Laboratorium nr 9

Temat: Wskaźniki, referencje, dynamiczny przydział pamięci, tablice dynamiczne.

Zakres laboratorium:

- wskaźniki
- referencje
- zastosowanie wskaźników wobec tablic
- dynamiczny przydział pamięci, operatory **`new`** i **`delete`**
- tablice dynamiczne
- jak sprawdzić, czy operacja alokacji pamięci się powiodła?
- zadania laboratoryjne

Wskaźniki

Każda zmienna ma unikalny adres wskazujący obszar pamięci zajmowany przez tą zmienną. Adres ten można przechowywać w tzw. zmiennej wskaźnikowej (wskaźniku).

Wskaźnik to obiekt (zmienna), która przechowuje adres do innego obiektu (zmiennej).

Przed użyciem wskaźnika, musimy go przypisać do konkretnego obiektu (przypisać mu adres tego obiektu). Wskaźnik można łatwo przestawić, by pokazywał na inny obiekt tego samego typu.

Do każdego wskaźnika można podstawić adres 0, zwany czasem NULL (np. `wsk=0;` lub `wsk=NULL;`). Ustawienie wskaźnika na ten adres powoduje, że wskaźnik nie pokazuje na nic sensownego.

Wskaźnik

```
float pi=3.14;   //definicja zmiennej pi
float *wsk=&pi;  //definicja wskaźnika do zmiennej typu float
```

LUB

```
float *wsk;
wsk=&pi;
```

```
cout<<"wartosc zmiennej pi: "<<pi;
cout<<"wartosc zmiennej wsk: "<<wsk;
```

→ 3.14
→ FFF0

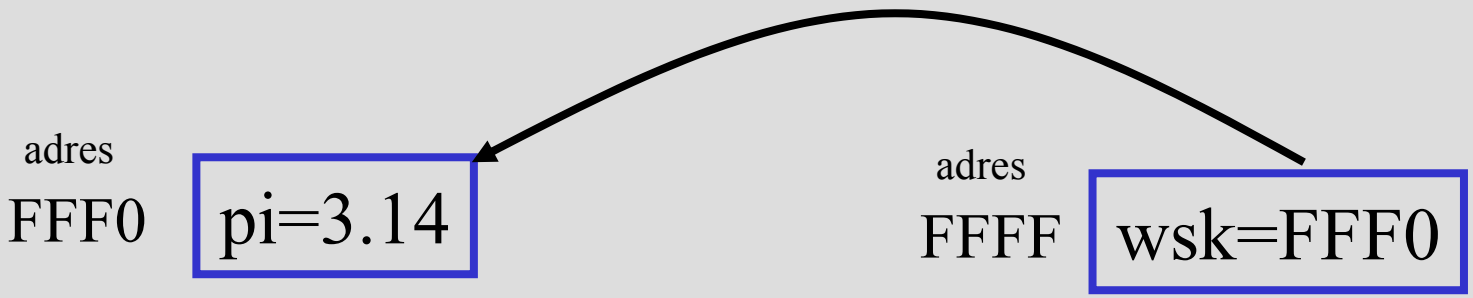
```
cout<<"adres zmiennej pi: "<<&pi;
cout<<"adres zmiennej wsk: "<<&wsk;
```

→ FFF0
→ FFFF

```
pi=3.141;        //zmiana wartości zmiennej pi
*wsk=3.1415;     //zmiana wartości zmiennej pi za pomocą wskaźnika

cout<<"wartosc pi za pomoca wskaznika: "<<*wsk;
```

→ 3.1415



Referencje

Referencja jest inną nazwą obiektu (zmiennej).

Podczas definiowania referencji trzeba ją od razu zainicjalizować.

Referencja nie jest kopią zmiennej, ale tą samą zmienną pod inną nazwą.

Referencja

```
int a=5;        //definicja i inicjalizacja zmiennej a
int &ref=a;     //definicja referencji o nazwie ref,
                //oraz ustawienie jej na zmienną a
```

```
cout<<"wartosc zmiennej a: "<<a;
cout<<"wartosc zmiennej ref: "<<ref;
```

→ 5
→ 5

```
ref++; //dodanie do ref wartosci 1
```

```
cout<<"wartosc zmiennej a: "<<a;
cout<<"wartosc zmiennej ref: "<<ref;
```

→ 6
→ 6

```
cout<<"adres zmiennej a: "<<&a;
cout<<"adres zmiennej ref: "<<&ref;
```

→ FFF0
→ FFF0

adres
FFF0

a=5
ref=5

Zastosowanie wskaźników wobec tablic

Przykład z ruchu wskaźnika wobec tablicy:

```
int *wskaznik;                  //definicja wskaźnika do typu int
int tablica[10]={0,1,2,3,4,5,6,7,8,9};  //definicja 10-elem. tablicy typu int

wskaznik=&tablica[0];           //ustawienie wskaźnika na początek tablicy (element 0)
wskaznik=tablica;               //ustawienie wskaźnika na początek tablicy (element 0)
cout<<*wskaznik<<endl;          //wyświetlenie pierwszego elementu (liczba 0)

wskaznik=&tablica[3];           //ustawienie wskaźnika na 4 element tablicy
cout<<*wskaznik<<endl;          //wyświetlenie 4 elementu (liczba 3)

wskaznik=wskaznik+1;            //przesunięcie wskaźnika na 5 element tablicy
cout<<*wskaznik<<endl;          //wyświetlenie 5 elementu (liczba 4)

wskaznik++;                     //przesunięcie wskaźnika na 6 element tablicy
cout<<*wskaznik;                //wyświetlenie 6 elementu (liczba 5)

cout<<*wskaznik++;              //wyświetlenie 6 elementu (liczba 5)
                                //i przesunięcie wskaźnika na element 7
cout<<*wskaznik<<endl;          //wyświetlenie 7 elementu (liczba 6)

cout<<*(wskaznik++);            //wyświetlenie 7 elementu (liczba 6)
                                //i przesunięcie wskaźnika na element 8
cout<<*wskaznik<<endl;          //wyświetlenie 8 elementu (liczba 7)

cout<<*(wskaznik+1);            //wyświetlenie 9 elementu (liczba 8) (bez przesuwania wskaźnika)

wskaznik+=2;                    //przesuniecie wskaźnika na 10 (ostatni element tablicy)
cout<<*wskaznik<<endl;          //wyświetlenie 10 elementu (liczba 9)

wskaznik++;                     //UWAGA: niebezpieczne, wskaźnik poza obszarem tablicy!!!
cout<<*wskaznik<<endl;          //wyświetlenie elementu spoza tablicy (śmieć)
```

Zostaną wyświetlone elementy tablicy w następującym porządku:

0 3 4 5 5 6 6 7 8 9 śmieć

Dynamiczny przydział pamięci, operatory **`new`** i **`delete`**

Operatory **new** i **delete** służą do dynamicznego alokowania pamięci operacyjnej komputera. Mogą być wykorzystane dla dowolnego typu danych, tj. wbudowanego lub zdefiniowanego przez użytkownika.

Dynamiczne przydzielanie pamięci umożliwia programiście tworzenie i usuwanie zmiennych, a tym samym pełne sterowanie czasem ich istnienia. Zmienne przechowywane są w specjalnie wydzielonym obszarze pamięci do swobodnego używania (ang. *heap* – zapas, ang. *free store* – swobodnie dostępny magazyn).

Do przydzielania i zwalniania pamięci służą odpowiednio słowa kluczowe **new** i **delete**. Za pomocą operatora **delete** kasuje się tylko obiekty stworzone operatorem **new**. Dostęp do zmiennych dynamicznych umożliwiają wskaźniki.

Cechy obiektów stworzonych operatorem **new**:

1 obiekty istnieją od momentu ich utworzenia operatorem **new** do momentu skasowania operatorem **delete**, **to my decydujemy o czasie życia obiektów**

2 **obiekty takie nie mają nazwy**, można nimi operować tylko za pomocą wskaźników

3 **obiektów tych nie obowiązują zwykłe zasady o zakresie ważności**, czyli to, w których miejscach programu są widzialne lub niewidzialne

4 w obiektach dynamicznych (tworzonych operatorem **new**) zaraz po ich utworzeniu tkwią jeszcze „śmieci” (przypadkowe wartości), musimy zatem **zadbać o wstępną inicjalizację tych obiektów**

Przykład 1:

```
float *nazwa;              //definicja wskaźnika do typu float
nazwa = new float;         //utworzenie obiektu float o odpowiedniej wielkości

LUB
float *nazwa = new float; //można w jednej linii zaalokować pamięć

*nazwa=1;                  //zapisanie wartości 1 do obiektu typu float

delete nazwa;              //zwolnienie pamięci zajmowanej przez obiekt
nazwa=0;                   //odłączenie wskaźnika od pamięci
```

Przykład 2:

```
int *wsk;                  //definicja wskaźnika do typu int
wsk = new int(10);         //inicjalizacja dynamicznie zaalokowanej pamięci

LUB
int *wsk = new int(10);    //można w jednej linii zaalokować pamięć

cout<<*wsk<<endl;          //wyświetlenie zawartości obiektu

delete wsk;                //zwolnienie pamięci zajmowanej przez obiekt
wsk=0;                     //odłączenie wskaźnika od pamięci
```

UWAGA: Po zwolnieniu zaalokowanej wcześniej pamięci, nadaj wskaźnikowi, którym się posługiwałeś wartość 0, aby mieć do niej dostęp. Spowoduje to odłączenie wskaźnika od tego obszaru pamięci.

Tablice dynamiczne

Przykład 1:

```
float *t;                  //definicja wskaźnika do typu float
t = new float[10];         //utworzenie 10-elementowej tablicy typu float

LUB
float *t = new float[10]; //można w jednej linii zaalokować tablicę

for(int i=0;i<10;i++)
   cin>>t[i];              //wczytanie danych do tablicy

delete [] t;               //zwolnienie pamięci zajmowanej przez tablicę
t=0;                       //odłączenie wskaźnika od pamięci
```

Przykład 2:

```
int *tab,rozmiar=5;        //definicja wskaźnika do typu int oraz zmiennej int
tab = new int[rozmiar];    //dynamiczne zaalokowanie pamięci o podanym rozmiarze

LUB
int *tab = new int[rozmiar];       //można w jednej linii zaalokować pamięć

for(int i=0;i<rozmiar;i++)
   cout<<tab[i]<<endl;     //wyświetlenie zawartości tablicy

delete [] tab;             //zwolnienie pamięci zajmowanej przez tablicę
tab=0;                     //odłączenie wskaźnika od pamięci
```

UWAGA: Po zwolnieniu zaalokowanej wcześniej pamięci, nadaj wskaźnikowi, którym się posługiwałeś wartość 0, aby mieć do niej dostęp. Spowoduje to odłączenie wskaźnika od tego obszaru pamięci.

Przykład 3:

```
int m = 3;                                  //liczba wierszy
int n = 5;                                  //liczba kolumn
int **macierz;


macierz = new int*[m];                      //KROK 1: alokacja wierszy

for (int j=0;j<m;j++)
   macierz[j]=new int[n];                   //KROK 2: alokacja kolumn


for(int i=0;i<m;i++)
   for(int j=0;j<n;j++)
      cin>>macierz[i][j];                   //wczytanie danych do macierzy


for (int i=0;i<m;i++)
   delete[] macierz[i];                     //KROK 1: usuwanie kolumn

delete[] macierz;                           //KROK 2: usuwanie wierszy


macierz=NULL;                               //odłączenie wskaźnika od pamięci
```

Jak sprawdzić, czy alokacja pamięci się powiodła?

```
double *t;                  //definicja wskaźnika do typu double
t = new double[10000];      //utworzenie 10000-elementowej tablicy typu double


if(!t)
{
   cout<<”brak pamięci”;
}
```

LUB

```
if(t==NULL)
{
   cout<<”brak pamięci”;
}
```

LUB

```
if(t==0)
{
   cout<<”brak pamięci”;
}
```

Zadania laboratoryjne