Podstawy programowania w języku C i C++.

Tematy:

Funkcje: deklarowanie, definiowanie, wywołanie

Cele nauczania: Podstawowe zasady programowania.
Praktyczna znajomość elementów programowania w języku C/C++

Docelowo : ***Opracowanie algorytmu wybranego zadania geodezyjnego i przedstawienie go w postaci schematu blokowego. Opracowanie programu w wybranym języku programowania Wykonanie dokumentacji programu .***

Szczegółowe dla lekcji:

Zapoznanie się z pojęciem funkcji, deklarowaniem, definiowaniem, wywoływaniem.

Umiejętność zastosowania funkcji w prostych programach.

Praktyczne wprowadzanie, uruchamianie i kompilacja programów z instrukcjami wejścia/wyjścia..

Cele operacyjne lub szczegółowe cele kształcenia

Po zakończeniu lekcji uczeń będzie umiał:

tworzyć i stosować podstawowe proste funkcje w programach C i C++

Część wstępna

Część wstępna - organizacyjna:

przywitanie, sprawdzenie obecności, wpisanie tematu lekcji.

Część merytoryczna

Ogniwo 1. Uświadomienie uczniom celów – *pogadanka*

Celem zajęć jest

zapoznanie się z pojęciem funkcji i jej zastosowaniem.

Po 2 lekcjach :

wpisuje i uruchamia programy, z zastosowaniem funkcji

Ogniwo 2 **Poznanie nowych faktów** – pogadanka

- W języku C i C++ można stosować funkcje.
- W C **funkcja** to wydzielona część programu, która przetwarza argumenty i ewentualnie zwraca wartość, która następnie może być wykorzystana jako argument w innych działaniach lub funkcjach.
  Funkcja może posiadać własne ***zmienne lokalne.***
- W języku C istnieją funkcje **biblioteczne** opracowane przez producenta danej implementacji translatora języka, realizujące pewne standardowe operacje.
  Nie wszystko da się jednak wcześniej przewidzieć i standaryzować.
  Użytkownik – programujący często musi opracowywać **własne, niestandardowe funkcje.**
- **Najważniejsze korzyści ze stosowania funkcji**:
  - Program z wykorzystaniem funkcji jest bardziej czytelny i zrozumiały
  - Jest bezpośrednia odpowiedniość między algorytmem a tekstem programu
  - Pewne powtarzające się fragmenty lub realizujące ściśle określone operacje mogą być wyodrębnione i zapisane w postaci jednej funkcji.

Ogniwo 3. Kształtowanie pojęć, poznawanie, systematyzowanie wiedzy -

- **Funkcje** w języku C są podobne do procedur i funkcji w Pascalu czy podprogramów w Fortranie.
  W języku C nie można jednak dokonywać **zagnieżdżeń** w funkcjach.
- Zaleca się by liczba linii tekstu nie przekraczała **60**.
- **Funkcje** pozwalają wydajnie skrócić tekst programu, gdy pewne jego fragmenty się powtarzają.
- Użytkownik może tworzyć własne biblioteki zorientowane problemowo (np. matematyka, raporty, grafika itp.), co pozwala zaoszczędzić czas przy powtarzających się lub podobnych problemach.
- W języku C **funkcja** to **wydzielona część programu**, która przetwarza argumenty i ewentualnie zwraca wartość, która następnie może być wykorzystana, jako argument w innych działaniach lub funkcjach.
- Funkcja może posiadać własne zmienne lokalne.
- **Najważniejsze korzyści ze stosowania funkcji:**
  - Program z wykorzystaniem funkcji jest bardziej czytelny i zrozumiały
  - Jest bezpośrednia odpowiedniość między algorytmem a tekstem programu
  - Pewne powtarzające się fragmenty programu lub realizujące ściśle określone operacje mogą być wyodrębnione i zapisane w postaci jednej funkcji.

Instrukcje wejścia i wyjścia w języku C - powtórzenie:

Wejście i wyjście programu

- Do podstawowych funkcji języka C, umożliwiających komunikację z otoczeniem należą:

dla operacji **wyjścia: putchar, puts, printf;**

dla operacji **wejścia**: **getchar, gets, scanf;**

Wejście:

- **Funkcja putchar** wysyła pojedynczy znak na zewnątrz - standardowo na ekran.
- **Funkcja puts()** wysyła łańcuch **s** do standardowego strumienia
- wyjściowego (stdout)
- Funkcja **printf()** jest bardziej uniwersalna. *Może wyświetlić dane dowolnego typu i współpracować z wieloma argumentami*.
  printf(ciag_formatujący, lista parametrów);
  Ciąg formatujący: **%d, %f, %c, %s, %o, %x, %e, %g**

Operacje wejścia wprowadzanie danych w C i C++

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

Podsumowanie – wejście, wyjście, liczby, zmienne, wzorce konwersji

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

Preprocesor, dyrektywy, funkcje biblioteczne C

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

Moduły biblioteczne dołącza się przez instrukcję
#include <nazwa>

Standardowo dołączamy **stdio.h** w C a **iostream.h** w C++

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

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

Dyrektywa **#define** definiuje stałą symboliczną lub
makrodefinicję,
np. #define NMAX 20 definiuje stałą NMAX o wartości 20.

Stałe symboliczne

Stała symboliczna jest nazwą zastępującą ciąg znaków

#define NAZWA tekst

Np.

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

Stałe symboliczne - makrodefinicje

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

Makrodefinicje proste:

#define identyfikator <ciąg-jednostek-leksykalnych>

#define PI 3.14159

#define TRUE 1

#define FALSE 0

#define NAPIS1 Siemianowice

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

#define IMIE_I_NAZWISKO IMIE+"Zalewski"

#define WCZYTAJ_IOSTREAM_H #include <iostream.h>

Makrodefinicje parametryczne

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

Funkcje

- **Funkcja** to blok instrukcji realizujących określony cel.
- Funkcje mogą być biblioteczne lub własne – napisane przez programistę
- Wywołujemy funkcję, aby wykonać zadanie, możemy przekazywać argumenty do funkcji a funkcja może zwracać wynik.
- Funkcje są szczególnie użyteczne, *kiedy trzeba wielokrotnie wykonywać to samo zadanie.*
- Z funkcji korzystamy także dla wspomagania organizacji trudnego programu. Gdy programy stają się większe, to praca z jedną dużą funkcją main() staje się coraz trudniejsza. Program można podzielić na części wykorzystując funkcje.
- Program dzielimy na sekcje, z których każda wykonuje oddzielne kompletne zadanie. Sekcje można podzielić na mniejsze jednostki.
- Tworzymy pojedyncze funkcje do wykonywania oddzielnych zadań i po powiązaniu ich razem w funkcji głównej main() powstaje gotowy cały program

Pojęcie funkcji w C

- W C **funkcja** (czasami nazywana podprogramem lub procedurą) to wydzielona część programu, która przetwarza argumenty i ewentualnie zwraca wartość, która następnie może być wykorzystana jako argument w innych działaniach lub funkcjach.
- Funkcja może posiadać własne zmienne lokalne.
- W odróżnieniu od funkcji matematycznych, funkcje w C mogą zwracać dla tych samych argumentów różne wartości.
- Funkcję w języku C tworzy się następująco:

typ identyfikator (typ1 argument1, typ2 argument2, typ_n argument_n)

{ /* instrukcje */ }

- Przykład funkcji:

int **iloczyn** (int x, int y)

{ int iloczyn_xy; iloczyn_xy = x*y; **return** iloczyn_xy; }

- Przyjęło się, że **procedura** od funkcji różni się tym, że ta pierwsza nie zwraca żadnej wartości. Zatem, aby stworzyć procedurę należy napisać:

void identyfikator (argument1, argument2, argument_n)

{ /* instrukcje */ }

Funkcje - przykłady

/* Przykład 1 - program do obliczenia objętości walca z wykorzystaniem funkcji */

```
/* Program p53c - Objętość walca - funkcja */
#include <stdio.h>
#include <conio.h>
#define PI 3.1415926
float  f_objwalca(float radius, float height);     // deklaracja funkcji – parametry formalne
int main()
{     float r, h, o;
      puts("Obliczenie objętości walca ");
      printf("Podaj promień r i wysokość h (oddzielone spacja) ==> ");   scanf("%f %f", &r, &h);
      o = f_objwalca(r, h);  // wywołanie funkcji – parametry aktualne r, h
      printf("Objętość walca o R = %.3f  i H = %.3f wynosi %.3f", r, h, o);
      getch();        return 0;
}
float  f_objwalca(float radius, float height)  // definicja funkcji – parametry formalne
{     return(PI*radius*radius*height);  }
```

Program składa się z 2 funkcji: **main()** – głównej, bez parametrów formalnych oraz **f_objwalca**() z 2 parametrami formalnymi.

Przykład 2. **Obliczenie pola koła – funkcja**

```
// polekola.cpp  - funkcja  Dev C++ lub C++ Builder
#include <cstdlib>    #include <iostream>    #include <math.h>   #include <conio.h>
using namespace std;
float pole_kola(float);
int main()
{
   float promien;
   cout << "Pole kola o promieniu 1 wynosi " << pole_kola(1) << endl;
   cout << "Podaj promien "; cin >> promien;
   cout << "Pole kola o promieniu " << promien << "  wynosi " << pole_kola(promien) << endl;
   getch();       return 0;
}
float pole_kola(float r)
{
   float pole;
   pole = M_PI * r * r;  // M_PI = 3.14159...
   return pole;
}
```

Przykład 3 – dodawanie 2 liczb – Dev C++

```
// dodaw.cpp - funkcja dodawanie() - Dev C++
#include <cstdlib>      #include <iostream>   using namespace std;

float dodawanie(float a, float b);                  // deklaracja funkcji

int main (int argc, char *argv[])
{     float la, lb;
      cout << "Dodawanie 2 liczb " << endl;
      do       // wykonuj instrukcje cyklu - pętla do ... while
      {        // choć raz się wykona - warunek sprawdzany na końcu
               cout << "Wprowadź 2 liczby (0   0 - koniec)  ";    cin >> la >> lb;
      cout << la << " + " << lb << "= ";              cout << dodawanie(la,lb) << endl;
   } while (la !=0);   // podczas gdy la <> 0
      system("PAUSE");       return EXIT_SUCCESS;
}

float dodawanie(float a, float b)   // definicja funkcji
{
   return a+b;
}
```

Definiowanie funkcji: deklaracja, definicja

- **Deklaracja funkcji** zgodnie ze standardem ANSI języka C:
  typ_funkcji nazwa_funkcji (lista_parametrów_formalnych); np. **int suma2(int a, int b);**
- Typ funkcji należy do zbioru typów dozwolonych w języku C.
  Nazwa funkcji jest budowana analogicznie jak nazwa zmiennej (bez spacji, zaczyna się od litery lub znaku podkreślenia).
  Lista parametrów **formalnych** to zestaw par:
  Typ_parametru_formalnego, nazwa_parametru_formalnego
  (oddzielone są przecinkami).
- Jeżeli lista parametrów formalnych jest pusta – funkcje **bezparametrowe**
  to w nawiasach nie ma parametrów lub słowo **void**.
- Jeżeli funkcja nie zwraca wartości, typ funkcji zastępowany jest słowem kluczowym **void**.
- Każda funkcja przed jej zdefiniowaniem w języku C++ powinna być **zadeklarowana** przez podanie tzw. prototypu funkcji, czyli **nazwy funkcji wraz z typem oraz w nawiasach parametrów formalnych**
- **Prototyp** jest **konieczny**, gdy wywołanie następuje przed podaniem definicji,
  ale prototyp warto zawsze umieścić na początku programu, co pozwala na dowolną kolejność definicji różnych funkcji.

Przykłady deklaracji funkcji:

- **int SumaKwadratow(int n);**
  // funkcja podaje wartość int, jeden parametr formalny typu int
- **void Fun1(int m);** *// funkcja nie zwraca wartości, jeden parametr typu int*
- **Fun2(void);** *// lista parametrów jest pusta*
- **void Fun3(void0;** *// funkcja nie podaje wartości i lista parametrów jest pusta*

Definicja funkcji – opis funkcji

Typ_funkcji NazwaFunkcji (lista_parametrów_formalnych) {Instrukcje}

Inaczej

Typ nazwa (deklaracja parametrów) {Instrukcje}

- Opis – **definicja** funkcji składa się z linii nagłówkowej (deklaracyjnej) oraz bloku ograniczonego klamrami { }, zawierającego lokalne deklaracje oraz instrukcje.
- Instrukcja **return** pokazuje miejsce, gdzie funkcja kończy działanie.
- Jeśli instrukcja **return** nie występuje w treści funkcji, to funkcja kończy swe działanie po osiągnięciu nawiasu zamykającego }
- Jeśli funkcja zwraca wartość to występuje instrukcja **return** wartość; lub **return (wartość);**
- Dozwolone jest też wystąpienie kilku instrukcji **return** w funkcji.

Przykłady definicji funkcji

```
void funkcja_a(void)  // wydruk 20 kresek _
{
 int i;printf(„\n”);
for (i=1; i<20; i+=) printf(“_“);  // brak instrukcji return
}
int funkcja_d (int a, int b, float c)
#define CG 15.51
{ if  (c>CG) return (b+2);
else return(a-b);  // 2 instrukcje return
}
int SumaKwadratow(int n)
{ int i, suma=0;
for (i=1; i<=n; i++)  suma += i*i;
return suma; }
```

W języku C++ wewnątrz jednej funkcji nie można definiować innej funkcji.

Wszystkie funkcje są zewnętrzne w stosunku do pozostałych.

Wywołanie funkcji

- Aby wywołać daną funkcję, należy podać jej **nazwę** oraz w nawiasach **listę parametrów aktualnych**.
- **Parametry aktualne** odpowiadają **parametrom formalnym** umieszczonym w definicji funkcji
- Wartość parametrów aktualnych jest przekazywana do parametrów formalnych i na nich są wykonywane obliczenia.
- **zmienna = nazwa_funkcji lista_parametrów_aktualnych)**
  np. **n=10;** wynik = **sumaKwadratow(n);**
- **nazwa_funkcji (lista_parametrów_aktualnych)** Np.
  funkcja_a(); funkcja_b(10);

Wywołanie funkcji - opis

- **Wywołanie funkcji** ma postać nazwy funkcji z podaną w nawiasach okrągłych listą argumentów, oddzielonych przecinkami.
- Liczba argumentów wywołania musi być dokładnie taka sama jak liczba parametrów w deklaracji i definicji funkcji.
  Nawet, jeśli funkcja jest bezargumentowa, **nawiasy musimy** podać: w przeciwnym przypadku nazwa zostanie potraktowana nie jako wywołanie, ale jak nazwa zmiennej wskaźnikowej wskazującej funkcję.
- Rolę argumentów mogą pełnić dowolne **wyrażenia**, których wartość jest typu zgodnego z zadeklarowanym typem odpowiedniego parametru funkcji.
  Nie musi być to typ identyczny z typem parametru: *wywołanie będzie prawidłowe, jeśli wartość argumentu można niejawnie przekształcić na wartość zadeklarowanego typu parametru*.
  Jeśli przy takiej konwersji może być tracona informacja, (czyli jest to konwersja zawężająca, a nie promocja), to zazwyczaj podczas kompilacji otrzymamy ostrzeżenia.

Program – obliczenie sumy kwadratów podanej liczby

```
// Program f_sumkw.cpp - Suma kwadratow      C++
#include <iostream.h>
#include <conio.h>
int SumaKwadratow(int n);   // prototyp - deklaracja funkcji

int main()
{
  int n, wynik;
  cout << "Obliczenie sumy kwadratow: 1*1+2*2+...i*i+...n*n dla podanej zmiennej n"
      << endl;
  cout << "Podaj wartosc zmiennej n ==> ";  cin >> n;
  wynik = SumaKwadratow(n);  // wywolanie funkcji
  cout << "Suma kwadratow dla n = " << n << "  wynosi " << wynik << endl;
  getch();
  return 0;
}

int SumaKwadratow (int n)    // definicja funkcji
{    int i, suma=0;
 for (i=1; i<=n; i++)    suma += i*i;    // suma = suma + i*i;
 return (suma);
 }
```

Wykorzystanie funkcji w różnych programach

Zaprojektowanie funkcji **SumaPoteg(),** której celem jest obliczenie wartości wyrażenia: **1k + 2k + … + nk**

```
// kwadrat1.cpp
#include <cstdlib> #include <iostream> #include <conio.h>
#include <math.h>>
using namespace std;
int SumaKwadratow(int n);
int SumaPoteg(int n, int k);
int main()
{  int n, k, wynik;
cout << "Podaj wartosc zmiennej n " ;
cin >> n;
wynik = SumaKwadratow(n);
cout << "Suma kwadratow dla n = " << n << "  wynosi "
<< wynik << endl;
cout << "Podaj wartosc zmiennej k " ;
cin >> k;
wynik=SumaPoteg(n, k);
cout << "Suma poteg " << k << " stopnia dla n = " << n
<< " wynosi " << wynik;
getch();          return 0;
}
```

```
/* definicja funkcji */
int SumaKwadratow(int n)
{
  int i, suma = 0;
  for (i=1; i<=n; i++)
   suma+=i*i;
   return (suma);
}

int SumaPoteg(int n, int k)
{
              int i, suma=0;
              for (i=1; i<=n; i++)
               suma += pow(i,k);
// pow - funkcja standardowa
     return (suma);
}
```

Wersja programu. w której jest **funkcja zdefiniowana w innym pliku**

```
// Plik kwadraty.cpp
#include <iostream.h> #include <conio.h>
#include "funkcje.h„  // Plik zewnętrzny
int SumaKwadratow(int n);
int main()
{
int n, k, wynik, wynik2;
cout << "Podaj wartosc zmiennej n do obliczenia
sumy kwadrarow: 1^2+2^2+...+n^2 ==> " ;  cin >>
n;
wynik = SumaKwadratow(n);
cout << "Suma kwadratow dla n = " << n << "
wynosi " << wynik << endl;
 cout << "Obliczenie wartości wyrażenia: 1^k + 2^k
+... n^k " << endl;
cout t << "Podaj wartość zmiennej k ==> " ; cin >>
k;
wynik2=SumaPoteg(n, k);
cout << "Suma poteg " << k << " stopnia dla n = "
<< n << " wynosi " << wynik2;          getch();
return 0; }
int SumaKwadratow(int n)
{    int i, suma = 0;
  for (i=1; i<=n; i++)   suma+=i*i;
 return (suma);
}
```

```
// Plik zewnętrzny funkcje.h
#include <math.h>
int SumaPoteg(int n, int k);


int SumaPoteg(int n, int k)
{
int i, suma=0;
for (i=1; i<=n; i++)
 suma += pow(i,k);
// pow - funkcja standardowa
 return (suma);
}
```

Program do obliczenia pola figur - funkcje

```
// Polafig1a.cpp  - pola figur - funkcje,  Dev C++
#include <cstdlib> #include <iostream>
#include <conio.h>  #include <math.h>
using namespace std;
// Deklaracje funkcji
float poleprostokata(); float poletrojkata();
float polekola();
// Funkcja glowna
int main ()
{  float x, y, pole;   int decyzja;
  cout << "Obliczenie pole figur - funkcje " << endl;
  cout << "\nPodaj nr figury lub 0 - koniec obliczen: \n " ;
  cout << "0 - wyjście z programu" << endl;
  cout << "1 - prostokat: a b " << endl;
  cout << "2 - trojkat: a h" << endl;
  cout << "3 - kolo: r " << endl;
  do  cin >> decyzja;
  while (!((decyzja >=0) && (decyzja<=3)));

  switch (decyzja)
    case 1: pole = poleprostokata();  break;
     case 2:  pole = poletrojkata();     break;
     case 3: pole = polekola();          break;
    default:  cout << "Koniec programu" << endl;  pole = 0;
    }

   cout << endl;     getch();     return 0;
}
```

```
// Definicje funkcji
float poleprostokata()
{ float a, b, p;
 cout << "Pole prostokata:  ";  cout << "podaj boki a b
=> ";  cin >> a >> b;
 p=a*b;  cout << "Pole prostokata wynosi: " << p <<
endl;
 return(p);
}

float poletrojkata()
{  float a, h, p;
 cout << "Pole trojkata: ";  cout << "podaj bok a  i
wysokosc h => ";
 cin >> a >> h;  p=0.5*a*h;  cout << "Pole trojkata
wynosi: " << p << endl;
 return(p);
}

float polekola()
{  float r, p;
 cout << "Pole kola:    ";  cout << "podaj promien => ";
cin >> r;
 p=M_PI*r*r;  cout << "M_PI=" << M_PI << "  ";
 cout << "Pole kola wynosi: " << p << endl;
 return(p);
}
```

```c
/* pola_1.cpp  pola figur: do, swich, funkcje – wielokrotne oblicz. */
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define WYS2 puts("\n\n")
// Deklaracje funkcji
void poleprost();
void poletrojk();
void poletrapez();
double polekola();

int main (int argc, char *argv[])
{
 int wybor;
 float p;

do
{
  putchar('\n');    puts("Program do obliczenia pol figur");
   puts("0. Koniec programu");    puts("1. Pole prostokata");
   puts("2. Pole trojkata");   puts("3. Pole trapezu");
   puts("4. Pole kola"); puts("5. Zakoncz program");
   wybor=getchar();

switch (wybor)
 {
 case '1': WYS2; poleprost(); WYS2; fflush(stdin);  break;
 case '2': WYS2; poletrojk(); WYS2; fflush(stdin);  break;
 case '3': WYS2; poletrapez(); WYS2; fflush(stdin); break;
 case '4': WYS2; p=polekola();
 printf("Pole kola = %lf", p); WYS2; fflush(stdin); break;
 case '5': case '0':  break;
 default: puts("Niepoprawny wybor, powtorz\n");
  }
} while ( (wybor != '5') && (wybor != '0'));

  // getch();
   return 0;
}

// Definicje funkcji
void poleprost()
{
 float a, b, p;
 puts("Obliczenie pola prostokata");
 printf("Podaj dlugosci bokow: a i b => "); scanf("%f %f", &a, &b);
 p=a*b;
 printf("Pole prostokata o bokach %f i %f = %f",a,b,p);
}

void poletrojk()
{
 float a, h, p;
 puts("Obliczenie pola trojkata");
 printf("Podaj dlugosci: a i h => "); scanf("%f %f", &a, &h);
 p=a*h/2.0;
 printf("Pole trojkata o boku %.3f i  wysokosci %.3f = %.3f",a,h,p);
 }

void poletrapez()
{
 float a, b, h, p;
 puts("Obliczenie pola trapezu");
 printf("Podaj dlugosci: a i b oraz h => "); scanf("%f %f %f", &a, &b, &h);
 p=0.5*(a+b)*h;
 printf("Pole trapezu o boku %.3f i  wysokosci %.3f = %5.3f",a,h,p);
 }

double polekola() // zwraca watosc
{
 double r, p, p1;
 double PI=4.0*atan(1.0),  pi = 4.0 * atan(1.0);
 puts("Obliczenie pola kola\n");
 printf("M_PI %lf  PI= %lf  pi = %lf",M_PI, PI, pi);
 printf("\nPodaj promien r  => ");
 scanf("%lf", &r);
 p=M_PI*r*r;
 p1=PI*r*r;
 printf("\nPole kola o promieniu %lf = M_PI*r*r= %lf",p);
 printf("\nPole kola o promieniu %lf = PI*r*r  = %lf",p1);
 printf("\n");
 return(p);
}
```

Widoczność zmiennych – zmienne lokalne

- W każdej funkcji prawie zawsze występują **zmienne**.
- Pojawia się problem zmiennych o tych samych nazwach, z których korzystają różne funkcje. Należy znać zasadę widoczności i dostępności zmiennych w programie.
- Zmienne **lokalne**: Zmienne zadeklarowane wewnątrz danego bloku instrukcji widoczne są tylko w tym bloku.
- **Przykład 1**: zmienna i zadeklarowana w pętli for – poza nią nie jest znana

```
include <cstdlib>        #include <iostream>   using namespace std;
int main(int argc, char *argv[]) {    for (int i = 0; i < 5; i++)       {       cout << i << endl;    }
    cout << i << endl;   // zmienna i nie jest znana – jest poza blokiem  deklaracji – program się nie skompiluje
    system("PAUSE");    return EXIT_SUCCESS; }
```

- **Przykład 2** – zmienna i = 5 wewnątrz funkcji main o wartosci 5, zmienna i w bloku – wartość 10 i ponowne wyświetlenie wartości i – równe 5.

```
#include <cstdlib>       #include <iostream>   using namespace std;
int main(int argc, char *argv[]) {      int i = 5;     cout << i << " - na poczatku;  ";
  {  int i = 10;  cout << i << " - w bloku instrukcji;  ";    }     cout << i << " - na koncu  "  << endl;
   system("PAUSE");    return EXIT_SUCCESS; }
```

Wynik: **5** - na początku; **10** - w bloku instrukcji; **5** - na końcu

Zmienne lokalne i globalne

- Zmienna dostępna tylko w określonym fragmencie kodu programu (np. wewnątrz pojedynczego bloku instrukcji, w funkcji) i tylko w nim dostępna) nazywana jest zmienną **lokalną**.
- Jeśli zmienna została zadeklarowana poza funkcją główną **main()** oraz poza wszystkimi funkcjami, widoczna jest w całym programie i nazywana jest zmienną **globalną**.

Przykładowy program ze zmienną globalną liczba

```
// zmglob.cpp - zmienna globalne i jej wykorzystanie w funkcji zamiast argumentów
#include <cstdlib>          #include <iostream>        using namespace std;
int liczba = 10;                // zmienna globalna
 int test();
int main (int argc, char *argv[]) {
   cout <<  "1) Wartość zmiennej liczba wewnątrz main(): " << liczba << endl;                    liczba = 40;
   cout <<  "2) Nowa wartość zmiennej liczba wewnątrz main: " << liczba << endl;
   int liczba2 = 0;                  liczba2 = test();
   cout <<  "4) Wartość przypisana do zmiennej liczba2: " << liczba2 << endl;
    system("PAUSE");     return EXIT_SUCCESS; }
int test() {       cout <<  " (3) Wartosc zmiennej liczba wewnatrz funkcji test: " << liczba << endl;
   return ++liczba; }
```

Wyniki:

1) Wartość zmiennej liczba wewnątrz main: **10**

2) Nowa wartość zmiennej liczba wewnątrz main: **40**

(3) Wartość zmiennej liczba wewnątrz funkcji test: **40**

4) Wartość przypisana do zmiennej liczba2: **41**

- Należy ograniczać korzystanie z funkcji globalnych do minimum – do takich sytuacji, gdzie zmienna globalna jest wykorzystywana przez większość funkcji a jej wartość nie jest zmieniana.
  Np. pobranie wartości od użytkownika i korzystanie z niej w różnych funkcjach programu.

Przekazywanie parametrów funkcji

- Podczas wywołania **funkcji** do parametrów formalnych są przekazywane wartości parametrów aktualnych. Takie przekazywanie nazywa się przekazywaniem przez **wartość**.
- Wyjątkiem są **tablice**, do których przekazywany jest adres pierwszego elementu tablicy.
- Przekazywanie wartości parametrów aktualnych do parametrów formalnych oznacza, że po zakończeniu wykonywania funkcji, wartość parametru aktualnego nie zmienia się, nawet, gdy wartość parametru formalnego ulegnie zmianie wskutek operacji wykonywanych przez funkcję.

Przykład – funkcja zwieksz() z przekazaniem parametrów przez wartość nie powoduje zmiany parametrów aktualnych – zmieniają się tylko wartości parametrów w ramach funkcji.

```
// zwieksz1.cpp
#include <cstdlib>  #include <iostream> #include <conio.h>
using namespace std;
void zwieksz(int a, int b)
{    a = a+100;      b = b+200;
cout << "2) Funkcja zwieksz(a, b)  a=x+100= " << a << "  b=y+200=" << b <<
endl;
}
int main()
{
    int x=10, y=10;
    cout << "1) Funkcja glowna:  x=" << x << "  y=" << y << endl;
    zwieksz(x,y);
    cout << "3) Funkcja glowna: x=" << x << "  y=" << y << endl;
    getch();
}
```

Zmiana wartości zmiennych (parametrów aktualnych) przez **użycie wskaźników.**
Parametrami aktualnymi są tu adresy zmiennych x i y.
Funkcja powoduje zmianę wartości zmiennych x i y.

```
// zwieksz2.cpp - zwiększenie wartości parametrów aktualnych przez użycie wskaźników
#include <cstdlib>          #include <iostream>          #include <conio.h>
using namespace std;
void zwieksz(int *a, int *b)
{
*a = *a+100;        *b = *b+200;
cout << "2) Funkcja zwieksz(a, b)  *a=x+100= " << *a << "  *b=y+200=" << *b <<
endl;
}
int main()
{  int x=10, y=10;
cout << "Zmiana wartości parametrów aktualnych przez użycie wskaźników" <<
endl;
cout << "1) Funkcja główna:  x=" << x << "  y=" << y << endl;
zwieksz(&x,&y);
cout << "3) Funkcja główna: x=" << x << "  y=" << y << endl; getch();
}
```

Zmiana wartości zmiennych przez referencję

```
// zwieksz3.cpp - zwiększenie wartości parametrów aktualnych przez referencję
#include <cstdlib>  #include <iostream>          #include <conio.h>
using namespace std;
void ZwiekszRef(int &a, int &b)
{
    a +=100;
    b +=200;
    cout << "2) Funkcja zwieksz(a, b)  a=x+100= " << a << "  b=y+200=" << b << endl;
}
int main()
{
    int x=10, y=10;
    cout << "Zmiana wartości parametrów aktualnych przez referencje" << endl;
    cout << "1) Funkcja główna:  x=" << x << "  y=" << y << endl;
    ZwiekszRef(x,y);
    cout << "3) Funkcja główna: x=" << x << "  y=" << y << endl;
    getch();
}
```

Praktyczny przykład wymagający zmiany wartości parametrów aktualnych – **zamiana wartości 2 zmiennych:**

```
// 1) zamianaR.cpp – zamiana wartości parametrów aktualnych przez referencję
#include <cstdlib>   #include <iostream> #include <conio.h>
using namespace std;
void ZmienRef(int &a, int &b)
{    int pom;
     cout << "2) Funkcja - przed zamiana: a=x=" << a << "  b=y=" << b << endl;
     pom =b;
     b=a;
     a=pom;
     cout << "3) Funkcja po zamianie a= " << a << "  b=" << b << endl;
}
int main()
{    int x=10, y=30;
     cout << "Zmiana wartości zmiennych przez referencje" << endl;
     cout << "1) Funkcja główna przed zamiana:  x=" << x << "  y=" << y << endl;
     ZmienRef (x, y);
     cout << "4) Funkcja glowna: po zamianie: x=" << x << "  y=" << y << endl;  getch();
}
```

Zamiana wartości zmiennych - parametrów aktualnych przez wskaźnik

```
// 2) zamianaW.cpp - zamiana wartości parametrów aktualnych przez wskaźnik
#include <cstdlib>    #include <iostream>          #include <conio.h>
using namespace std;
void ZmienWsk (int *a, int *b)
{
     int pom;
     cout << "2) Funkcja - przed zamiana: a=x=" << *a << "  b=y=" << *b << endl;
     pom =*b;          *b=*a;     *a=pom;
     cout << "3) Funkcja po zamianie a= " << *a << "  b=" << *b << endl;
}
int main()
{    int x=10, y=30;
     cout << "Zmiana wartości zmiennych przez wskaznik" << endl;
     cout << "1) Funkcja glowna przed zamiana:  x=" << x << "  y=" << y << endl;
     ZmienWsk (&x, &y);
     cout << "4) Funkcja główna: po zamianie: x=" << x << "  y=" << y << endl;
     getch();
}
```

Podsumowanie – przekazywanie parametrów do funkcji

- Domyślnym sposobem przekazywania parametrów do funkcji jest przekazywanie przez wartość.
  Nie trzeba się wtedy martwić czy przypadkiem funkcja nie zmieni wartości parametru.
- W przypadku potrzeby zmiany parametru aktualnego należy wykorzystać wskaźniki lub zastosować referencję.

Argumenty funkcji głównej main()

- **int main (int argc, char *argv[]) { … return 0; }**
- Funkcja main akceptuje dwie zmienne:
  - int **argc** - liczbę całkowitą pokazującą liczbę argumentów w wierszu poleceń przy wywoływaniu programu (łącznie z nazwą programu),
  - char ***argv[ ]** - wskaźnik do tablicy ciągów znakowych, zawierających argumenty z wiersza poleceń;
    alternatywnie ***argv[]** może być zadeklarowany jako ****argv**, jeśli programiście ta postać bardziej odpowiada.
- Przykład

  Jeśli program został wywołany z systemu operacyjnego za pomocą następującego wiersza poleceń:

  program1 abc def 6

  to **argc** wynosi 4,

  argv:

  arg[0] = "program1" arg[1] = "abc"

  arg[2] = "def" arg[3] = "6"

Argumenty funkcji głównej – program przykładowy

//argfglow.cpp - argumenty funkcji głównej

```
#include <cstdlib>   #include <iostream>   using namespace std;
int main(int argc, char *argv[])
{   cout << "Argumenty funkcji glownej\n";      cout << "Liczba argumentow: " << argc << endl;
   int i = 0;
   for (int i=0; i < argc; i++)
   {       cout << i << "-argument -> " << argv[i] << endl;    }
   cout << endl;        system("PAUSE");    return EXIT_SUCCESS;  }
```

Po uruchomieniu skompilowanego programu z parametrami 1, 2, 3, 4 i skierowaniu do pliku

*Np.. **Argflow.exe 1 2 3** > w1.txt otrzymamy plik wyników:*

Argumenty funkcji głównej

Liczba argumentów: 4

0-argument -> **argfglow.exe**

1-argument -> **1**

2-argument -> **2**

3-argument -> **3**

Aby kontynuować, naci□śnij dowolny klawisz . . .

Argument 0 (zerowy) to nazwa programu, pozostałe:1, 2, 3 – kolejne argumenty wywołania

Funkcje matematyczne

Standardowe funkcje matematyczne w C

abs	wartość bezwzględna
acos	arcus cosinus
asin	arcus sinus
atan	arcus tangens
atan2	arcus tangens, z użyciem znaków argumentów w celu określenia ćwiartki
ceil	najmniejsza liczba całkowita, nie mniejsza od podanej
cos	cosinus
cosh	cosinus hiperboliczny
div	zwraca iloraz i resztę z dzielenia
exp	zwraca *e* podniesione do podanej potęgi
fabs	wartość bezwzględna liczby zmiennoprzecinkowej
floor	największa liczba całkowita, nie większa od podanej
fmod	zwraca resztę z dzielenia dwóch liczb
frexp	zwraca podaną liczbę w postaci mantysy i wykładnika liczby *2*
labs	wartość bezwzględna liczby typu *long* (*long integer*)
ldexp	oblicza $num*2^{exp}$
ldiv	zwraca iloraz i resztę z dzielenia dla liczb typu *long* (*long integer*)
log	logarytm naturalny (o podstawie *e*)
log10	logarytm dziesiętny
modf	zwraca część całkowitą i ułamkową podanej liczby
pow	zwraca wartość pierwszej liczby podniesionej do potęgi drugiej liczby
sin	sinus
sinh	sinus hiperboliczny
sqrt	pierwiastek kwadratowy
tan	tangens
tanh	tangens hiperboliczny

Program do obliczenia azymutu z wykorzystaniem funkcji atan2(y,x)

```
/* azatan2. - obliczenie azymutu - while */
#include <stdio.h>          #include <math.h>          #include <conio.h>
#define PI 3.14159265
int main ()
{
  double x, y, result, az, d;  int kont=1;
  puts("Obliczenie azymutu na podstawie funkcji atan2(dy,dx)");                    puts("tan(Az)=DY/DX");
  printf("\nWprowadz DX (999 - koniec) =>  ");          scanf("%lf", &x);

  while (x != 999)
  {
        printf("Wprowadz DY  ");              scanf("%lf", &y);
         printf("x=%f  y=%f\n",x,y);          result = atan2 (y,x) * 200.0 / PI;
         az=result;    if (result <0) az=result+400.0;
         d=sqrt(x*x+y*y);
         printf ("atan2 dla (DX=%lf, DY=%lf) = %lf[grad]\n", x, y, result);
         printf ("DX=%.3f DY=%.3f  Az = %.4f[grad]  Dl=%.3f\n", x, y, az, d  );
         printf("\nWprowadz DX (999 - koniec) => ");
         scanf("%lf", &x);
  }    // while

  return 0;
}
```

Program do obliczenia azymutu z wykorzystaniem funkcji atan2(y,x)

```
/* az_atan2 - obliczenie azymutu  - instrukcja do…while */
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define PI 3.14159265
int main ()
{   double x, y, result, az, d;   int kont=1;
  puts("Obliczenie azymutu na podstawie funkcji atan2(dy,dx)");   puts("tan(Az)=DY/DX");
  do  {
  printf("\nWprowadz DX  ");   scanf("%lf", &x);
  printf("Wprowadz DY  ");   scanf("%lf", &y);
  printf("x=%f  y=%f\n",x,y);   result = atan2 (y,x) * 200.0 / PI;   az=result;
  if (result <0) az=result+400.0;
  d=sqrt(x*x+y*y);   printf ("atan2 dla (DX=%lf, DY=%lf) = %lf[grad]\n", x, y, result);
  printf ("DX=%.3f DY=%.3f  Az = %.4f[grad]  Dl=%.3f\n", x, y, az, d  );
  printf("\n 1- obliczenia, 0 - koniec ");
  scanf("%d",&kont);
  } while (kont != 0);
    return 0;
}
```

Funkcje matematyczne

Funkcje matematyczne znajdują się w bibliotece math.h, należy na początku kodu programu dopisać dyrektywę #include <cmath.h>

Niektóre przydatne funkcje matematyczne:

- potęgowanie: **pow(x, y);**
- Pierwiastek kwadratowy: **sqrt(x);**
- Liczba e do potęgi x: **exp(x);**
- Logarytm naturalny: **log(x);**
- Logarytm dziesiętny: **log10(x);**
- funkcje trygonometryczne – argument w radianach Kat[rad]=Kat[grad]*PI/200.0; PI=**M_P**I lub PI=3.1415926;
  - **sin(x_rad);**
  - **cos(x_rad);**
  - **tan(x_rad);**
  - **atan(x)**
  - **atan2(y,x);** - w przypadku wyniku ujemnego należy dodać kąt 2*PI a następnie zamienić ewentualnie radiany na stopnie lub grady:
  - Kat[st] = Kat[rad]*180.0/PI; Kat[grad] = Kat[rad]*200.0/PI;

```
// Program funmat1.cpp - funkcje matematyczne
#include <cstdlib>                  #include <iostream>              #include <math.h>                  using namespace std;
const double PI=M_PI, ROG=200.0/M_PI, ROS=180.0/M_PI;
double potega(double x, double y);  double az_atan2(double dy, double dx); double az(double dx, double dy);
int main()  {
   double a, b, c,  ag, as, ar, ad, bd;      cout << "Funkcje matematyczne "  << endl;
   cout << "Podaj 2 liczby (calkowite lub rzeczywiste z kropka dzies.) a i b \n";      cout << "a => " ; cin >> a;     cout << "b => " ; cin >> b;
   ad=fabs(a); bd=fabs(b);      cout << "|a|= fabs(a) = " << ad << "  |b| = fabs(b) = " << bd << endl;
   cout << a << " + " << b << " = " << a+b << endl;     cout << a << " - " << b << " = " << a-b << endl;
   cout << a << " * " << b << " = " << a*b << endl;     cout << a << " / " << b << " = " << a/b << endl;
   cout << int(a) << " % " << int(b) << " = " << (int(a) % int(b))<< endl;
   cout << "Potega: a^b = pow(a,b) = " << a << "^" << b << " = " << pow(a,b) << " = " << potega(a,b) << endl;
   cout << "Pierwiastek(fabs(a)) = " << sqrt(fabs(a)) << "  Pierwiastek(fabs(b)) = " << sqrt(fabs(b)) << endl;
   cout << "|a| = " << fabs(a) << "  ln|a| = log|a| = " << log(ad) << "  log10|a| = " << log10(ad) << endl;
   cout << "|b| = " << fabs(b) << "  ln|b| = log|b| = " << log(bd) << "  log10|b| = " << log10(bd) << endl;
   cout << "atan2(b,a) [rad] = " << atan2(b,a) << "  atan2(a,b) [rad] = " << atan2(a,b) << endl;
   cout << "az_atan2(a,b)[grad]=" << az_atan2(a,b) << "  az_atan2(b,a)[grad]= " << az_atan2(b,a) << endl;
   cout << "azymut(a,b)=" << az(a,b) << "  azymut(b,a) " << az(b,a) << endl;     cout << endl;
   cout << "Podaj kat w [grad] =>  "; cin >> ag;     ar=ag/ROG; as=ar*ROS;
   cout << "Kat[rad] = " << ar << "  Kat[stop] = " << as << endl;      cout << "sin = " << sin(ar) << "   cos = " << cos(ar) << "   tan = " << tan(ar) << "  ctg
= " << 1/tan(ar) << endl;
   cout << "Podaj kat w [stop] =>  "; cin >> as;     ar=as/ROS; ag=ar*ROG;     cout << "Kat[rad] = " << ar << "  Kat[grad] = " << ag << endl;
   cout << "sin = " << sin(ar) << "   cos = " << cos(ar) << "   tan = " << tan(ar) << "  ctg = " << 1/tan(ar) << endl;
   cout << "Podaj kat w [rad] =>  "; cin >> ar;     ag=ar*ROG; as=ar*ROS;     cout << "Kat[grad] = " << ag << "  Kat[stop] = " << as << endl;
   cout << "sin = " << sin(ar) << "   cos = " << cos(ar) << "   tan = " << tan(ar) << "  ctg = " << 1/tan(ar) << endl;
   cout << endl;     cout << endl;     system("PAUSE");     return EXIT_SUCCESS; }

double potega (double x, double y) {                  return (pow(x,y));  }

double az_atan2 (double dx, double dy) {   double a;   a=atan2(dy,dx);   a=a*ROG;
  if (a <0) a+=400.0;   return a; }

double az (double dx, double dy)
{  double a, pi, rg, rs;
 pi = 4.0 * atan(1.0);  rg = 200.0 / pi;  rs = 180.0 / pi;
  if (dx==0)  { //  IF dx==0
  if (dy>0) a = pi / 2;    else   a = 1.5 * pi;   } //  IF dx==0
  else     {  // if (dx != 0)
     a = atan(dy / dx);                  if (dx < 0)  a = a + pi;
                        else // dx >=0
          {  if (dy < 0)   a = a + 2 * pi;  }      } // if (dx != 0)
   a=a*rg ;    return a; }
```

Wskaźniki do funkcji

- W C można nie tylko podstawić daną w miejsce zmiennej
  (jak w każdym języku programowania)
  ale także można podstawić w miejsce funkcji wywoływanej w programie, inną **funkcję**, która jest potrzebna (np. sin)
- Aby wskazać funkcję stosujemy wskaźnik do funkcji.

Np. deklaracja

double (*funkcja) (double);

należy rozumieć następująco:

  - „Za pomocą wskaźnika do funkcji ***funkcja** można wskazać takie funkcje, które:
  - pobierają jeden argument typu double **float**;
  - zwracają do programu wartość typu double **float**”

- Dostępne są standardowe funkcje matematyczne zdefiniowane w pliku **math.h** oraz inne funkcje własne.

```
// Program wskfunk.c - wskaźniki do funkcji
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <string.h>
#define ROG (200.0/M_PI)
#define NIESK 9E9
// Deklarcje funkcji wlasnych
double Odwrotn( double );  // <-- Deklaracja zwyklej funkcji
double ctg(double a);          // <-- Deklaracja zwyklej funkcji
// -- Wskaznik do funkcji ---
double (*Funkcja)(double ARG);
// Deklaracje zmiennych
double Liczba, Wynik;
int WYBOR;
char nazwa[10];
main()          // funkcja glowna
{
 puts("Wskazniki do funkcji");
 do  {
  printf("\nMENU\n");
  printf("0 - Koniec obliczen \n");
  printf("1 - Odwrotnosc 1/x - funkcja wlasna\n");
  printf("2 - Pierwiastek - z math.h \n");
  printf("3 - Sinus - z math.h\n");
  printf("4 - Cosinus - z math.h \n");
  printf("5 - Tangens - z math.h\n");
  printf("6 - Cotangens - funkcja z tan\n");
  scanf("%d", &WYBOR);
  if (WYBOR != 0)   {printf("Podaj Liczbe: ");   scanf("%lf",
&Liczba);   }
  else { printf("Koniec. Nacisnij Enter! ");  break;};
 switch(WYBOR)
    {
    case 0: printf("Koniec \n"); break;
    case 1: Funkcja=Odwrotn; strcpy(nazwa,"(1/x)
"); break;
    case 2: Funkcja=sqrt; strcpy(nazwa,"sqrt");
break;
    case 3: Funkcja=sin; Liczba=Liczba/ROG;
strcpy(nazwa,"sin"); break;
    case 4: Funkcja=cos; Liczba=Liczba/ROG;
strcpy(nazwa,"cos"); break;
    case 5: Funkcja=tan; Liczba/=ROG;
strcpy(nazwa,"tan"); break;
    case 6: Funkcja=ctg; Liczba/=ROG;
strcpy(nazwa,"ctg"); break;
    default: printf("Zly wybor! Powtorz. "); continue;
  } // do
  Wynik=Funkcja(Liczba);       // Wywolanie
wybranej funkcji
  printf("WYNIK funkcji %s(%lf) = %lf\n", nazwa,
Liczba, Wynik);
} while (WYBOR != 0);
  getch();   return 0;
}
// definicje funkcji wlasnych
double Odwrotn(double a)
{   printf("Odwrotnosc liczby: ");
  if (a!=0) a=1/a; else printf("Nieskonczonosc\n");
return a;}
double ctg(double a)
{  a              = tan(a);   if (a!=0) a=1/a;
    else   {  a=NIESK;          printf("ctg =
nieskonczonosc\n");          }   return a;}
```