout <<i*j<<" | ";
    }
    if ( (i*j)>=10 && (i*j)<100)
    {
        cout <<i*j<<" | ";
    }
    if ((i*j)>=100 && (i*j)<1000)
    {
        cout <<i*j<<"| ";
    }
}
cout << "\n";
}
cout << endl << "\a\n\n";
system ("pause");
return 0;
}

```

```

#include <iostream> // Sawicki Paweł, Kowolik Ryszard INF IV, gr A,10.2007
using namespace std;
int main()
{
    cout << "Tabliczka mnozenia dla liczb od 1 do 10" << endl <<endl << " ";
    for (int i=1; i<=15; i++)
    {
        if (i<10)
        {
            cout <<i<<" ";
        }
        else
        {
            cout <<i<<" ";
        }
    }
    cout<<"\n" << " ";
    for (int i=1; i<=15; i++)
    {
        cout<<"-----";
    }
    cout<<"\n\n";
    for (int i=1; i<=15; i++)

```

```
{
if (i<10)
{
cout <<i<<" | ";
}
else
{
cout <<i<<" | ";
}
for (int j=1; j<=15; j++)
{
if ((i*j)<10)
{
cout <<i*j<<" ";
}
if ((i*j)>=10 && (i*j) < 100)
{
cout <<i*j<<" ";
}
if ((i*j)>=100 && (i*j) < 1000)
{
cout <<i*j<<" ";
}
}
cout << "\n";
}
cout << endl << "\a\n\n";
system ("pause");
return 0;
}
```

## struktura powtórzenia – pętla warunkowa while

`while` możemy tłumaczyć jako: powtarzaj instrukcję tak długo jak warunek jest spełniony. Pętla ta może udawać pętlę `for`, ale jej działanie i zastosowanie jest znacznie szersze, ponieważ warunek może być warunkiem logicznym.

```
while (warunek)
{
// Instrukcje;
}
```

Przykład:

```
#include <iostream>

using namespace std;

int main()
{
int i=0;

while(i<7)
{
cout << "To jest mądry tekst" << endl;
i++;
}
return 0;
}
```

## struktura powtórzenia – pętla warunkowa do...while

Tłumaczyć możemy ją jako: wykonuj instrukcję, dopóki warunek jest spełniony. Struktura do/while jest podobna do struktury while, ale w strukturze while warunek kontynuacji pętli jest sprawdzany zanim jej ciało zostanie wykonane, natomiast w do/while sprawdza warunek kontynuacji pętli po tym, jak ciało pętli zostaje wykonane. Wynika z tego, że blok instrukcji zostanie wykonany przynajmniej jeden raz.

### Składnia instrukcji:

```
do {  
    // instrukcja;  
    // instrukcja;  
} while(warunek);
```

Przykład:

```
#include <iostream>  
using namespace std;  
  
main()  
{  
    int i; i=0;  
  
    do{  
        cout << "To jest mądry tekst " << endl;  
        i++;  
    }while(i<7);  
  
    system("pause");  
}
```

## instrukcja skoku (przerwania) - break

Do przerywania wykonywania pętli służy instrukcja **break**. Może być ona używana w dowolnej pętli i powoduje wykonanie skoku do pierwszej instrukcji za pętlą.

Instrukcja **break** kiedy jest wykonywana w strukturze while, for, do/while, lub switch powoduje bezpośrednie wyjście (i przerwanie wykonania) z tej struktury.

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    int a;  
  
    for(a=1;a<=20;a++)  
    {  
        if(a==10) { system("pause"); break; }  
  
        cout<<a<<" ";  
    }  
}
```

```
    return 0;
}
```

Gdy wartość zmiennej **a** osiągnie wartość 10 pętla **for** zostanie przerwana, nie zostanie nawet wyświetlona jej wartość.

## instrukcja skoku - continue

Jeśli zależy nam z kolei na skoku na początek bieżącej pętli, służy do tego celu słowo **continue**, ale pamiętając musimy, że **continue** ma zupełnie inne znaczenie dla pętli **while** i **do-while**, niż dla pętli **for**, bo w przypadku tych pierwszych powoduje normalny skok na początek pętli (dokładnie to skok do miejsca, w którym następuje sprawdzanie warunku), podczas gdy w **for** powoduje przejście do następnej iteracji.

Wyrażenie **continue**, kiedy jest wykonywana w strukturze **while**, **for**, **do/while**, lub **switch** pomija pozostałe wyrażenia w bloku tej struktury i kontynuuje wykonywanie następnej instrukcji.

```
#include <iostream>
using namespace std;

int main()
{
    int a;

    for(a=1;a<=20;a++)
    {
        if(a==10) {system("pause"); continue;}
        cout<<"a="<<a<<"\t";
    }

    system("pause");
    return 0;
}
```

Gdy wartość zmiennej **a** będzie się równała 10 dalsze wyrażenia w pętli **for** zostaną pominięte tzn. program wyświetli wszystkie wartości zmiennej **a** od 1 do 20 z pominięciem wyświetlenia wartości 10.

## instrukcja skoku - goto

Najbardziej ogólną instrukcją skoku jest **goto**, która wymaga jako argumentu etykiety, do której się skacze. Używanie tej instrukcji nie jest zalecane.

Instrukcja **goto** jest postrzegana przez programistów jako spadek po BASIC'u, ponieważ za jej pomocą możemy wykonywać skoki do deklarowanej etykiety. Większość domorosłych programistów uważa, że jej stosowanie jest przykładem złej znajomości rzeczy, ale w rzeczywistości użycie **goto** w uzasadnionym przypadku ma swój sens. Jest mało uzasadnionych przypadków użycia tej instrukcji.

Konstrukcja **goto**:

```
goto nazwa_etykiety;

// a w kodzie programu umieszczamy:
```

```
nazwa_etykiety:
```

```
// instrukcje_etykiety;
```

Ograniczenia instrukcji `goto` nie pozwalają jej na przeskoczenie definicji zmiennej, po każdej etykiecie musi wystąpić co najmniej jedna instrukcja. Instrukcja ta może przydać się choćby do natychmiastowego opuszczenia wielokrotnie zagnieżdżonej pętli, która trzeba by było przerywać w sposób **break 4**; (by opuścić cztery bloki instrukcji).

Ponadto **goto** nie może spowodować przejścia do wykonania instrukcji znajdującej się poza funkcją zawierającą `goto`. Instrukcja ta przeczy filozofii programowania, gdzie charakterystyczną cechą jest sekwencyjność: instrukcje są wykonywane w określonym porządku: od góry w dół i z lewej do prawej. Zastosowanie instrukcji skoku powoduje złamanie tej zasady. Istnieje niewiele sytuacji, w których zastosowanie tej instrukcji byłoby uzasadnione.

Demonstracja działania pętli z `break`, pętli z `continue` i pętli z `goto`:

```
#include <iostream>
using namespace std;
int main()
{
    int x;

    cout << "Start petli z break" << endl;
    for (x = 1; x < 5; x++)
    {
        if (x == 3) break;
        cout << " x = " << x << endl;
    }

    cout << endl << endl << endl;
    cout << "Start petli z continue" << endl;
    for (x = 1; x < 10; x++)
    {
        if (x == 3) continue;
        cout << " x = " << x << endl;
    }
    cout << endl << endl << endl;
    cout << "Start petli z goto" << endl;
    for (x = 1; x < 10; x++)
    {
        if (x == 2) goto etykietal;
        cout << " x = " << x << endl;
    }

    etykietal:
        cout << "oto instrukcje z etykiety 1" <<endl;

    cout << endl << endl << endl;
    system ("pause");
}
```

## Ćwiczenie

Napisz:

1. Program, który obliczy średnie zużycie paliwa, który będzie pobierał liczbę przejechanych kilometrów i zatankowanych litrów
2. Rozbuduj poprzedni program o obliczenie średniego zużycia na podstawie przykładowych danych (użyj do tego

poprzedniego programu), a następnie możliwość dowolnej ilości tankowań, o wartość początkową ilości paliwa w baku, oraz o wyświetlenie ile jeszcze paliwa pozostało w baku oraz ile na tym paliwie można przejechać kilometrów.

3. Napisz program wyświetlający kolejne potęgi liczby 2
4. Napisz program odczytujący promień koła i obliczający oraz wyświetlający średnicę, obwód oraz pole.
5. Napisz program odczytujący 3 niezerowe wartości zmiennoprzecinkowe(float) oraz określający i wyświetlający informację, czy mogą one stanowić długość boków trójkąta.
6. Napisz program który wyznaczy wszystkie trójki pitagorejskie których wartość jest mniejsza niż 500. Trójka pitagorejska to trzy odcinki o długości całkowitej, z których można zbudować trójkąt prostokątny.

```
#include <iostream> // program 1
using namespace std;
int main()
{
    float LiczbaKilometrow, LiczbaLitrow, SrednieZuzycie;
    cout << " Podaj liczbe przejechanych kilometrow : ";
    cin >> LiczbaKilometrow;
    cout << endl << endl;
    cout << " Podaj liczbe zuzytego paliwa : ";
    cin >> LiczbaLitrow;
    cout << endl << endl;
    SrednieZuzycie = LiczbaLitrow / ( LiczbaKilometrow / 100 );
    cout << "Srednie zyzycie paliwa wynosi " << SrednieZuzycie
        << " na 100 kilometrow" << endl << endl;
}
```

```
#include <iostream> // program 2
using namespace std;

int main()
{
    float LiczbaKilometrow, LiczbaLitrow, SrednieZuzycie, Bak, StanLicznika, Temp;
    cout << " Podaj liczbe przejechanych kilometrow : ";
    cin >> LiczbaKilometrow;
    cout << endl << endl;
    cout << " Podaj liczbe zuzytego paliwa : ";
    cin >> LiczbaLitrow;
    cout << endl << endl;
    SrednieZuzycie = LiczbaLitrow / ( LiczbaKilometrow / 100 );
    cout << "Srednie zyzycie paliwa wynosi " << SrednieZuzycie
        << " na 100 kilometrow" << endl << endl;
    system ("pause");
    cout << endl << endl;
    cout << " Podaj poczatkowy stan licznika: ";
    cin >> StanLicznika;

    cout << endl << endl;
    cout << " Podaj poczatkowa ilosc paliwa w baku: ";
    cin >> Bak;
    cout << endl << endl;
    cout << " Stan Licznika : " << StanLicznika << " kilometrow " << endl
        << " Stan Paliwa : " << Bak << " litrow " << endl
        << " Paliwa wystarczy na : " << (Bak / SrednieZuzycie) *100 << " kilometrow"
        << endl << endl;

    system ("pause");

    int Wybor;

    for (;;)
    {
        cout << endl << endl;;
        cout << " 0 - wyjscie " << endl
            << " 1 - tankowanie " << endl
    }
```

```

        << " 2 - zmiana stanu licznika " << endl
        << " 3 - przejechane kilometry " << endl
        << " 4 - korekta " << endl;
cin >> Wybor;

switch(Wybor)
{
    case 0: cout << "pa,pa" ; exit(0);

    case 1: cout << endl << endl;
            cout << " Podaj ilosc tankowanego paliwa : ";
            cin >> Temp;
            Bak = Bak + Temp;
            cout << endl << endl;
            cout << " Stan Licznika          : " << StanLicznika << " kilometrow " << endl
                 << " Stan Paliwa          : " << Bak << " litrow " << endl
                 << " Paliwa wystarczy na : " << (Bak / SrednieZuzycie) *100 << " kilometrow"
                 << endl << endl;
            break;

    case 2: cout << endl << endl;
            cout << " Podaj stan aktualny licznika : ";
            cin >> Temp;
            Temp = Temp - StanLicznika;
            StanLicznika = StanLicznika + Temp;
            Bak = Bak - (SrednieZuzycie * (Temp/100));
            cout << endl << endl;
            cout << " Przejechano          : " << Temp          << " kilometrow " << endl
                 << " Stan Licznika          : " << StanLicznika << " kilometrow " << endl
                 << " Stan Paliwa          : " << Bak << " litrow " << endl
                 << " Paliwa wystarczy na : " << (Bak / SrednieZuzycie) *100 << " kilometrow"
                 << endl << endl;
            break;

    case 3: cout << endl << endl;
            cout << " Podaj ilosc przejechanych kilometrow : ";
            cin >> Temp;
            StanLicznika = StanLicznika + Temp;
            Bak = Bak - (SrednieZuzycie * (Temp/100));
            cout << endl << endl;
            cout << " Przejechano          : " << Temp          << " kilometrow " << endl
                 << " Stan Licznika          : " << StanLicznika << " kilometrow " << endl
                 << " Stan Paliwa          : " << Bak << " litrow " << endl
                 << " Paliwa wystarczy na : " << (Bak / SrednieZuzycie) *100 << " kilometrow"
                 << endl << endl;
            break;

    case 4: cout << endl << endl;
            cout << " Podaj stan licznika          : "; cin >> StanLicznika;
            cout << " Podaj ilosc paliwa w baku : "; cin >> Bak;
            cout << " Podaj srednie zuzycie      : "; cin >> SrednieZuzycie;
            break;

    default: cout << endl << " -- nie ma takiej opcji -- ";break;
}
}
}

```

```

#include <iostream> // program 3
using namespace std;
int main()
{
    int x,n,wynik(1);
    int i;
    cout << endl << " Podaj x : "; cin >> x;
    cout << endl << " Podaj n : "; cin >> n;
    cout << endl << endl;
    cout << x << " " << n << endl;
    for(int i=0;i<n;i++)

```

```
{
    wynik *= x;
}
cout << endl << " wynik = " << wynik
    << endl << endl;

system ("pause");
}
```

```
#include <iostream> // program 4

using namespace std;

int main()
{
    float r,pole,obwod;

    cout << " Podaj dlugosc promienia kola : "; cin >> r; cout << endl << endl;
    pole = r*r*3.14;
    obwod = 2*3.14*r;

    cout <<"pole wynosi " << pole
        <<" obwod wynosi " << obwod <<endl << endl;

    system ("pause");
}
```

```
#include <iostream> // program 5.1

using namespace std;

int main()
{
    float a,b,c;

    bool test(true);

    cout << " Podaj dlugosc boku a : "; cin >> a;
    cout << " Podaj dlugosc boku b : "; cin >> b;
    cout << " Podaj dlugosc boku c : "; cin >> c;

    if (a+b<c) test = false;
    if (a+c<b) test = false;
    if (b+c<a) test = false;

    if (test)
    {
        cout <<"Z podanych odcinkow mozna zbudowac trojkat" <<endl<<endl;
    }
    else
    {
        cout<< "nie mozna zbudowac trojkata"<<endl<<endl;
    }

    system ("pause");
}
```

dopisz sprawdzenie, czy trójkąt jest prostokątny, oparte (podobnie jak wyżej) na zmiennej typu bool

```
#include <iostream> // program 6
```

```
using namespace std;

int main()
{
    int a,b,c;

    const int ile (500);          // określenie górnego zakresu poszukiwań

    for(int a=1;a<ile;a++)

    {

        for(int b=a;b<ile;b++)

        {

            for(int c=a;c<ile;c++)

            {

                if (a*a + b*b == c*c)

                {

                    cout << a <<" "<< b << " "<< c << endl;

                }

            }

        }

    }

    system ("pause");
}
#include <iostream>                // program 5.2

using namespace std;

int main()
{
    float a,b,c;

    bool test(true);

// pobranie danych
    cout << " Podaj dlugosc boku a : "; cin >> a;
    cout << " Podaj dlugosc boku b : "; cin >> b;
    cout << " Podaj dlugosc boku c : "; cin >> c;

// test - czy odcinki spełniają warunki trójkąta
    if (a+b<c) test = false;
    if (a+c<b) test = false;
    if (b+c<a) test = false;

// wyświetlenie komunikatu, czy z odcinków można zbudować trójkąt
    if (test)
    {
        cout <<"Z podanych odcinkow mozna zbudowac trojkat" <<endl<<endl;

//-----
// reset stanu test - ustawiamy, że domyślnie trójkąt nie jest prostokątny
        test = false;

// sprawdzenie, czy trójkąt jest prostokątny
```

```
if ((a>b) && (a>c))
{
    if (c*c + b*b == a*a) {test= true;}
}

if ((b>a) && (b>c))
{
    if (c*c + a*a == b*b) {test= true;}
}

if ((c>a) && (c>b))
{
    if (b*b + a*a == c*c) {test= true;}
}

// wyświetlenie komunikatu, czy trójkąt jest prostokątny
if (test)
{
    cout <<"trójkąt jest prostokątny" <<endl<<endl;
}
else
{
    cout<< "trójkąt nie jest prostokątny"<<endl<<endl;
}

//-----
}
else
{
    cout<< "nie można zbudować trójkąta"<<endl<<endl;
}

system ("pause");
}
```

## Rozdział VI, Funkcje

Funkcja to część kodu, którą możemy wywołać za pomocą jej nazwy (identyfikatora). W odróżnieniu od Pascala, nie mamy do dyspozycji procedur (i dobrze). Funkcja jest podprogramem, wyłączonym z funkcji głównej w celu poprawienia czytelności kodu, oraz (opcjonalnego) wielokrotnego jej użycia. Na szczególną uwagę zasługują dwa zagadnienia: przekazywanie argumentów do funkcji i ich zwracanie, oraz przeciążanie funkcji.

Funkcję deklarujemy w następujący sposób:

```
<typ_zwracanej_wartosci> nazwa_funkcji ( <argumenty_funkcji> );  
{  
    // ciało funkcji  
}
```

Typem zwracanej wartości jest jeden z typów podstawowych jak char czy int lub typy złożone (struktury i klasy), następnie podajemy nazwę funkcji a po niej w nawiasach po przecinku wskazujemy jakie argumenty funkcja będzie przyjmować. Funkcje, które nie zwracają żadnej wartości deklarujemy jako void:

```
#include <iostream>  
  
using namespace std;  
  
void napis()  
{  
    cout << "Funkcja działa poprawnie. Albo nie. Sam nie wiem." << endl;  
  
    return;  
}  
  
int dodaj(int a, int b)  
{  
    return a+b;  
}  
  
int main()  
{  
    napis(); // wywołanie funkcji napis()  
  
    int wynik = dodaj(5,7); // wywołanie funkcji dodaj()  
  
    cout << "5 + 7 = " << wynik << endl;  
  
    system("pause");  
  
    return 0;  
}
```

Wartość funkcji zwracamy przez słowo return po którym podajemy zwracaną do funkcji nadrzędnej wartość. Zwrócenie wartości jest jednocześnie zakończeniem działania funkcji.

### Ćwiczenie

Dopisz do powyższego programu trzy funkcje: odejmij, pomnóż i podziel, oraz pobranie liczb a,b od użytkownika

(prosty kalkulator). Sprawdź, czy przy dzieleniu liczba b jest różna od zera. Wyświetl wszystkie wyniki, oraz zbuduj ładny (user-friendly) interfejs programu.

```
#include <iostream>
using namespace std;
// ----- funkcje -----
void pomoc()
{
    cout << "Program wykonuje cztery podstawowe dzialania na liczbach." << endl
         << "Podaj dwie liczby, a program obliczy ich sume, roznice," << endl
         << " iloczyn i iloraz. Pamietaj, ze nie wolno dzielic przez zero"
         << endl << endl;
    return;
}
float suma(float a, float b)
{
    return a+b;
}
float roznica(float a, float b)
{
    return a-b;
}
float iloczyn(float a, float b)
{
    return a*b;
}
float iloraz(float a, float b)
{
    if (b!=0) {return a/b;}
    return 0;
}
// ----- funkcja glowna -----

int main()
{
    pomoc(); // wywołanie funkcji napis()

    cout << endl << "Podaj a = "; float a; cin >> a; // pobranie danych
    cout << endl << "Podaj b = "; float b; cin >> b;

    float wynik;

    wynik=suma(a,b); // uzycie zmiennej
    cout << endl << a << " + " << b << " = " << wynik; // do wyświetlenia wyniku

    cout << endl << a << " - " << b << " = " << roznica(a,b); // wyświetlenie bezpośrednio
    // po wykonaniu obliczeń

    wynik=iloczyn(a,b); // uzycie zmiennej
    cout << endl << a << " * " << b << " = " << wynik; // do wyświetlenia wyniku

    // gdy wynik dzielenia jest równy 0, oznacza, że próbowano dzielić przez 0

    wynik=iloraz(a,b); // uzycie zmiennej

    if (wynik ==0)
    {
        cout << endl << "UWAGA - dzielenie przez 0 jest niedozwolone";
    }
}
```

```
    }
    else
    {
        cout<<endl<<a<<" / "<<b<<" = "<<wynik;
    }

    cout <<endl<<endl<<endl;
    system("pause");
    return 0;
}
```

## Przeciążanie funkcji

Przeciążanie nazwy funkcji polega na zadeklarowaniu kilku funkcji z identyczną nazwą, różniące się parametrami. W zależności od ilości parametrów (i ich rodzaju) w wywołaniu, zostanie użyta odpowiednia funkcja. Przeciążanie funkcji pozwala na użycie tego samego działania w odniesieniu do różnego typu danych.

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

double dodaj(double a, double b)
{
    return a+b;
}

int main()
{
    dodaj(5,4);    // dodaj(int,int)
    dodaj(5.0,4); // dodaj(double,double)
    return 0;
}
```

## Funkcje inline

Jeżeli definicja (nie deklaracja) funkcji jest poprzedzona słowem inline oraz funkcja nie jest rekurencyjna i nie zawiera żadnych pętli to w miejscu wywołania tej funkcji kompilator wstawi jej kod, co spowoduje zwiększenie rozmiaru pliku wykonywalnego ale może spowodować także przyspieszenie jego działania.

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

int main()
{
    int a,b,c;
    a = 5;
    b = 9;
    c = dodaj(a,b);
    return 0;
}
```

co w procesie kompilacji jest równoważne zapisowi:

```
int main()
{
    int a,b,c;
    a = 5;
    b = 9;
    c = a+b;
    return 0;
}
```

## Rekurencja

Rekurencja polega na odwoływaniu się funkcji do samej siebie. Funkcja w swoim kodzie posiada wywołanie samej siebie, zatem tworzone są kolejne „zagnieżdżenia” funkcji w funkcji, co powoduje niebezpieczeństwo kontroli wywołań funkcji – tak by taka rekurencja miała koniec -, by funkcja nie pracowała w nieskończoność.

Dobrym przykładem ilustrowania rekurencji jest obliczanie silni:

```
#include <iostream>

using namespace std;

int silnia(int n)
{
    switch (n)
    {
        case 1: return 1; break;
        default: return n * silnia(n-1);
    }
}

int main()
{
    int wej;
    cout << "Silnie jakiej liczby obliczyc?\n";
    cin >> wej;
    cout << silnia(wej);
}
```

Czasami rekurencja jest mało wydajna i można ją zastąpić zwykłą pętlą:

```
int silnia(int n)
{
    int x=1;
    while(a > 1) x *= a--;
    return x;
}
```

## Wartości domyślne funkcji

Język C++ pozwala nam na nadanie wartości domyślnych argumentom funkcji w czasie jej definicji. Wartości domyślne nadajemy argumentom idąc od końca listy argumentów tzn.:

```
int dodaj(int a = 5, int b)           //nieprawidłowo - nie nadano wartości domyślnej
{ return a+b; }                       //arg. b więc nie można nadać go arg. a
```

```
double dodaj(double a=3.1415, double b=2.7181) //prawidłowo
{ return a+b; }

float dodaj(float a, float b=0.5772) //prawidłowo
{ return a+b; }

int main()
{
    dodaj(); //dodaj(3.1415, 2.7181)
    dodaj(1); //błąd - niejednoznaczne
    dodaj(1.0); //dodaj(1,2.7181)
    dodaj(float(1)); //dodaj(1,0.5772)
    return 0;
}
```

Odwołując się do funkcji z parametrami domyślnymi możemy zatem pomijać te parametry, które posiadają wartości domyślne.

## Prototypy funkcji

Język C++ pozwala nam na zadeklarowaniu funkcji (podanie jej prototypu), a umieszczenie właściwego kodu funkcji w innym miejscu kodu (na przykład na końcu).

```
#include <iostream>

using namespace std;

void napis(); // prototyp funkcji napis();
int dodaj(int a, int b); // prototyp funkcji dodaj();
// właściwy kod tych funkcji znajduje się po funkcji głównej

int main()
{
    napis(); // wywołanie funkcji napis()

    int wynik = dodaj(5,7); // wywołanie funkcji dodaj()

    cout << "5 + 7 = " << wynik << endl;

    system("pause");

    return 0;
}

void napis()
{
    cout << "Funkcja działa poprawnie. Albo nie. Sam nie wiem." << endl;

    return;
}

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

```

#include <iostream>
#include <iostream>
using namespace std;
//-----
int nww(int x, int y);          // prototyp funkcji NWW
int nwd(int x, int y);        // prototyp funkcji NWD
//-----
int a,b,c,d,e,f,x,y,z,temp;  // zmienne globalne
//-----
int main()
{
    system("cls");

    cout << endl << endl
         << " Program demonstruje cztery podstawowe dzialania"<<endl
         << " wykonywane na ulamkach zwyklych           "<<endl
         << endl << endl << endl << endl << endl;

    system("pause");

    int Wybor;

    for (;;)
    {
        system ("cls");
        cout << endl << endl << endl << endl;
        cout << " 0 - wyjscie           " << endl
             << " 1 - dodawanie         " << endl
             << " 2 - odejmowanie        " << endl
             << " 3 - mnozenie           " << endl
             << " 4 - dzielenie          " << endl << endl;
        cout << " Twój wybór to : "; cin >> Wybor;

        switch(Wybor)
        {
            case 0:
                cout << endl << endl << endl
                     << "bye!!" << endl << endl;
                system ("pause");
                exit(0);
            break;

            case 1:
                // ----- pobranie danych -----
                system("cls"); cout <<endl;
                cout << "podaj kolejno: czesc calkowita, licznik, mianownik ";
                cout <<endl<<endl<<"podaj trzy liczby dla pierwszego ulamka :";
                cin>>x; cin>>a; cin >>b;
                cout <<endl<<endl<<"podaj trzy liczby dla drugiego ulamka :";
                cin>>y; cin>>c; cin >>d;

                cout <<endl<< "podałeś ulamki : " <<endl<<endl;
                cout <<"    " << a << "          " << c << endl
                     << x << " --- + " << y << " --- = " << endl
                     <<"    " << b << "          " << d <<endl<<endl;

                if (a==0||b==0||c==0||d==0)
                {cout<<"nie wolno wpisywać zera"<<endl;system ("pause");break;}

                system ("pause"); cout<<endl;

```

```

if(x!=0) // zamiana na ułamki niewłaściwe
{
    temp=abs(x)*abs(b)+abs(a);
    if (a*b*x<0) { a = -temp;}
    else { a = temp;}
}
if(y!=0) // zamiana na ułamki niewłaściwe
{
    temp=abs(y)*abs(d)+abs(c);
    if (y*c*d<0) {c = -temp;}
    else {c=temp;}
}
b=abs(b);
d=abs(d);

cout << "po zamianie na ułamki niewłaściwe :" <<endl<<endl;
cout <<" " << a << " " << c << endl
    << " --- + ---- = " << endl
    <<" " << b << " " << d <<endl<<endl;
system ("pause");cout<<endl;

temp=nwd(a,b); // skrócenie ułamków
a/=temp; b/=temp; // skrócenie ułamków
temp=nwd(c,d); // skrócenie ułamków
c/=temp; d/=temp; // skrócenie ułamków

cout << "po skroceniu ułamekow :" <<endl<<endl;
cout <<" " << a << " " << c << endl
    << " --- + ---- = " << endl
    <<" " << b << " " << d <<endl<<endl;
system ("pause");cout<<endl;

temp=nww(b,d); // zamiana na wspólny mianownik
a=a*(temp/b); b=temp; // zamiana na wspólny mianownik
c=c*(temp/d); d=temp; // zamiana na wspólny mianownik

cout << "sprowadzenie do wspólnego mianownika :" <<endl<<endl;
cout <<" " << a << " " << c << endl
    << " --- + ---- = " << endl
    <<" " << b << " " << d <<endl<<endl;
system ("pause");cout<<endl;

e=a+c; // wykonanie dodawania
f=temp;

cout << "po dodaniu mamy :" <<endl<<endl;
cout <<" " << e << endl
    << " ---- = " << endl
    <<" " << f << endl<<endl;
system ("pause");cout<<endl;

temp=nwd(e,f); // skrócenie wyniku
e/=temp;
f/=temp;

cout << "po skroceniu mamy :" <<endl<<endl;
cout <<" " << e << endl
    << " ---- = " << endl
    <<" " << f << endl<<endl;

```

```
system ("pause");cout<<endl;

z=0;
if (abs(e)>=abs(f))          // część całkowita
{
    if (e>0&&f>0)
    {
        while (e>=f)
        {
            e-=f;
            z++;
        }
    }
    else if (e<0&&f>0)
    {
        e=abs(e);
        while (e>=f)
        {
            e-=f;
            z++;
        }
        e=-e;
    }
    else if (e>0&&f<0)
    {
        f=abs(f);
        while (e>=f)
        {
            e-=f;
            z++;
        }
        f=-f;
    }
    else if (e<0&&f<0)
    {
        e=abs(e);
        f=abs(f);

        while (e>=f)
        {
            e-=f;
            z++;
        }
        e=-e;
        f=-f;
    }
}

if (e!=0)
{
    cout << "po wyciagnieciu cz. calkowitej mamy wynik :" <<endl<<endl;
    cout <<"    " << e << endl
        << z << " ---- = " << endl
        <<"    " << f << endl<<endl;
    system ("pause"); cout<<endl;
}
else
{
    cout << "po wyciagnieciu cz. calkowitej mamy wynik :" <<endl<<endl;
    cout << endl << z << endl << endl ;
    system ("pause"); cout<<endl;
}
}
```

```

break;

case 2:
// ----- pobranie danych -----
system("cls"); cout <<endl;
cout << "podaj kolejno: czesc calkowita, licznik, mianownik ";
cout <<endl<<endl<<"podaj trzy liczby dla pierwszego ulamka :";
cin>>x; cin>>a; cin >>b;
cout <<endl<<endl<<"podaj trzy liczby dla drugiego ulamka :";
cin>>y; cin>>c; cin >>d;

cout <<endl<< "podales ulamki :" <<endl<<endl;
cout <<" " << a << " " << c << endl
    << x << " --- - " << y << " --- = " << endl
    <<" " << b << " " << d <<endl<<endl;

if (a==0||b==0||c==0||d==0)
{cout<<"nie wolno wpisywac zera"<<endl;system ("pause");break;}

system ("pause"); cout<<endl;

if(x!=0) // zamiana na ulamki niewlasciwe
{
    temp=abs(x)*abs(b)+abs(a);
    if (a*b*x<0) { a = -temp;}
    else { a = temp;}
}
if(y!=0) // zamiana na ulamki niewlasciwe
{
    temp=abs(y)*abs(d)+abs(c);
    if (y*c*d<0) {c = -temp;}
    else {c=temp;}
}
b=abs(b);
d=abs(d);

cout << "po zamianie na ulamki niewlasciwe :" <<endl<<endl;
cout <<" " << a << " " << c << endl
    << " --- - ---- = " << endl
    <<" " << b << " " << d <<endl<<endl;
system ("pause");cout<<endl;

temp=nwd(a,b); // skrócenie ulamków
a/=temp; b/=temp; // skrócenie ulamków
temp=nwd(c,d); // skrócenie ulamków
c/=temp; d/=temp; // skrócenie ulamków

cout << "po skroceniu ulamkow :" <<endl<<endl;
cout <<" " << a << " " << c << endl
    << " --- - ---- = " << endl
    <<" " << b << " " << d <<endl<<endl;
system ("pause");cout<<endl;

temp=nww(b,d); // zamiana na wspolny mianownik
a=a*(temp/b); b=temp; // zamiana na wspolny mianownik
c=c*(temp/d); d=temp; // zamiana na wspolny mianownik

cout << "sprowadzenie do wspolnego mianownika :" <<endl<<endl;
cout <<" " << a << " " << c << endl

```

```

        << " --- - ---- = " << endl
        <<" " << b << "          " << d <<endl<<endl;
system ("pause");cout<<endl;

e=a-c;          // wykonanie odejmowania
f=temp;

cout << "po odejmowaniu mamy :" <<endl<<endl;
cout <<" " << e << endl
    << " ---- = " << endl
    <<" " << f << endl<<endl;
system ("pause");cout<<endl;

temp=nwd(e,f);    // skrócenie wyniku
e/=temp;
f/=temp;

cout << "po skroceniu mamy :" <<endl<<endl;
cout <<" " << e << endl
    << " ---- = " << endl
    <<" " << f << endl<<endl;
system ("pause");cout<<endl;

z=0;
if (abs(e)>=abs(f))          // część całkowita
{
    if (e>0&&f>0)
    {
        while (e>=f)
        {
            e-=f;
            z++;
        }
    }
    else if (e<0&&f>0)
    { e=abs(e);
      while (e>=f)
      {
          e-=f;
          z++;
      }
      e=-e;
    }
    else if (e>0&&f<0)
    { f=abs(f);
      while (e>=f)
      {
          e-=f;
          z++;
      }
      f=-f;
    }
    else if (e<0&&f<0)
    { e=abs(e);
      f=abs(f);

      while (e>=f)
      {
          e-=f;
          z++;
      }
    }
}

```

```

        e=-e;
        f=-f;
    }

}

if (e!=0)
{
    cout << "po wyciagnieciu cz. calkowitej mamy wynik :" <<endl<<endl;
    cout <<"    " << e << endl
        << z << " ---- = " << endl
        <<"    " << f << endl<<endl;
    system ("pause"); cout<<endl;
}
else
{
    cout << "po wyciagnieciu cz. calkowitej mamy wynik :" <<endl<<endl;
    cout << endl << z << endl << endl ;
    system ("pause"); cout<<endl;
}

}

break;

case 3:
// ----- pobranie danych -----
system("cls"); cout <<endl;
cout << "podaj kolejno: czesc calkowita, licznik, mianownik ";
cout <<endl<<endl<<"podaj trzy liczby dla pierwszego ulamka :";
cin>>x; cin>>a; cin >>b;
cout <<endl<<endl<<"podaj trzy liczby dla drugiego ulamka :";
cin>>y; cin>>c; cin >>d;

cout <<endl<< "podales ulamki :" <<endl<<endl;
cout <<"    " << a << "    " << c << endl
    << x << " --- * " << y << " --- = " << endl
    <<"    " << b << "    " << d <<endl<<endl;

if (a==0||b==0||c==0||d==0)
{cout<<"nie wolno wpisywac zera"<<endl;system ("pause");break;}

system ("pause"); cout<<endl;

// ----- MNOZENIE -----

if(x!=0) // zamiana na ulamki niewlasciwe
{
    temp=abs(x)*abs(b)+abs(a);
    if (a*b*x<0) { a = -temp;}
    else { a = temp;}
}
if(y!=0) // zamiana na ulamki niewlasciwe
{
    temp=abs(y)*abs(d)+abs(c);
    if (y*c*d<0) {c = -temp;}
    else {c=temp;}
}
b=abs(b);
d=abs(d);

```

```

cout << "po zamianie na ulamki niewlasciwe :" <<endl<<endl;
cout <<" " << a << " " << c << endl
    << " --- * ---- = " << endl
    <<" " << b << " " << d <<endl<<endl;
system ("pause");cout<<endl;

temp=nwd(a,b); // skrócenie ułamków
a/=temp; b/=temp; // skrócenie ułamków
temp=nwd(c,d); // skrócenie ułamków
c/=temp; d/=temp; // skrócenie ułamków

cout << "po skroceniu ulamkow :" <<endl<<endl;
cout <<" " << a << " " << c << endl
    << " --- * ---- = " << endl
    <<" " << b << " " << d <<endl<<endl;
system ("pause");cout<<endl;

e=a*c; f=b*d; // wykonanie mnożenia

cout << "po wykonaniu mnozenia mamy :" <<endl<<endl;
cout <<" " << e << endl
    << " ---- = " << endl
    <<" " << f << endl<<endl;
system ("pause");cout<<endl;

temp=nwd(e,f); // skrócenie wyniku mnożenia
e/=temp;
f/=temp;

cout << "po skroceniu mamy :" <<endl<<endl;
cout <<" " << e << endl
    << " ---- = " << endl
    <<" " << f << endl<<endl;
system ("pause");cout<<endl;

z=0;
if (abs(e)>=abs(f)) // część całkowita
{
    if (e>0&&f>0)
    {
        while (e>=f)
        {
            e-=f;
            z++;
        }
    }
    else if (e<0&&f>0)
    {
        e=abs(e);
        while (e>=f)
        {
            e-=f;
            z++;
        }
        e=-e;
    }
    else if (e>0&&f<0)
    {
        f=abs(f);
        while (e>=f)
        {
            e-=f;
            z++;
        }
    }
}

```

```

        }
        f=-f;
    }
    else if (e<0&&f<0)
    {
        e=abs(e);
        f=abs(f);

        while (e>=f)
        {
            e-=f;
            z++;
        }
        e=-e;
        f=-f;
    }
}

if (e!=0)
{
    cout << "po wyciagnieciu cz. calkowitej mamy wynik :" <<endl<<endl;
    cout <<"    " << e << endl
        << z << " ---- = " << endl
        <<"    " << f << endl<<endl;
    system ("pause"); cout<<endl;
}
else
{
    cout << "po wyciagnieciu cz. calkowitej mamy wynik :" <<endl<<endl;
    cout << endl << z << endl << endl ;
    system ("pause"); cout<<endl;
}
break;

case 4:
    // ----- pobranie danych -----
    system("cls"); cout <<endl;
    cout << "podaj kolejno: czesc calkowita, licznik, mianownik ";
    cout <<endl<<endl<<"podaj trzy liczby dla pierwszego ulamka :";
    cin>>x; cin>>a; cin >>b;
    cout <<endl<<endl<<"podaj trzy liczby dla drugiego ulamka :";
    cin>>y; cin>>c; cin >>d;

    cout <<endl<< "podales ulamki :" <<endl<<endl;
    cout <<"    " << a << "    " << c << endl
        << x << " --- : " << y << " --- = " << endl
        <<"    " << b << "    " << d <<endl<<endl;

    if (a==0||b==0||c==0||d==0)
    {cout<<"nie wolno wpisywac zera"<<endl;system ("pause");break;}

    system ("pause"); cout<<endl;

    // ----- DZIELENIE -----

    if(x!=0) // zamiana na ulamki niewlasciwe
    {
        temp=abs(x)*abs(b)+abs(a);
        if (a*b*x<0) { a = -temp;}
        else { a = temp;}
    }

```

```

}
if(y!=0) // zamiana na ułamki niewłaściwe
{
    temp=abs(y)*abs(d)+abs(c);
    if (y*c*d<0) {c = -temp;}
    else {c=temp;}
}
b=abs(b);
d=abs(d);

cout << "po zamianie na ułamki niewłaściwe :" <<endl<<endl;
cout <<" " << a << " " << c << endl
    << " --- : ---- = " << endl
    <<" " << b << " " << d <<endl<<endl;
system ("pause");cout<<endl;

temp=nwd(a,b); // skrócenie ułamków
a/=temp; b/=temp; // skrócenie ułamków
temp=nwd(c,d); // skrócenie ułamków
c/=temp; d/=temp; // skrócenie ułamków

cout << "po skroceniu ułamkow :" <<endl<<endl;
cout <<" " << a << " " << c << endl
    << " --- : ---- = " << endl
    <<" " << b << " " << d <<endl<<endl;
system ("pause");cout<<endl;

temp=c;c=d;d=temp; // odwrócenie drugiego ułamka

cout << "po zamianie dzielenia na mnozenie :" <<endl<<endl;
cout <<" " << a << " " << c << endl
    << " --- * ---- = " << endl
    <<" " << b << " " << d <<endl<<endl;
system ("pause");cout<<endl;

e=a*c; f=b*d; // wykonanie mnożenia

cout << "po wykonaniu mnozenia mamy :" <<endl<<endl;
cout <<" " << e << endl
    << " ---- = " << endl
    <<" " << f << endl<<endl;
system ("pause");cout<<endl;

temp=nwd(e,f); // skrócenie wyniku mnożenia
e/=temp;
f/=temp;

cout << "po skroceniu mamy :" <<endl<<endl;
cout <<" " << e << endl
    << " ---- = " << endl
    <<" " << f << endl<<endl;
system ("pause");cout<<endl;

z=0;
if (abs(e)>=abs(f)) // część całkowita
{
    if (e>0&&f>0)
    {
        while (e>=f)
        {
            e-=f;
        }
    }
}

```



```
//----- FUNKCJE -----  
//-----  
int nww(int x, int y)          // funkcja oblicza najmniejszą wspólną  
{                             // wielokrotność dwóch liczb  
    int tmp;  
    int xy = x*y;  
  
    while (y) {  
        tmp = x*y;  
        x = y;  
        y = tmp;  
    }  
  
    return (xy/x);  
}  
//-----  
int nwd(int x, int y)          // funkcja oblicza największy wspólny dzielnik  
{                             // dwóch liczb  
    while(y!=0)  
    {  
        int z=y;  
        y=x%y;  
        x=z;  
    }  
    return x;  
}  
//-----
```

## Rozdział VII, Przestrzeń nazw

### przestrzeń std

Wszystkie standardowe elementy języka są umieszczone w przestrzeni o nazwie **std**, którą deklarujemy poprzez:

```
using namespace std;
```

po czym możemy użyć poleceń zawartych w tej przestrzeni:

```
#include <iostream>

using namespace std;           // deklaracja standardowej przestrzeni nazw

int main()
{
    cout << "Hello World!" << endl;    // użycie cout z przestrzeni std

    return 0;
}
```

Możemy przestrzeń nazw określać każdorazowo przed wywołaniem polecenia zawartego w tej przestrzeni, np.:

```
#include <iostream>

int main( )
{
    std::cout << "Hello World!" << std::endl;    // wywołanie polecenia

    return 0;
}
```

### własna przestrzeń nazw

W różnych przestrzeniach możemy tworzyć funkcje o tych samych nazwach (w celu grupowania tych samych poleceń dla różnych obiektów). Możemy także tworzyć własne przestrzenie nazw:

```
#include <iostream>

namespace MojaPrzestrzen
{
    void hello( )
    {
        std::cout << "Hello World!" << std::endl;
    }
}

int main( )
{
    MojaPrzestrzen::hello( );

    return 0;
}
```

```
}
```

Ponadto, tworząc przestrzeń nazw możemy najpierw zadeklarować jej funkcje, a kod tych funkcji podać później:

```
#include <iostream>

namespace Matematyka
{
    int dodaj ( int a, int b );           // zapowiedz funkcji przestrzeni
    int odejmij( int a, int b );       // 'prototypy' funkcji
}

using namespace std;

int main( )
{
    cout << Matematyka::dodaj( 10, 20 ) << endl;

    return 0;
}

namespace Matematyka                    // tresc funkcji
{
    int dodaj( int a, int b )
    {
        return a+b;
    }

    int odejmij( int a, int b )
    {
        return a-b;
    }
}
```

Możemy także zapisać treść funkcji z przestrzeni nazw w ten sposób:

```
int Matematyka::dodaj( int a, int b )
{
    return a+b;
}
```



## Rozdział VIII, Preprocesor,

### preprocesor

Preprocesor to programem uruchamiany przed kompilatorem, odpowiedzialny za przetwarzanie dyrektyw (poleceń dla preprocesora):

**#include <nazwa> lub #include "nazwa"**

Dyrektywa dołączająca podany plik nagłówkowy do kodu programu. Pliki nagłówkowe pochodzące z C mają postać *nazwa.h*, natomiast biblioteki C++ określone są jedynie nazwą i poprzedzone są zwykle literką *c* (np.: *cstdio*). Podanie biblioteki poprzez <nazwa> określa dołączenie pliku nagłówkowego znajdującego się w miejscu określonym przez kompilator (zwykle bibliotek standardowych), natomiast `#include "nazwa.h"` oznacza dołączenie biblioteki znajdującej się w bieżącym katalogu lub o określonej ścieżce dostępu (np.: własnej biblioteki, musimy obowiązkowo także dodać rozszerzenie pliku - niekoniecznie *\*.h*).

**#define nazwa\_zmiennej wartość\_zmiennej**

Dyrektywa służąca do tworzenia makrodefinicji, pozwalająca na zastąpienie dowolnego ciągu znaków pewnym identyfikatorem, może służyć także do definiowania stałych, np.: `#define liczba_pi 3.14`

**#ifdef nazwa\_zmiennej**

Dyrektywa warunkowa, sprawdzająca czy jest zdefiniowana zmienna o określonej nazwie. Jeżeli tak, to wykonuje kod aż do wystąpienia dyrektywy `#endif`.

**#ifndef**

Dyrektywa warunkowa, z tym wyjątkiem, że wykonuje część kodu, w momencie nie spełnienia warunku.

**#endif**

Dyrektywa ta kończy działanie obu powyższych dyrektyw warunkowych.

**#else**

Dyrektywa występująca w parze z dyrektywami warunkowymi. Wykonuje ona jakiś kod w momencie niespełnienia jakiegoś warunku.

Ponadto występują jeszcze dyrektywy: `#error`, `#pragma`, `#line`, lecz nie będziemy ich na razie omawiać.