PODSTAWY PROGRAMOWANIA
Elementy języka C++

Poniżej podano podstawowe pojęcia dotyczące procesu tworzenia programu.

Pierwszym etapem tworzenia programu **edycja programu.** Jest to zapis algorytmu przy pomocy poleceń określonego języka programowania. Program zapisywany jest w pliku tekstowym z odpowiednim rozszerzeniem. Można tu używać dowolnego edytora (np. notatnik). Jednak do edycji i uruchamiania programu najczęściej wykorzystuje się określone środowiska programistyczne z wbudowanymi edytorami (np. Matlab, RAD Studio czy Dev). Wbudowane specjalistyczne edytory, ułatwiają pisanie np. zaznaczając kolorami słowa kluczowe języka czy dokonując tabulacji tekstu (odpowiednich przesunięć ułatwiających interpretację treści). Po utworzeniu programu następuje jego **kompilacja, linkowanie i debagowanie**.

Kompilator

To (ang. compiler) program służący do tłumaczenia kodu napisanego w jednym języku (np. C++,C, Pascal,) na równoważny kod w języku maszynowym (zero-jedynkowym).

Interpreter

To program komputerowy, który analizuje kolejne polecenia kodu źródłowego programu i na bieżąco je wykonuje (np. Matlab). Jest to inaczej niż w procesie kompilacji, podczas której nie wykonuje się wejściowego programu (kodu źródłowego), lecz tłumaczy go do wykonywalnego kodu maszynowego.

Podsumowując: kompilacja to tłumaczenie kodu źródłowego na kod wynikowy, a interpretacja to tłumaczenie połączone z natychmiastowym wykonaniem programu. Zarówno kompilator jak interpretator są też określane mianem ***translatora.***

Konsolidator

(ang. ***linker***) lub program, który łączy zadane pliki obiektowe i biblioteki tworząc w ten sposób plik wykonywalny programu.

Debugger

program komputerowy służący do dynamicznej analizy innych programów, w celu odnalezienia i identyfikacji zawartych w nich błędów, zwanych z angielskiego bugami (robakami). Proces nadzorowania wykonania programu za pomocą debuggera określa się mianem ***debugowania.***

IDE

(ang***. Integrated Development Environment*** - zintegrowane środowisko programistyczne) to aplikacja lub zespół aplikacji (środowisko) służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania.

RAD

(ang. ***Rapid Application Development*** - szybkie tworzenie aplikacji) to ideologia i technologia polegająca na udostępnieniu programiście dużych możliwości prototypowania oraz dużego zestawu gotowych komponentów.

VCL (ang. ***Visual Component Library***) – biblioteka komponentów graficznych

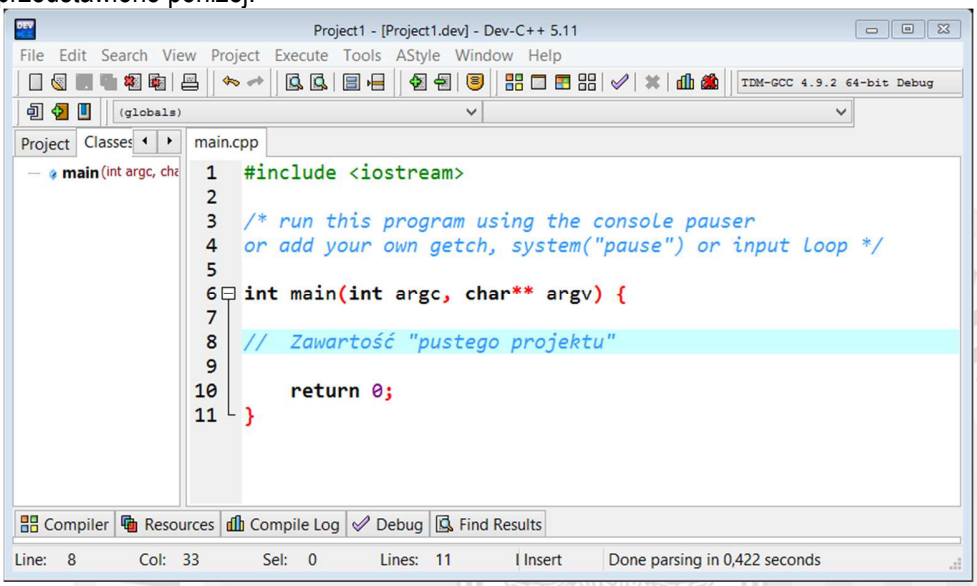
Przykładowe środowiska programistyczne

DEV-C++5.11

Aby utworzyć nowy projektu („pusty”) należy wybrać opcje:

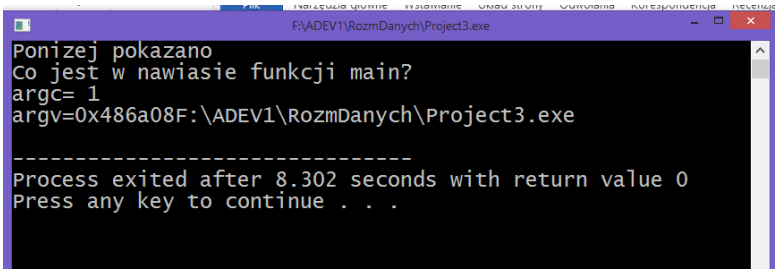
File→ New→ Project→ Console Aplication (z opcją C++)

Następnie wybrać (utworzyć) katalog do zapisu projektu (np. RowKwad) z zapisać projekt nadając mu nazwę lub akceptując Project1. Okno projektu (bez wpisanych instrukcji- jest to „pusty” projekt) przedstawiono poniżej.

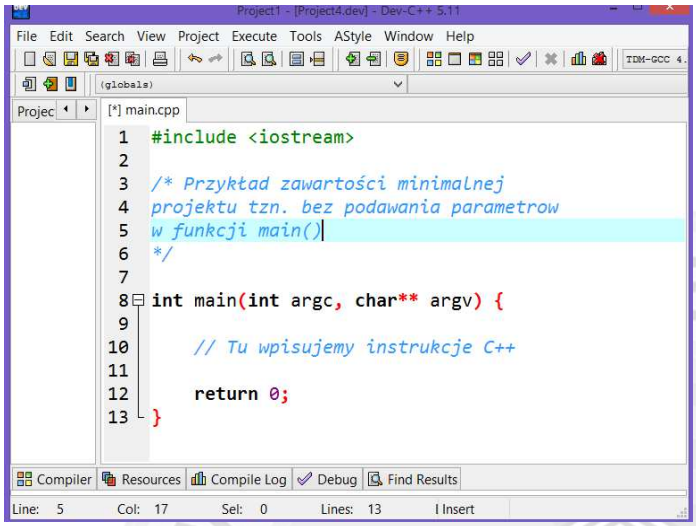


Z kolei poniżej pokazano program (c++ DEV) drukujacy (patrz w czarnym oknie) parametry funkcji main z rysunku powyzej. Dotyczy to wiersza o numerze 6 tzn. int `main`(int `argc`, char** `argv`). Są to parametry, które funkcja przekazuje do system operacyjnego. Z wydruku widać, że jest to ścieżka dostępu do uruchamianego program.

```
1  #include <iostream>
2  using namespace std;
3  int main(int argc, char** argv) {
4  cout<< "Ponizej pokazano\n"<<
5  "Co jest w nawiasie funkcji main?"<<endl;
6  cout<<"argc= "<<argc<<endl;
7  cout<<"argv="<<cout<<argv[0]<<endl;
8  return 0;
9  }
```

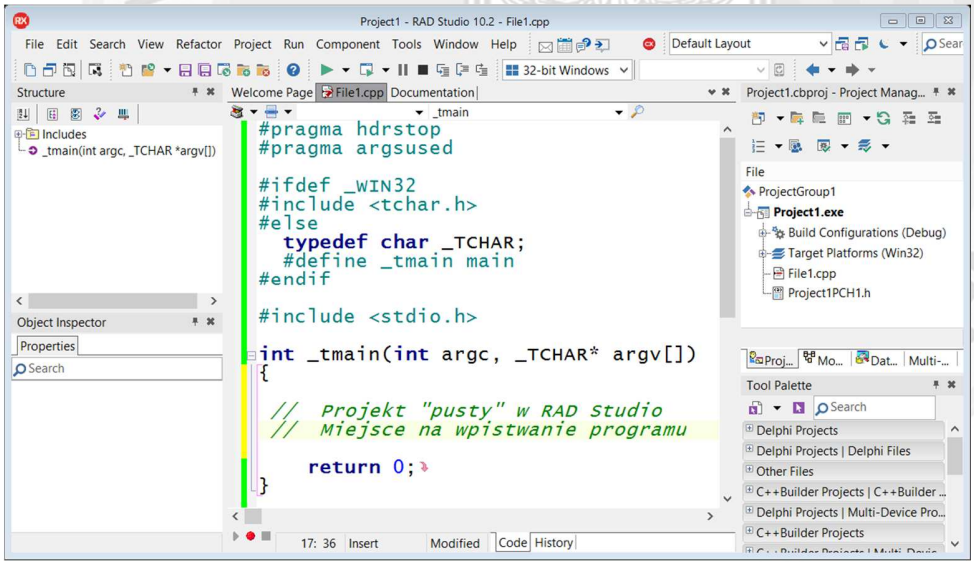


Na etapie nauki pisania prostych pragramów wystarczy by delinicja funkcji głównej była jak poniżej.



RAD Studio 10.2 (Tokyo)

W środowisku RAD Studio należy przeprowadzić podobne postępowanie jak dla DEV. Okno projektu dla RAD Studio przedstawiono poniżej.



W środowisku RAD definicja funkcji głównej może być identyczna jak dla DEV – w wersji najbardziej zubożonej.

Struktura funkcji main( ) w C++

```
// - pojedyncza linia komentarza
/* początek długiego (kilka linii) komentarza
.....
.....
*/
#include < Plik> // pliki nagłówkowe (może być ich kilka )
...
int main( )   // nawias może nie zawierać parametrów
{
        deklaracje zmiennych;
              ...
        Instrukcja 1;
        Instrukcja 2;
              ...
   system(''pause'');
   return 0;
}
```

Instrukcje:
- podstawienia;
- obliczane wyrażenia matematyczne;
- obliczane wyrażenia logiczne
- instrukcje sterujące (if, switch, goto) ;
- pętle (for, while, do)
- itp

Dyrektywy preprocesora w programach C++
Obsługa WE/WY w C++

Dyrektywy preprocesora są włączane przez polecenia #include <Plik>. Podstawowe to:

stdio - Standard Input/Output- funkcje standard. obsługi we/wy;
conio - Konsole Input/Output - funkcje obsługi we/wy konsoli;
iostream - I/O stream - obiekt obsługi we/wy;
cmath (math) - zbiór funkcji matematycznych;
iomanip - zestaw funkcji pozwalajacych na redagowanie wygladu informacji
fstream – umoźliwia realizację operacji związ zapisu/odczytu w pliku dyskowym.
ctime – umożliwia wykorzystanie daty I czasu
cstdlib – umożliwia korzystanie z generatorów liczb

Są to nazwy gotowych, „firmowych" plików tzw. „nagłówkowych" (ang.: header files) o charakterystycznym rozszerzeniu *.h.
W uproszczeniu, można przyjąć, że:

- do programów w C i C++ należy dołączyć pewne pliki nagłówkowe,
- ilość i rodzaj tych plików zależy od tego, jakie funkcje są stosowane w programie.

Jeśli stosujemy funkcję biblioteczną C, to należy dyrektywą #include <.....> dołączyć podane wyżej pliki nagłówkowe. Pozwalają one przykładowo na wykorzystanie poniższych funkcji .

Nazwa	Co to jest?	Co robi?	Plik nagłówk.	Biblioteki
getch( )	funkcja	wczytuje pojedynczy znak z klawiatury	<conio.h>	c
sqrt()	funkcja	oblicza pierwiastek kwadratowy	<math.h>	c
puts( )	funkcja	wyprowadza na wyjście tekst	<stdio.h>	c
scanf()	funkcja	wczytuje dane z klawiatury	<stdio.h>	c
printf( ) cprintf()	funkcja	wyprowadza dane (*wymaga podania wczesniej koloru np.:*	<stdio.h>	c
cout	obiekt	wyprowadza dane	<iostream.h>	C+ +
cin	obiekt	wczytuje dane z klawiatury	<iostream.h>	C+ +

Znaki specjalne w tekstach wyprowadzanych na konsolę (przez obiekt cout, instrukcje printf, cprintf i puts):

***__*n*__*- nowa linia (new line),*

***__*t*__ *– tabulacja pozioma (do wyrównywania tekstów o rożnej długości),*

***__*r*__*- powrót kursora (wskazującego na konsoli m-ce wydruku kolejnego znaku) go początku wiersza,*

***__*a*__ *– generuje znak dźwiękowy (np. żeby zwrócić uwagę iż należy coś wpisać.*

***__*b*__*- cofnij kursor o jeden znak*

Znaki formatujące w cout, printf() i puts():

Znak	Zastosowanie	Uwagi
%s	Do wyprowadzania napisów	*printf"(Nazwisko: %s ", N"Kowal")*
%c	Do wstawiania pojedynczego znaku	
%d	Do wstawiania liczby typu int	*Można użyć też printf"(" ILOSC: %5d ", IL)*
%f	Do wstawiania liczby float, double	*printf"(" Wynik: %8.4d ", Wyn)*
%x	Wyprowadzi zmienną typu integer w formacie heksadecymalnym	*printf"(" Wynik: %x ", 254) da wydruk Wynik: fe*
%X	Wyprowadzi zmienną typu integer w formacie heksadec. (duze litery)	
%e	Wyprowadzi zmienną w formacie wykładniczym	
%g	Automatyczny wybór formatu f lub g	

Typy danych w programach C++ i sposób ich deklaracji

Aby przechowywanie i wyszukiwanie danych w pamięci przebiegało poprawnie, w językach programowania dane są podzielone na pewne *typy danych*. C++ wyróżnia cztery podstawowe typy (standard ANSI), char, int, float i double.

Występują też modyfikatory: *signed/unsigned oraz long/short*

Typ	Znak	Ilość bajtów	Zakres wartości
char	jest	**1**	0... + 255
int	jest	**2**	-32768…+32767
short			jw
long	jest	**4**	-2 147 483 648 … 2 147 483 647
usigned char	-	**1**	0-255
unsigned int	-	**2**	0- 65 535
unsigned short	-	**2**	jw
enum	-	**2**	jw
float	jest	**4**	-3.4E+38 .... + 3.4E+38 (dokładność: 7 cyfr)
double	jest	**8**	-1.7E+308... + 1.7E+308 (dokładność: 15 cyfr);
long double	jest	**10**	-3.4E+4932 …+3.4E+4932
void	-		dotyczy parametrów funkcji

Przykłady deklaracji zmiennych w programie:

```
char imie[20],  Nazwisko[30]  ;
unsigned int rok, procent;
float  wynik;
int  i, j, tabl[ ]={- 7, -1, 3, 7, 17, 23}; // tablica zawierajaca 6 liczb
double suma;
char szef[ ]=”Kowalski Jan”  //tablica szef zawierajaca nazwisko i imie
```

ZŁOŻONE TYPY DANYCH struct

U1.cpp

```
// ZLOZONE STRUKTURY DANYCH - UZYCIE struct
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <string.h>
struct ADRES {
  char Ul[10];
  int  Nr_d;
  int  Nr_m;
};
struct KREWNI {
  char Nazw[12];
  char Imie[10];
  char Pokr[8];
  int   Tel;
    struct ADRES M_Zam;
};
     // TABLICE Z DANYMI STR
int main()
{ struct KREWNI SPIS[30];
                  // Definiowanie i zmiany d
cout <<"----Wprowadzanie z klawiatury----\n";
  cout << "Podaj nazw.: ";
  cin  >>  SPIS[1].Nazw  ;
  cout << "Podaj imie : ";
  cin  >>  SPIS[1].Imie ;
  cout << "Podaj ulice: ";
  cin  >>  SPIS[1].M_Zam.Ul ;
  cout << "Podaj nr   : ";
  cin  >> SPIS[1].M_Zam.Nr_d;

cout << "\n Wprowadzono dane: ";
cout << "\n    nazwisko:   " << SPIS[1].Nazw;
cout << "\n    imie:       " << SPIS[1].Imie;
cout << "\n    ulica       " << SPIS[1].M_Zam.Ul;
cout << " Nr " << SPIS[1].M_Zam.Nr_d << endl;

cout <<"\n--Inna forma wprowadz/zmiany danych w tabl.:-
SPIS[2].Tel = 1234567;
SPIS[2].M_Zam.Nr_d = 3;
strcpy(SPIS[2].Nazw    ,"Kowalska");
strcpy(SPIS[2].Imie    ,"Malgorzata");
strcpy(SPIS[2].M_Zam.Ul, "Mila");

  printf("\n nazw : %s" ,  SPIS[2].Nazw);
  printf("\n imie : %s" ,  SPIS[2].Imie);
  printf("\n ulica: %s" ,  SPIS[2].M_Zam.Ul);
  printf("\n Nr   : %d" ,  SPIS[2].M_Zam.Nr_d);
getch();
return 0;
}
```

```
----Wprowadzanie z klawiatury----
Podaj nazw.: NOWAKOWSKA
Podaj imie : ALICJA
Podaj ulice: Dluga
Podaj nr   : 123

 Wprowadzono dane:
    nazwisko:   NOWAKOWSKA
    imie:       ALICJA
    ulica       Dluga Nr 123

--Inna forma wprowadz/zmiany danych w tabl.:--
 nazw : Kowalska
 imie : Malgorzata
 ulica: Mila
 Nr   : 3
```

Rezerwacja tablicy o nazwie SPIS na 30 obiektów typu KREWNI

Tu wprowadzane są informacje o jednym obiekcie typu KREWNI i zapamiętywane są w tablicy SPIS jako jej pierwszy element

Operatory arytmetyczne

Operatory addytywne

- **x + y** *//dodawanie*
- **x - y** *//odejmowanie*

Przykład

```
int x=32, y=12, w1,w2,w3;
w1= x +y;
w2= x –y;
w3= x - -x;
```

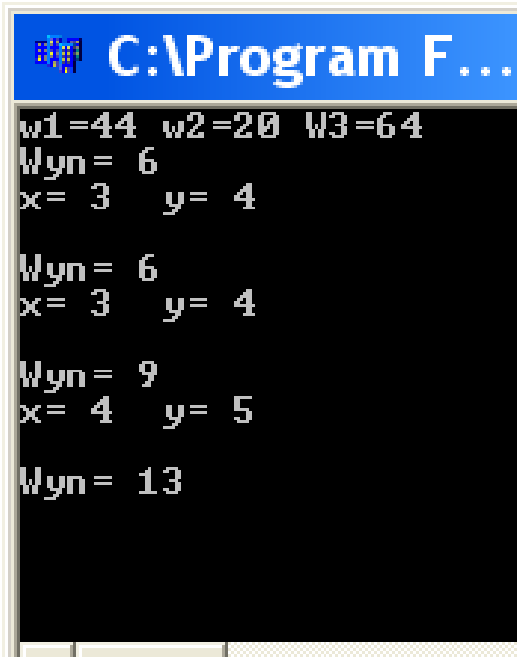
Operatory inkrementacji i dekrementacji

- **przedrostkowy** **--op** **++op**
- **przyrostkowy** **op--** **op++**

Przykład:

```
int x=32, y=12, w1,w2,w3, wyn;
  w1= x +y;
  w2= x -y;
  w3= x - -x; //

  x=2, y=3;
  wyn= (x++) + (++y); // 2+4 !
// tu: wyn= 6 x=3, y=4
  x=2, y=3;
  wyn= x++ + ++y;   // wyn→6
  wyn= (++x)+ (++y); // wyn→9
  wyn += x;          // wyn=13
```



Operatory multiplikatywne

- **x * y** **//mnożenie**
- **x / y** **//dzielenie**

Operatory bitowe

Operatory funkcji binarnych

- `x & y` // *x AND y*
- `x ^ y` // *x EXOR y*
- `x | y` // *x OR y*
- `~ y` //*NOT y*

Przykład:

```
int     x= 6, y=3, w;   //x= 0...0110
                        // y=0...0011
w=x & y;          // w = 2;   0...010
x=31, y=7          // x= 0...0011111
                   //  y= 0...0000111

w=x ^y   //w=24     // w=0...0011000
x=24, y=7
w=x | y  // wynik: w=31
```

Operatory przesunięcia

Wyrażenie z operatorami przesunięcia

- `x<< y` *//przesuń x w lewo o y bitów*
- `x >> y` *//przesuń x w prawo o y bitów*

Przykład:

```
int             a, b;
a =2;                   //a=2;      0... 0010
b=a<<2;                 //b= 8;     0... 1000
a= 13;                  //a= 13;    0...01101

b= a >> 1;              //b=6;      0... 0110
```

Instrukcje sterujące

Instrukcja „albo, albo” if…else

```
if      (Wyrazenie1) {
  Instrukcja1;
}
else if  (Wyrazenie2)
  Instrukcja2;
else if  (Wyrazenie3)
      ■ ■ ■
else
  InstrukcjaN;
```

- ….// ciąg dalszy programu

W instrukcji:

- Poszczególne **Wyrażenia** przyjmują wartość prawdy lub fałszu (są wyrażeniami logicznymi),
- Jeśli prawdziwe jest np. **Wyrażenie 2** to wykonywana jest **Instrukcja2**.
- Jeśli żadne z wyrażeń jest nieprawdziwe to wykonana zostanie **InstrukciaN**

Wyrażenia i operatory logiczne w instrukcjach sterujących

Operatory i wyrażenia logiczne

- **x && y** *//koniunkcja*
- **x || y** *//alternatywa*
- **! x** *//negacja*

Przykład użycia operatorów:

```
if ( (x>-10 && x< -1 ) || (x>1 && x< 10) )
    cout << "x jest w przedziale";      // True
else
   cout<< "x jest poza  przedzialem"; //False
```

Przykład:

```
int      c,d, x= 1, y= 2;
         //binarnie: x= 0...001; y= 0...010;
c=x & y;   //dla bitowego & operatora będzie c=0
d=x && y;  //a dla operatora && koniunkcji d=1
```

INSTRUKCJE STERUJĄCE Instrukcja goto

Instrukcja **go to** (inaczej inst.. skoku) przekazuje sterowania do określonego miejsca wewnątrz wykonywanej funkcji
Wywołanie:

```
goto    etykieta;

    ■ ■ ■
etykieta:
   instrukcja;
```

Miejsce etykiet w programie:

- ***etykieta przed instrukcją skoku***

```
        ■ ■ ■
    etykieta1:
                ■ ■ ■
        goto    etykieta1;
```

- **etykiety po instrukcji skoku**

```
        ■ ■ ■
        goto   etykieta2;
         ■ ■ ■
    etykieta 2:
```

Instrukcje sterujące

Instrukcja „wyboru" - switch

Umożliwia wybór jednego z wielu wariantów w zależności od wartości zmiennej sterującej.

Przykład:
Niech W będzie oceną cało- liczbową. Należy napisać skrót oceny.

```
...
int W;
   cin >> W;  // Wpisz ocenę -liczba całkowita
   switch (W)  {
     case '2':    cout <<"ndst";  break;
     case '3':    cout <<"dst";    break;
     case '4':    cout <<"db";     break;
     case '5':    cout <<"bdb";  break;
     default:    cout <<"Niewlasciwa ocena";
    }
```

Instrukcje sterujące

Instrukcja „kontynuuj" - **continue**

Jej użycie dopuszczalne jedynie w pętlach
(for, while, do ... while).

- W przypadku pętli **while, do ... while** przeniesienie sterowania z wnętrza pętli do wyrażenia warunkowego
- W przypadku pętli **for** przeniesienie sterowania z wnętrza pętli do wyrażenia zwiększającego licznik pętli
- w przypadku pętli zagnieżdżonych instrukcja jest związana z najbliższą pętlą

Instrukcje sterujące

Instrukcja „przerwij" - break

Jej użycie dopuszczalne jest tylko w instrukcjach iteracyjnych **(for, while, do ... while)** i instrukcji wyboru **(switch).** Powoduje **zakończenie** aktualnego poziomu instrukcji iteracyjnej lub wyboru.

Instrukcje iteracyjne

Instrukcja pętli "dla" - for

```
for (wyrażenie1; wyrażenie2; wyrażenie3)
    {
          ■ ■ ■
      instrukcje;
          ■ ■ ■
    }
```

- **Wyrażenie1**- inicjujące: inicjowanie liczników pętli
- **Wyrażenie2**- warunkowe: jeśli jest niezerowe (prawda) wykonywana jest instrukcja lub blok instrukcji {……}
- **Wyrażenie3** : zwiększenie/zmniejszenie liczników pętli

Nieskończona pętla for

```
for (; ;)      // lub  for(;1;)
{
  instrukcja 1;
  instrukcja 2;
   ■ ■ ■
}
```

Uwaga:

- Jedną z instrukcji pętli musi być instrukcja **break,** gdyż jej brak spowodowałby, że pętla będzie wykonywana nieskończenie wiele razy,
- Tego typu pętla może być też zastosowana jeśli wykorzystywane są tzw. przerwania (dotyczy to specjalizowanych programów).

Przykład wykorzystania dwóch instrukcji for do zmiany koloru tła i czcionki na ekranie konsoli

Uwaga:
Instrukcje cprintf, textcolor nie działają w niektórych środowiskach

Petle_kolor_tla_czcionki.cpp

```
/*Program ilustruje działanie petli zewnetrznej i wewn.
   - Zewnetrzna- zmienia tlo ekranu
   -  wewnetrzna- zmienia kolor napisow na  tle  */

#include <stdio.h>
#include <conio.h>
//-------------------------------------------------------------
int main()
{
int i,j;
  for(i=0; i<=15;i++){          // wybor tla
    textbackground(i);
    cprintf("\n\r Tlo i= %2d",i);

        for (j=15; j>=0; j--){ // kolor napisow
          textcolor(j);
          cprintf(" j=%d",j);
        }
  }
        getch();
        return 0;
}
//-----------------
```

F:\C++M_Tlo_czcionka_ZmianaKoloru\Project1.exe

```
Tlo i=  4 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
Tlo i=  5 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
Tlo i=  6 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
Tlo i=  7 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
Tlo i=  8 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
Tlo i=  9 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
Tlo i= 10 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
Tlo i= 11 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
Tlo i= 12 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
Tlo i= 13 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
Tlo i= 14 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
Tlo i= 15 j=15 j=14 j=13 j=12 j=11 j=10 j=9 j=8 j=7 j=6 j=5 j=4 j=3 j=2 j=1 j=0
```

Instrukcje iteracyjne

Instrukcja "dopóki"- while

```
while ( warunek)
{
      instrukcja1;
      instrukcja2;
      ■ ■ ■
}
```

Instrukcje są wykonywane dopóki **warunek** jest prawdziwy (true)

Uwaga:

Jedna z instrukcji w pętli musi mieć wpływ na wyrażenie **warunek** – gdyż w przeciwnym razie byłaby to pętla absolutnie nieskończona.

Instrukcje iteracyjne

Przykład: nieskończona pętla while

```
■ ■ ■

while (1)
{
        instrukcja1;
        instrukcja2
        ■ ■ ■
}
```

- Instrukcje są wykonywane nieskończenie wiele razy bo warunek=1 jest zawsze prawdziwy,
- Wyjście z takiej pętli może nastąpić jeśli któraś z instrukcji spowoduje, że nastąpi wywołanie instrukcji **break.**

Instrukcje iteracyjne

Instrukcja "wykonuj dopóki" do ... while

```
do
{
    ■ ■ ■
  instrukcja_n;
  instrukcja_n+1;
    ■ ■ ■
}
while (wyrażenie- warunkowe)
```

- Instrukcje są wykonywane do momentu, gdy **wyrażenie-warunkowe** osiągnie wartość fałsz (to wyrażenie jest sprawdzane dopiero po wykonaniu)
- Aby pętla była skończona jedna z instrukcji w jej wnętrzu musi mieć wpływ na wartość wyrażenia- warunkowego

Instrukcje iteracyjne

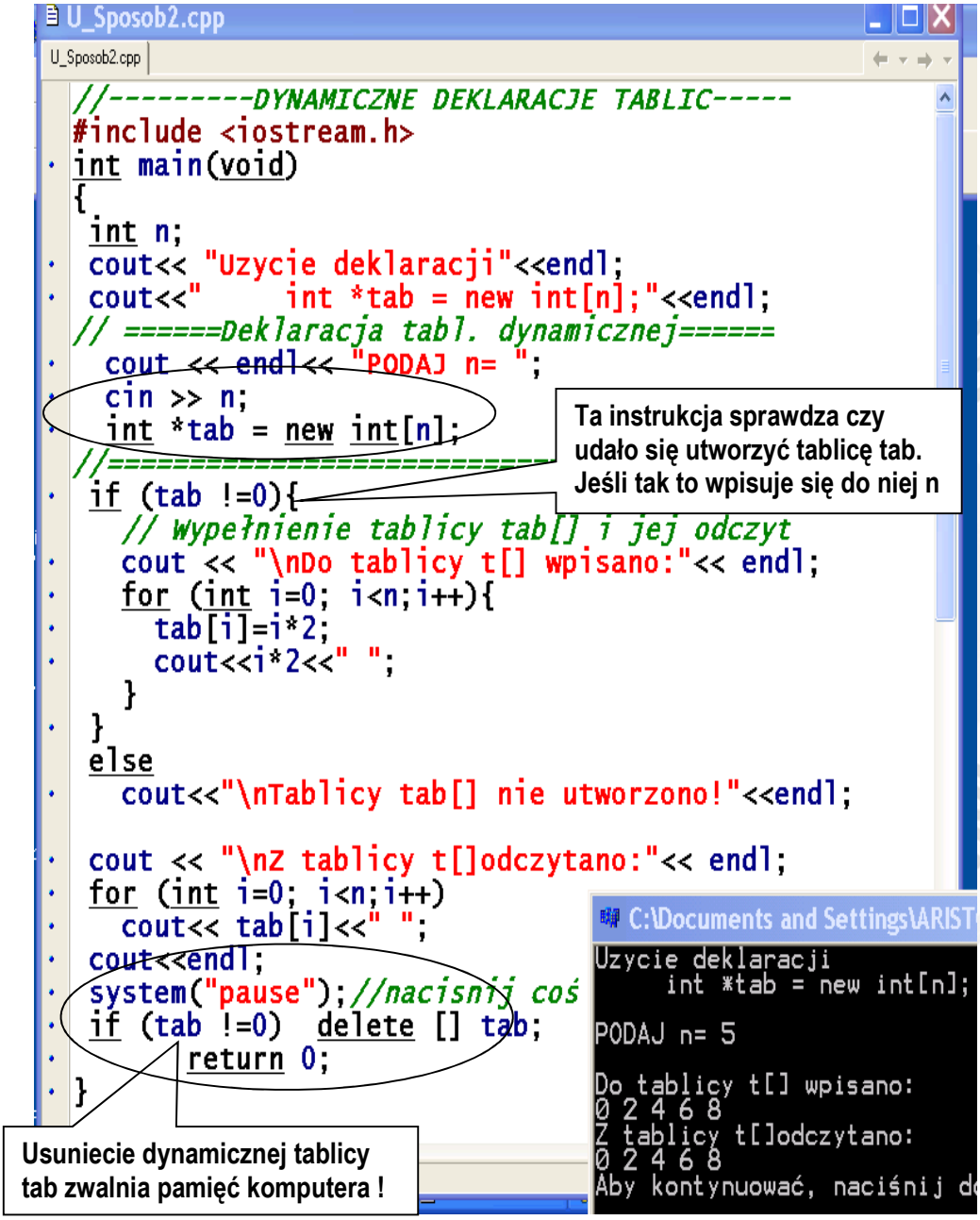
Pętla nieskończona do ... while

```
do {
■ ■ ■
  instrukcja_n;
  instrukcja_n+1;
    ■ ■ ■
} while (1)
```

- Instrukcje są wykonywane nieskończenie wiele razy bo warunek=1 jest zawsze prawdziwy,
- Wyjście z takiej pętli może nastąpić jeśli któraś z instrukcji spowoduje, że nastąpi wywołanie instrukcji **break.**

Tablice dynamiczne

W programie pokazano utworzenie tablicy typu int o n elementach (n-jest podawane przez operatora). Następnie pokazano zapis i odczyt n liczb do/z tablicy tab[ ] Na końcu pokazano usunięcie tablicy z pamięci przed zakończeniem programu.



Użycie wskaźników w operacjach tablicowych

W programie pokazano deklarację zmiennej wskaźnikowej (tu *p) oraz tablicy T typu int zawierającą 7 liczb.

Na przykładzie tej tablicy pokazano dostęp do jej elementów z wykorzystaniem :

- **indeksów elementów tablicy i instrukcji for**
- **wskaźnika adresu p=&T[0] początku tablicy,**
- **operatora *p wyłuskania wartości z pamięci o adresie p,**
- **modyfikacji wskaźnika (*p++ lub *(p+k),**

U_tab1.cpp

```
/* ZASTOSOWANIE WSKAŹNIKÓW DO WYPROWADZANIA
    INFORMACJI Z TABLICY LICZBOWEJ
     np.  {44, 66, 33, 77, 55, 11, 22}      */
#include <iostream.h>
#include <iomanip.h>
int main()
{
 char znak;
 int *p,k, N,
  T[]={44, 66, 11, 77, 55, 33, 22};
 // *p - tzw. zmienna wskaźnikowa
 N=sizeof(T)/4;  //sizeof - daje wynik w bajtach
 cout <<" I-  WYPROWADZANIE LICZB z T[]";
 cout <<" -UZYCIE INDEKSOW\n";
 for (k=0; k<N;k++)
    cout <<setw(5)<<T[k];
 cout <<"\n II- WYPROWADZANIE LICZB z T[]" ;
 cout <<" - UZYCIE WSKAZNIKOW\n";
 // &T[0]- operator ustala adres początku T[]
 // i zapamietuje go w zmiennej p
 p=&T[0];
 cout<<"Adres pocz. tablicy T= "<<hex<<p<<endl;
 for (k=0; k<N; k++)
    cout<< setw(5)<< dec<< *p++;
 //*p -operator wyłuskania zawartości z adresu p
 cout <<"\n III- WYPROWADZANIE LICZB z T[]" ;
 cout <<" -INNE UZYCIE WSKAZNIKOW\n";
 p=&T[0];
 for (k=0; k<N; k++)
    cout<< setw(5)<< *(p+k);
 cin>> znak;
}
```

```
 I-  WYPROWADZANIE LICZB z T[] -UZYCIE INDEKSOW
   44   66   11   77   55   33   22
 II- WYPROWADZANIE LICZB z T[] - UZYCIE WSKAZNIKOW
Adres pocz. tablicy T= 12ff60
   44   66   11   77   55   33   22
 III- WYPROWADZANIE LICZB z T[] -INNE UZYCIE WSKAZNIKOW
   44   66   11   77   55   33   22
```

Tablice dwu wymiarowe, zastosowanie instrukcji for do ich indeksowania

W poniższym programie pokazano:

- deklarację tablicy dwu wymiarowej,
- zastosowanie dwóch pętli for do indeksowania elementów tablicy wypełnienia jej liczbami ułamkowymi (pętla zewnętrzna indeksuje wiersze, wewnętrzna elementy w kolumnach),
- wyświetlenie liczb z tablicy z precyzją 4 miejsc po przecinku (instrukcją printf).
- rzutowanie typów (tu działania na zmiennych typu int dają wynik float).

Unit1.cpp

```
#include <iomanip.h>
#include <iostream.h>
#include <conio>
int main()

{
  double elem, Tab[10][10];
  int     i, j;
  char znak;

    for (i= 0; i<10; i++) {
      for (j= 0; j<10; j++)
        {
          elem=double(i + j)/ (i +j+1);
          Tab[i][j]= elem;
        }
    }

  // teraz wyswietl  Tab[i,j]
  for (i= 0; i<10; i++) {
      for (j= 0; j<10; j++)
      printf("%6.4f  ",Tab[i][j]); //tu nie cout
      cout<<endl;
      }
   cin >> znak;
    return 0;
}
```

RZUTOWANIE TYPU double

Jeśli wpisać tylko:
elem=(i + 1) / (i+j+1)
to zawsze będzie elem =0!
I cała tablica T będzie zawierała zera. Błąd!

C:\Program Files\Borland\CBuilder6\Projects\Project1.e...

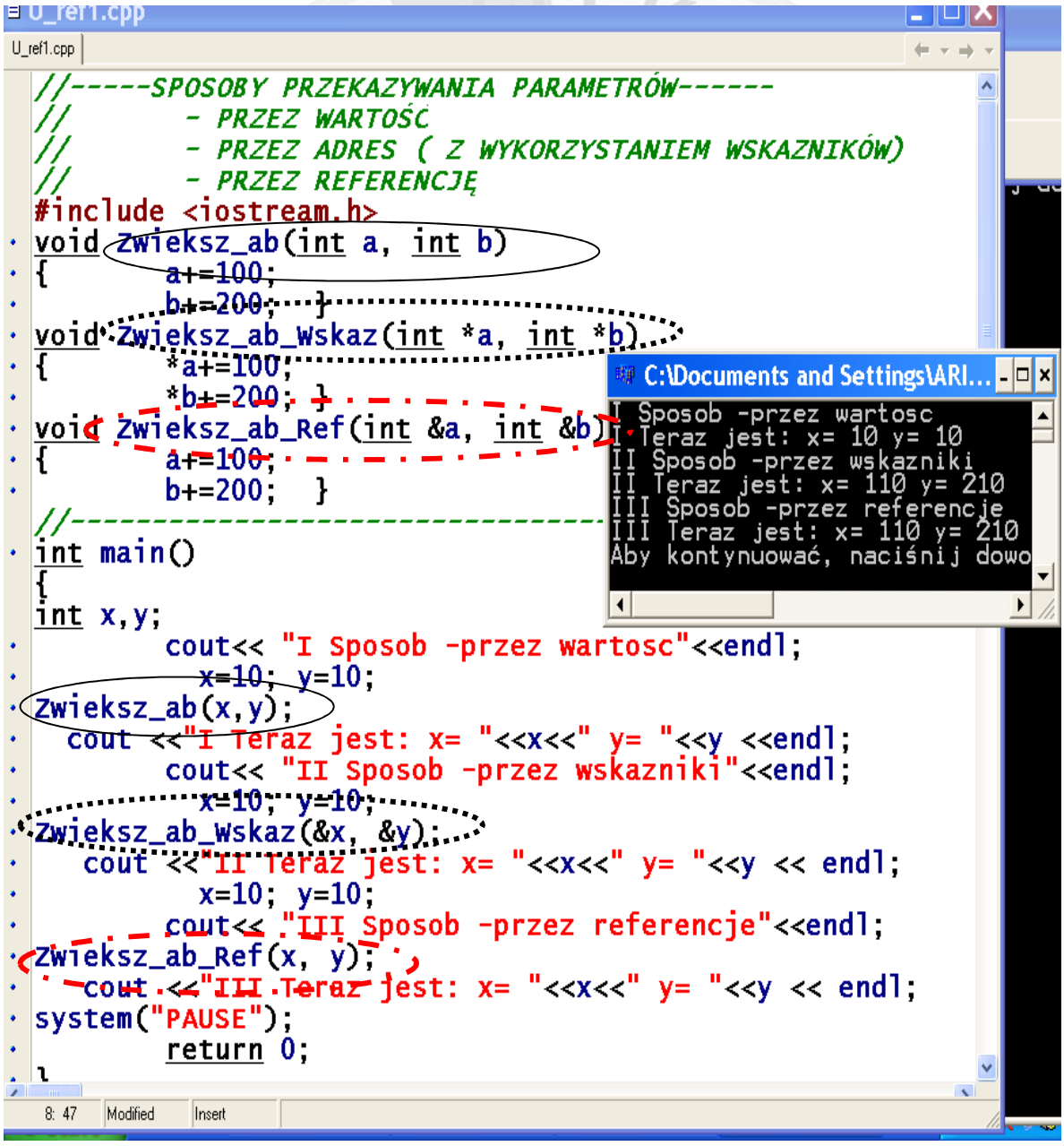
```
0.0000  0.5000  0.6667  0.7500  0.8000  0.8333  0.8571  0.8750  0.8889  0.9000
0.5000  0.6667  0.7500  0.8000  0.8333  0.8571  0.8750  0.8889  0.9000  0.9091
0.6667  0.7500  0.8000  0.8333  0.8571  0.8750  0.8889  0.9000  0.9091  0.9167
0.7500  0.8000  0.8333  0.8571  0.8750  0.8889  0.9000  0.9091  0.9167  0.9231
0.8000  0.8333  0.8571  0.8750  0.8889  0.9000  0.9091  0.9167  0.9231  0.9286
0.8333  0.8571  0.8750  0.8889  0.9000  0.9091  0.9167  0.9231  0.9286  0.9333
0.8571  0.8750  0.8889  0.9000  0.9091  0.9167  0.9231  0.9286  0.9333  0.9375
0.8750  0.8889  0.9000  0.9091  0.9167  0.9231  0.9286  0.9333  0.9375  0.9412
0.8889  0.9000  0.9091  0.9167  0.9231  0.9286  0.9333  0.9375  0.9412  0.9444
0.9000  0.9091  0.9167  0.9231  0.9286  0.9333  0.9375  0.9412  0.9444  0.9474
```

Program strukturalny, sposoby przekazywania parametrów

Pokazano 3 sposoby przekazywania parametrów:

- **przez wartość funkcja o nazwie: Zwieksz_ab( )**
- **przez adres funkcja o nazwie: Zwieksz_ab_Wskaz( )**
- **przez referencję funkcja o nazwie: Zwieksz_ab_Ref( )**

Przed wywołaniem każdej z funkcji ustawiano x=10 y=10.
Pokazano jak zmieniają się wartości x i y po wykonaniu funkcji

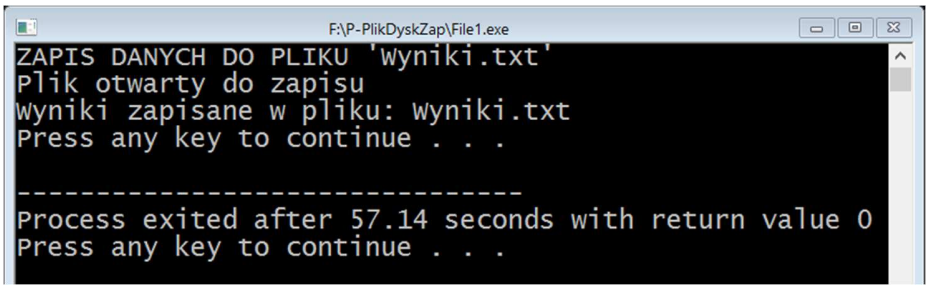


Praca z plikami

Przykład:

Napisz program obliczający wartości funkcji sin i cos w przedziale [0 ,180] stop. z krokiem 10 stop. Wartości argumentu oraz sin i cos **zapisz w pliku** dyskowym Wyniki.txt.

```
1   //-------ZAPIS DANYCH DO PLIKU DYSKOWEGO----------------
2   #include <iostream>
3   #include <fstream>
4   #include <cmath>
5   #include <string>
6   using namespace std;
7   int main( ) {
8   char znak; int pole=10; double x;
9   cout<<"ZAPIS DANYCH DO PLIKU 'Wyniki.txt'"<<endl;
10  ofstream Zbior("Wyniki.txt");
11  if (Zbior) // Jesli otwarty to wyświetl
12  {
13    cout << "Plik otwarty do zapisu"<<endl;
14   Zbior << noskipws; // zapobiega pomijaniu spacji
15  // i znaków sterujących
16    Zbior.width(pole); Zbior << "x";
17    Zbior.width(pole); Zbior << "sin( x)";
18    Zbior.width(pole); Zbior << "cos( x)" <<endl <<endl;
19    Zbior.precision(5);// 5 m-ca po kropce
20    Zbior << showpoint;// zawsze wstaw kropke
21    Zbior << fixed; // nie wykładniczo
22    for (int alfa=0; alfa<=180; alfa=alfa +10) {
23        x=alfa*M_PI/180;
34        Zbior.width(pole); Zbior << alfa;
45        Zbior.width(pole); Zbior << sin(x);
26       Zbior.width(pole); Zbior << cos(x) <<endl;
27     }
28      cout << "Wyniki zapisane w pliku: Wyniki.txt\n";
29  }
30  else
31        cout << "PLIKU NIE UTWORZONO !" << endl;
32  return 0;
33  }
```



Przykład:

Napisz program **odczytujący** wartości funkcji sin i cos w przedziale [0 ,180] stop. z krokiem 10 stop. zapisane w pliku dyskowym Wyniki.txt.

```
        //-------Odczyt DANYCH Z PLIKU DYSKOWEGO---------------
1       #include <iostream>
2       #include <fstream>
3       using namespace std;
4       int main( ) {
5         char znak;
6         int i=0;
7         cout<<"ODCZYT DANYCH Z PLIKU 'Wyniki.txt'"<<endl;
8         ifstream Zbior("Wyniki.txt");
9          if (Zbior) // Jesli otwarty to wyświetl
10         {
11            cout << "Plik otwarty do odczytu"<<endl;
12            Zbior >> noskipws; // zapobiega pomijaniu spacji
13                               // i znaków sterujących
14            while (Zbior >> znak)  {
15               cout << znak ; //Petla wyprowadza znak po znaku
16            }
17         }
18         else
19             cout << "PLIK NIEDOSTEPNY !" << endl;
20      return 0;
21      }
22
```

```
F:\P-PlikDyskZap\File1odcz.exe
ODCZYT DANYCH Z PLIKU 'Wyniki.txt'
Plik otwarty do odczytu
         x   sin( x)   cos( x)

         0   0.00000   1.00000
        10   0.17365   0.98481
        20   0.34202   0.93969
        30   0.50000   0.86603
        40   0.64279   0.76604
        50   0.76604   0.64279
        60   0.86603   0.50000
        70   0.93969   0.34202
        80   0.98481   0.17365
        90   1.00000   0.00000
       100   0.98481  -0.17365
       110   0.93969  -0.34202
       120   0.86603  -0.50000
       130   0.76604  -0.64279
       140   0.64279  -0.76604
       150   0.50000  -0.86603
       160   0.34202  -0.93969
```

Literatura:

[1] Walczak-Struzińska Anna, Walczak Krzysztof, Nauka programowania dla początkujących C++, Wydawnictwo W&W ion 2016

[2] Stasiewicz Andrzej, C++ Ćwiczenia 2004, Helion 2004

[3] Daniluk Andrzej, C++ Builder. Ćwiczenia, Helion 2003

[4] Grębosz Jerzy, Symfonia C++ Standard. Programowanie w języku C++ orientowane obiektowo, Edition 2000

[5] Majczak A., C++ przykłady praktyczne, Wydawnictwo Mikon 2003