❽

```
#include <iostream.h>
// rozpoznawanie cyfr
main()
{
int liczba;

      cout << "Podaj liczbe: ";
      cin >> liczba;
      switch (liczba)
      {
      case 1:
            cout << "jeden" << endl;
            break;
      case 2:
            cout << "dwa" << endl;
            // tu nie ma break
      case 3:
            cout << "trzy" << endl;
            break;
      default:
            cout << "nie znam" << endl;
            break;
      }
}
```

Po uruchomieniu dla liczby 2 widać co się dzieje. Mianowicie program wykonuje wszystkie operacje do napotkania pierwszego break.Jeżeliby nie było break przy case „1" to wtedy na ekran drukowałoby jeden, dwa, trzy ... i dopiero tutaj napotkałoby break.

---

❾

```
#include <iostream.h>
int k=33;   // zmienna globalna
main()
{
      cout << "main (przed blokiem): k=" << k << endl;
      {     //poczatek bloku: zakres lokalny
            int k=1;
            cout << "blok: k=" << k << endl;
            cout << "dostep do zmiennej globalnej k=" <<
                  ::k << endl;
      }     //koniec bloku
      cout << "main (za blokiem): k=" << k << endl;
}
```

To raczej się nie przyda – zakres ważności zmiennych i odwoływanie się do zmiennej globalnej, z wewnątrz bloku w którym zmienna ma nadaną inną wartość lokalną. Gdyby globalne k nie było „przesłonięte" przez to wewnętrzne k, wtedy rzecz jasna cout << k by wyświetlało to jedno jedyne globalne.

---

❶❷

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

```
main()
{
char znak;

      for (znak='a'; znak<='z'; znak++)
            cout << znak ;
      cout << endl ;
}
```

Jak to działa że a+1=b ? Opiera się to na tablicy znaków ASCII – mianowicie liczbom od 0 do chyba 255 przypisane są znaki char. Można popróbować w „Start”->”Uruchom”->”cmd”. Trzymając lewy alt (przy włączonej klawiaturze numerycznej) wklepać jakąś liczbę i powinien się jakiś znak pojawić. I właśnie to „a” jest interpretowane jako liczba – numer znaku ASCII, a „b” ma przypisany muner o jeden większy. Podobnie zamienia się małe litery na duże – odejmując od każdej małej litery 32 uzyskuje się jej „dużą” postać.

---

❶❸

```
#include <iostream.h>
main()
{
unsigned long silnia, liczba, licz;

      cout << "Podaj liczbe: ";
      cin >> liczba;
      silnia=1;
      licz=2;
      if (liczba)
            while (licz<=liczba)   //for ( ; licz<=liczba; )
            {
                  silnia *= licz;
                  licz ++;
            }
      cout << liczba << "! =" << silnia << endl;
}
```

if (liczba) – jeżeli warunek jest prawdziwy to zawartość jest wykonywana. Prawda oznacza wartość różną od zera. Więc jeżeli liczba != 0 to wtedy wykonywana jest instrukcja licząca silnię. Jeżeli liczba jest zerem to zgodnie z definicją silni drukowana jest na końcu wartość początkowa – 1.

---

❶❹

```
#include <iostream.h>
main()
{
int liczba;

      do
      {
            cout << "Podaj liczbe: ";
            cin >> liczba;
            if (!liczba) cout << "zero - koniec petli\n";
            else cout << "To nie jest zero\n";
      } while (liczba);
}
```

Podobnie jak w poprzednim – tylko na odwrót. „!” to negacja. Więc warunek jest prawdziwy jeżeli liczba jest zerem.

Zasadnicza różnica pomiędzy do...while a while {}

W pierszym przypadku instrukcja wykonywana jest przynajmniej jeden raz i dopiero zostaje sprawdzony warunek. W drugim przypadku instrukcja wykonana jest tylko przy spełnionym warunku.

---

❶❼

To zadanie (którego tutaj nie ma) było na zajęciach i jest jednym ze sztandarowych.
Czyli:
***zdefiniuj** funkcję potęgującą która jako parametr przyjmuje stopień potęgi oraz liczbę potęgowaną. Ma to być algorytm potęgowania a nie pow(x,y).*
***zdefiniuj** funkcję która wyświetli komunikat o zakończniu pracy programu*
***w programie main()** niech liczba będzie wprowadzana z klawiatury a funkcja potęgująca wywołana dla 2 i 3 potęgi*

❶❽

Różnica pomiędzy przekazywaniem argumentów przez wartość i przez referencję. Omówione na stronie 83 (według numeracji stron) Tomu pierwszego Symfonii.

```
#include <iostream.h>
//---------------------------
void zeruj(int a, int &b)
{
      cout << "-------------\n";
      cout << "Funkcja\n";
      cout << "a=" << a << "\tb=" << b << endl;
      a=0; b=0;
      cout << "Po wyzerowaniu:\n";
      cout << "a=" << a << "\tb=" << b << endl;
      cout << "-------------\n";
      return;
}
//---------------------------
main()
{
int A=7, B=77;
      cout << "Main\n";
      cout << "A=" << A << "\tB=" << B << endl;
      zeruj(A,B);
      cout << "Main\n";
      cout << "Po wywolaniu funkcji:\n";
      cout << "A=" << A << "\tB=" << B << endl;
}
```

---

❷❿

Początek tablic – Jacko H. mówił że zabawa we wpisywanie i wypisywanie z tablic na pewno pokaże się na egzamie.

```
#include <iostream.h>
main()
{
int t[4];
      for (int i=0; i<4; i++)
            t[i]=2*i;
      for (i=0; i<4; i++)
            cout << "t[" << i << "]=" << t[i] << endl;
}
```

==========

❷❶

```
#include <iostream.h>
void dodaj2(int tablica[], int ile);
//----------------------------
main()
{
const int rozmiar=4;
int tab[rozmiar];       // można wykorzystac zmienna rozmiar
                        // gdyz jest to zmienna z modyfikatorem const
int i;
      for (i=0; i<rozmiar; i++)
            tab[i]=i;
      dodaj2(tab,rozmiar);

      for (i=0; i<rozmiar; i++)
            cout << "tab[" << i << "]=" << tab[i] << endl;
}
//-------------------------------
void dodaj2(int tablica[], int ile)
{
int i;
      for (i=0; i<ile; i++)
            tablica[i] += 2;
}
```

Ważny jest komentarz od Karbowksiego (ten wyżej). Normalnie rozmiar tablicy może być określony zmienną tylko przy dynamicznej alokacji tablic – z tym jednym wyjątkiem. Jako że „stała zmienna” nie może ulec modyfikacji to może określać rozmiar tablicy ;-)

==========

❷❸

WAŻNY PROGRAM !!!!

Prawdopodobna treść zadania:

Zdefiniuj funkcję wyświetlającą zawartość tablicy
Zdefiniuj funkcję kopiującą zawartość jednej tablicy do drugiej.

W funkcji main:

zainicjalizuj tablicę char jakimś słowem (tablica źródłowa)
wyświetl jej zawartość
zadeklaruj tablicę docelową o odpowienim rozmiarze
skopiuj zawatrtość (korzystając z finkcji oczywiście)
wyświetl zawartość tablicy docelowej

ofkors rozwiązanie nie jest jedno – można używać zarówno for i while – co kto lubi. Ale właśnie można się przyjrzeć tutejszemu forowi – Habelek wspominał że mogą być fory z nie określonymi wszystkimi trzema warunkami i tutaj mamy właśnie takiego – bez drugiego – czyli ograniczenia ilości powtórzeń.

```
#include <iostream.h>
void kopiuj(char cel[], char zrodlo[]);
void wypisz(char tab[]);
//----------------------------
main()
{
char sTab1[] = { "kot"};
char sTab2[10];
      cout << "sTab1=";
      wypisz(sTab1);
      kopiuj(sTab2,sTab1);
      cout << "sTab2=";
      wypisz(sTab2);
}
//-----------------------------
void kopiuj(char cel[], char zrodlo[])
{
int i;
      for (i=0; ; i++)
      {
            cel[i]=zrodlo[i];
            if (zrodlo[i]==NULL) break;
      }
}
//-----------------------------
void wypisz(char tab[])
{
int i;
      for (i=0; ; i++)
            if (tab[i]==NULL) break;
            else cout << tab[i];
      cout << endl;
}
```

==========

❷❹

Podobnie – tylko że używamy gotowej funkcji kopiującej stringi – strcpy(x,y) – kopiuje y do x.

Funkcja wypisująca znowu jest oparta na for z wyjściem warunkowym (jak to mniejwięcej ujął Habel). Bo za każdą iteracją (tak to nazywają) rozpatrywany jest jeszcze „if" i albo pętla się kończy „break" albo też wypisywana jest kolejna litrka i przechodzi do kolejnej iteracji ;-)

Jak dla mnie zdecydowanie lepiej wygląda i=0; while (tab[i]!=null){cout << tab[i];i++;}
Ale jak w poprzednim napisałem – pogłebia to znajomość fora ;-)

```
#include <iostream.h>
#include <string.h>
void wypisz(char tab[]);
//----------------------------
main()
{
char sTab1[] = { "kot"};
char sTab2[10];
      cout << "sTab1=";
      wypisz(sTab1);
      strcpy(sTab2,sTab1);
      cout << "sTab2=";
      wypisz(sTab2);
}
//-----------------------------
void wypisz(char tab[])
{
int i;
      for (i=0; ; i++)
            if (tab[i]==NULL) break;
            else cout << tab[i];
      cout << endl;
}
```

==========

❷❺

Ostra jazda ze wskaźnikami – ten program pozwala się nauczyć podstaw ich obsługi.
Przydatna może być też strona 153 z Symfonii.

```
#include <iostream.h>
//----------------------------
main()
{
int nLiczba=5;
int nLiczba2=10;
int *pInt; // p w nazwie bo “pointer” = wskaźnik
int *pInt2;
int *pInt3;
float Liczba3=1.2;
float *pFloat;
void *pWsk;

      cout << "nLiczba=" << nLiczba << endl; // pojawi sie 5
      pInt = &nLiczba;  // ustawienie wskaznika
      cout << "pInt=" << pInt << endl; // pojawi sie adres
      cout << "(*pInt)=" << *pInt << endl; // pojawi sie 5
      cout << "Przypisanie *pInt=6\n" ;
      *pInt = 6 ;
      cout << "(*pInt)=" << *pInt << endl; // pojawi sie 6
      cout << "nLiczba=" << nLiczba << endl; // pojawi sie 6
      cout << "zmiana obiektu wskazywanego\n" ;
      pInt = &nLiczba2;
      cout << "nLiczba2=" << nLiczba2 << endl; // pojawi sie 10
      cout << "(*pInt)=" << *pInt << endl; // pojawi sie 10
```

```
        // -------------------------------
        pFloat = &Liczba3;
        cout << "(*pFloat)=" << *pFloat << endl; // pojawi sie 1.2
        // -------------------------------
        // pFloat=pInt; // <<< - zostanie zasygnalizowany blad
        pWsk = pInt;
        cout << "pWsk=" << pWsk << endl; // pojawi sie adres
        // cout << *pWsk; // <<< - blad
        pInt2 = (int*)pWsk;  // musi byc rzutowanie
        cout << "(*pInt2)=" << *pInt2 << endl; // pojawi sie 10

        pFloat = (float*)pInt; // musi byc rzutowanie
        cout << "(*pFloat)=" << *pFloat << endl; // pojawi sie
                    // przypadkowa liczba mimo prawidlowego adresu
        pInt3 = (int*)pFloat; // musi byc rzutowanie
        cout << "(*pInt3)=" << *pInt3 << endl; // pojawi sie 10
    }
```

==========

❷❻

*arytmetyka wskaźników - *(wsk1+1) oznacza chwilowe przesunięcie wskaźnika do kolejnego elementu tablicy. On cały czas jest przypisany do jej zerowego (pierwszego) el. – „wsk1=tab ”i tylko karzemy mu na momencik skoczyć do pokoju obok ;-) Co z resztą widać... bo nie dodajemy w programie +1 tylko +i – za każdym razem przesuwa się o „i” i wraca tam gdzie jest ustawiony.*

Później jest już inaczej ze wsk2. Przypisany jest do adresu konkretnege elementu, a potem jest trwale przesunięty o kolejne 2 elementy – widać że odbywa się tam przypianie do adresu czyli „=”.

```
#include <iostream.h>
main()
{
int *wsk1, *wsk2;
int tab[10];
int i;

      wsk1=tab;
      for (i=0; i<10; i++)
            *(wsk1+i)=i*10;
      cout << "Elementy tablicy:\n";
      for (i=0; i<10; i++)
            cout << i << ": " << *(wsk1+i) << endl;
      cout << endl;
      wsk2 = &tab[3];
      cout << "3: " << *wsk2 << endl;
      wsk2 += 2;
      cout << "(3+2): " << *wsk2 << endl;
}
```

==========

❷❼

Tutaj przesyłanie przez wskaźnik – a więc jako argument jest adres „&....”, i na ten adres pokazuje odpowiedni parametr funkcji. Oczywiście zmiany zmiennej są trwałe – tak samo jak przy przesyłaniu argumnetów przez referencję.

Zawsze gdy parametrem deklarowanej funkcjijest wskaźnik to argumnetem jest adres. Tyle że pamiętamy że adresem tablicy jest jej pierwszy element – i przesyłając tablicę jako argument podajemy nazwę tablicy bez „&" (ważne przy stringach). Tutaj jednak mamy zwykłą zmienną int.

```
#include <iostream.h>
void zmien(int *wsk);
//---------------------
main()
{
int liczba=1;

      cout << "Przed wywolaniem funkcji: " << liczba << endl;
      zmien(&liczba);
      cout << "Po wywolaniu funkcji: " << liczba << endl;
}
//---------------------
void zmien(int *wsk)
{
      *wsk=100;
}
```

==========

❷❽

O przesyłeniu tablic. Wiec można tablicę przesłać zwyczajnie jako tablice (w sumie tak nie robimy) albo jako wskaźnik. Wtedy są dwie opcje:

1. *można korzystać z arytmetyki wskaźnika - *(wsk+i)*
2. *można przejść na zwykły zapis tablicowy – czyli wsk[i] – jest to równoważne.* ***Zwykle robimy własnie tak.***

```
#include <iostream.h>
void fkc_tab(int tab[], int rozmiar);
void fkc_wsk1(int *wsk, int rozmiar);
void fkc_wsk2(int *wsk, int rozmiar);
//---------------------
main()
{
int tablica[4] = { 1, 2, 3, 4 };
      fkc_tab(tablica,4);
      fkc_wsk1(tablica,4);
      fkc_wsk2(tablica,4);
      cout << endl;
}
//---------------------
void fkc_tab(int tab[], int rozmiar)
{
      cout << "\nPrzeslano jako tablice\n" ;
      for (int i=0; i<rozmiar; i++)
            cout << tab[i] << "\t";
}
//---------------------
void fkc_wsk1(int *wsk, int rozmiar)
{
      cout << "\nPrzeslano jako wskaznik - wariant 1\n" ;
      for (int i=0; i<rozmiar; i++)
            cout << *(wsk+i) << "\t";
}
```

```
//---------------------
void fkc_wsk2(int *wsk, int rozmiar)
{
      cout << "\nPrzeslano jako wskaznik - wariant 2\n" ;
      for (int i=0; i<rozmiar; i++)
            cout << wsk[i] << "\t";
}
```

==========

❷❾

Ten to w sumie tylko do przejrzenia – pojawia się dynamiczna alokacja, zarówno tablicy jak i pojedynczego obiektu int.

```
#include <iostream.h>
main()
{
int *wsk1, *wsk2;
int i, rozmiar;
      cout << "Podaj rozmiar tablicy: ";
      cin >> rozmiar;
      wsk1 = new int[rozmiar];
      if (wsk1==NULL)
            cout << "Brak pamieci\n";
      else
      {
            for (i=0; i<rozmiar; i++)
                  wsk1[i]=i*10;
            cout << "Tablica:\n";
            for (i=0; i<rozmiar; i++)
                  cout << i << ": " << wsk1[i] << endl;
            wsk2 = new int;
            *wsk2=10;
            cout << "*wsk2 = " << *wsk2 << endl;
            wsk2=wsk1;  // utrata dostepu do obiektu
            delete [] wsk1;
            wsk1=NULL;
            //delete wsk2; // niebezpieczenstwo zalamania programu
      }
}
```

==========

❸❿

A tutaj wskaźnik do wskaźnika... czyli dwie gwiazdki.

Osobiście.... bo C++ wywołuje bardzo osobiste przeżycia :] ... rozumiem zastosowanie tego tylko na podstawie tego oraz jednego z dodatkowych przykładów. Mianowicie żeby zaalokować dynamicznie coś z poziomu funkcji opierając się na wskaźniku zadeklarowanym w funkcji main... i wtedy jako argument jest adres wskaźnika, a parametrem funkcji jest wskaźnik do wskaźnika.

Chociaż można to zrobić naczej – a mianowicie tablicę docelową alokować wykorzystując wskaźnik zadeklarowany dopiero w funkcji – i wtedy funkcja działa na jednym parametrze – spróbuj to zrobić. Ale pewni z jakiś powodów się tego nie robi – może dlatego że w takiej sytuacji tablica docelowa istnieje tylko do momentu wygaśnięcia funkcji :]

Funkcja kody właśnie przerabia char na numer kodu ASCII o którym pisałem wcześniej. A dokładnie rzutuje char na int :]

```
#include <iostream.h>
char* dopisz_spacje(const char *wsk1, char *wsk2);
char* dopisz_spacje2(const char *wsk1, char **wsk2);
void kody(const char *wsk, int rozmiar);
//--------------------
main()
{
char tab1[] = { "tablica1" };
char tab2[20] = { 't','a','b','l','i','c','a','2' };
char tab3[] = { 't','a','b','l','i','c','a','3','\0' };
char *wsk1 = "wskaznik1";
char *wsk2 = { "wskaznik2" };
char *wsk=NULL;

      cout << "tab1: " << tab1 << endl;
      wsk = new char[80];
      cout << "t a b 1 : " << dopisz_spacje(tab1,wsk) << endl;
      delete [] wsk;
      wsk=NULL;
      cout << "t a b 1 (wersja 2): " << dopisz_spacje2(tab1,&wsk)
                        << endl;
      delete [] wsk;
      cout << "tab2: ";       kody(tab2,20);
      cout << "tab3: " << tab3 << endl;
      cout << "wsk1: " << wsk1 << endl;
      cout << "wsk2: " << wsk2 << endl;
}
//--------------------
char* dopisz_spacje(const char *wsk1, char *wsk2)
{
int ile_znakow, i;

      for (ile_znakow=0; *(wsk1+ile_znakow)!=NULL; ile_znakow++)
            /* */;
      for (i=0; i<ile_znakow; i++)
      {
            wsk2[2*i] = wsk1[i];
            wsk2[2*i+1] = ' ';
      }
      wsk2[2*ile_znakow]=NULL;
      return wsk2;
}
//------------------------
char* dopisz_spacje2(const char *wsk1, char **wsk2)
{
int ile_znakow, i;

      for (ile_znakow=0; *(wsk1+ile_znakow)!=NULL; ile_znakow++)
            /* */;
      *wsk2 = new char[2*ile_znakow+1];
      for (i=0; i<ile_znakow; i++)
      {
            (*wsk2)[2*i] = wsk1[i];
            (*wsk2)[2*i+1] = ' ';
      }
      (*wsk2)[2*ile_znakow]=NULL;
      return *wsk2;
}
```

```
//----------------------
void kody(const char *wsk, int rozmiar)
{
int i;
      for (i=0; i<rozmiar; i++)
            cout << int(*(wsk+i)) << ',';
      cout << endl;
}
```

==========

❸❶

Wskaźnki do funkcji – i tablica wskaźników do funkcji..... tego zasadniczo nie robiliśmy :]

```
#include <iostream.h>
int dodaj_dwa(int a);
int dodaj_trzy(int a);
int wywolaj(int (*wsk)(int), int liczba);
//----------------------
main()
{
int a=10;
int (*wsk_fkc)(int parametr);
int (*tab_wsk_fkc[2])(int); // tablica wskaznikow do funkcji

      cout << "dodaj_dwa: " << dodaj_dwa(a) << endl;
      cout << "dodaj_trzy: " << dodaj_trzy(a) << endl;
      wsk_fkc=dodaj_dwa;
      cout << "wsk. do dodaj_dwa: " << (*wsk_fkc)(a) << endl;
      wsk_fkc=dodaj_trzy;
      cout << "wsk. do dodaj_trzy: " << (*wsk_fkc)(a) << endl;
      cout << "f(dodaj_trzy): " << wywolaj(dodaj_trzy,a) << endl;
      tab_wsk_fkc[0]=dodaj_dwa;
      tab_wsk_fkc[1]=dodaj_trzy;
      cout << "tablica ze wsk. do dodaj_trzy: "
            << (*tab_wsk_fkc[1])(a) << endl;
}
//------------------------
int dodaj_dwa(int a)
{     return a+2; }
//------------------------
int dodaj_trzy(int a)
{     return a+3; }
//------------------------
int wywolaj(int (*wsk)(int), int liczba)
{     return (*wsk)(liczba); }
```

==========

❸❷

Przeładowania funkcji .... Samo w sobie to jest proste. Nazywamy parę funkcji tak samo, a kompilator rozpoznjaje potem o którą nam chodzi po rodzajach parametrów- czyli w programie nie może być dwóch funkcji o takiej samej nazwie i takim samym typie i kolejności parametrów.

```
#include <iostream.h>
void drukuj(int);
void drukuj(float);
void drukuj(char);
void drukuj(int, float);
```

```
      void drukuj(float, int);
      //-----------------
      main()
      {
      int a = 10;
      float b = 12.6;
      char c = 'a';

            drukuj(a);
            drukuj(b);
            drukuj(c);
            drukuj(a,b);
            drukuj(b,a);
      }
      //-----------------
      void drukuj(int a)
      { cout << a << endl; }
      //-----------------
      void drukuj(float a)
      { cout << a << endl; }
      //-----------------
      void drukuj(char a)
      { cout << a << endl; }
      //------------------
      void drukuj(int a, float b)
      { cout << a << "   " << b << endl; }
      //------------------
      void drukuj(float b, int a)
      { cout << a << "   " << b << endl; }
```

==========