Wskaźniki i dynamiczna alokacja pamięci – wiadomości wstępne

Na dzisiejszych zajęciach omówione zostaną podstawy obsługi wskaźników w języku C++. Jest to jeden z najważniejszych mechanizmów tego języka, stawiający jednak duże wymagania: nieumiejętne użytkowanie zmiennych wskaźnikowych prowadzi do trudnych do wykrycia błędów, podczas gdy prawidłowe ich użycie udostępnia programiście duże możliwości.

Do zagadnień związanych ze wskaźnikami należy:

- deklarowanie wskaźników
- nadawanie wartości wskaźnikom
- użytkowanie wskaźników do odwoływania się do wskazywanych obszarów pamięci

Czym jest wskaźnik? Wskaźnik jest typem złożonym języka, służącym do informowania, gdzie w pamięci komputera znajdują się poszukiwane dane lub funkcje. Zmienne wskaźnikowe służą do przechowywania adresu wybranego przez nas obiektu. Zmienna typu wskaźnikowego jest przystosowana do przechowywania adresu zmiennej lub funkcji znanego kompilatorowi typu. Jeżeli mamy zapamiętany we wskaźniku adres np. zmiennej o nazwie „*indeks*“, to możemy za pomocą odpowiedniego operatora działającego na wskaźnik zagwarantować sobie dostęp do tej zmiennej, nie używając jej nazwy. Zagadnienia te zostaną omówione na przykładzie poniższego programu (plik *wskazniki.cpp*)

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

void inkrementuj(int *wskaznik);
void inkrementuj(int wartosc);

int main()
{
int liczba = 10;
int *wsk;
char napis[25] = "Absolute zero is cool...";

   cout<<"Program ilustruje uzywanie wskaznikow"<<endl<<endl;

   cout<<"Wartosc zmiennej liczba wynosi obecnie "<< liczba << endl;

   cout<<"Wpisujemy do wskaznika adres zmiennej liczba..."<<endl<<endl;
   // Najpierw "wycelujemy" wskaznikiem na zmienna liczba
   // Adres zmiennej liczba pobierzemy operatorem & (ampersand)
   wsk = &liczba;

   // Uzycie zmiennej liczba poprzez wskaznik
   cout<<"Wpiszemy do zmiennej wskazywanej prze wsk wartosc 1000"<<endl;
   *wsk = 1000;
   cout<<"Wartosc zmiennej liczba wynosi obecnie "<< liczba << endl << endl;

   cout<<"Funkcja moze zmienic wartosc zmiennej, jesli ma do niej wskaznik"<<endl;
   inkrementuj(wsk);
   cout<<"Wartosc zmiennej liczba wynosi obecnie "<< liczba << endl << endl;

   cout<<"Bez wskaznika to sie jej nie uda..."<<endl;
   inkrementuj(liczba);
   cout<<"Wartosc zmiennej liczba wynosi obecnie "<< liczba << endl << endl;

   cout<<"Do komorki tablicy mozna odniesc sie poprzez wskaznik..."<<endl;
   cout<<"Tablica napis zawiera znaki: ";
   for(liczba = 0; liczba<25; liczba ++) cout<< *(napis + liczba);
   cout<<endl;
}

void inkrementuj(int *wskaznik)
```

```
{
   *wskaznik += 1;
}

void inkrementuj(int wartosc)
{
   wartosc += 1;
}
```

Program ten podczas swojej pracy wyświetla na ekranie następującą treść:

```
Program ilustruje uzywanie wskaznikow

Wartosc zmiennej liczba wynosi obecnie 10
Wpisujemy do wskaznika adres zmiennej liczba...

Wpiszemy do zmiennej wskazywanej prze wsk wartosc 1000
Wartosc zmiennej liczba wynosi obecnie 1000

Funkcja moze zmienic wartosc zmiennej, jesli ma do niej wskaznik
Wartosc zmiennej liczba wynosi obecnie 1001

Bez wskaznika to sie jej nie uda...
Wartosc zmiennej liczba wynosi obecnie 1001

Do komorki tablicy mozna odniesc sie poprzez wskaznik...
Tablica napis zawiera znaki: Absolute zero is cool...
```

Do deklaracji wskaźnika stosuje się operator * w następujący sposób: *int *wsk;* Instrukcja ta oznacza definicję zmiennej *wsk*, będącej wskaźnikiem (o czym informuje gwiazdka po prawej stronie nazwy typu) do zmiennych typu *int*. Tak wygląda najprostsza deklaracja zmiennej wskaźnikowej. Można deklarować wskaźniki do bardziej złożonych obiektów, na przykład:

- **char *tab[];** - *tab* jest tablicą wskaźników do zmiennych typu char
- **char (*wsk)[];** - *wsk* jest wskaźnikiem do tablicy zmiennych typu char
- **void (*fun)(double)**; - *fun* jest wskaźnikiem do funkcji nie zwracającej wartości, o jednym argumencie typu double

Przy tworzeniu tego typu deklaracji decydujące znaczenie ma hierarchia operatorów języka, która znaczenie pierwszorzędne nadaje nawiasom kwadratowym i okrągłym, zaś mniejsze operatorowi *. Możliwe są znacznie bardziej uwikłane deklaracje wskaźników, jednak podane przed chwilą przykłady w praktyce zbliżone są stopniem trudności do najczęściej stosowanych.

Mając zdefiniowaną zmienną wskaźnikową, trzeba nadać jej wartość (przed pierwszym użyciem wskaźnika), będącą adresem obiektu, który będzie przez nią wskazywany. Aby tego dokonać, należy przypisać do zmiennej wskaźnikowej adres wybranego obiektu. Do uzyskania adresu zmiennej służy operator **&** (ang. *ampersand*). Jego użycie sprowadza się do postawienia go przed obiektem, którego adres chcemy pobrać np. **wsk = &indeks;** gdzie *indeks* to zmienna odpowiedniego typu (tego, do którego wskazywania deklarowaliśmy wskaźnik). W przypadku tablic oraz funkcji ich nazwa jest symbolicznym oznaczeniem ich adresu. Przypisanie wskaźnikowi wartości jest konieczne, gdyż wskaźnik zawsze ma pewną wartość, a zatem wskazuje pewne miejsce w pamięci. Jeśli spróbujemy użyć wskaźnika przed przypisaniem mu pożądanego adresu, możemy zniszczyć przez przypadek jakieś dane lub wywołać zakończenie programu przez system operacyjny, który uzna program, odwołujący się do nie swoich danych jako niebezpieczny. To jest pierwsze z zagrożeń, na jakie napotykamy przy pracy ze wskaźnikami.

Mając wskaźnik zainicjalizowany adresem pożądanej zmiennej, możemy odnosić się do tej zmiennej poprzez wskaźnik. Stawiając przed zmienną wskaźnikową symbol gwiazdki (jest to nazywane w literaturze *dereferencją* lub *wyłuskaniem* wskaźnika), otrzymujemy dostęp do wskazywanego obiektu. Możemy takiego wyrażenia użyć w miejscach, gdzie może pojawić się zmienna typu, wskazywanego przez użyty wskaźnik. Interesująca jest metoda odnoszenia się do elementu tablicy poprzez wskaźnik. Jeżeli *tab* jest nazwą tablicy, w celu odniesienia się do komórki o indeksie *i*, piszemy

***(tab + i)**

Jak rozumieć tę instrukcję? Istnieje kardynalna zasada mówiąca, że **„Nazwa tablicy jest adresem jej pierwszego elementu (czyli elementu o indeksie, równym 0)“**. Ponadto cechą wskaźników jest to, że do wskaźnika można dodać liczbę całkowitą, w wyniku czego otrzyma się zawsze poprawny adres elementu tablicy danego typu,

odległej od pokazywanej przez wskaźnik o ilość elementów, równą dodanej liczbie. Po wykonaniu dodawania można zastosować operator * (gwiazdka) do wyniku (nawiasy wstawione są ponownie z racji hierarchii operatorów), przez co otrzymamy wskazywany obiekt, czyli komórkę tablicy o indeksie *i*.

Ostatni aspekt poruszony w powyższym programie to użycie wskaźnika jako argumentu funkcji. Języki C oraz C ++ charakteryzuje to, że argumenty w tych językach przekazywane funkcjom przez wartość. Oznacza to, że funkcja dostaje kopie wartości dostarczonych jej argumentów i nie będzie w stanie np. zmienić wartości zmiennej, podanej jej jako argument. W pewnych sytuacjach chcielibyśmy jednak móc dokonać modyfikacji. Można tego dokonać, tworząc argument, będący wskaźnikiem do wartości, którą chcemy zmodyfikować - dokonując dereferencji wskaźnika mamy możliwość dokonania zmiany wartości wskazywanego obiektu. W języku C był to jedyny dostępny mechanizm, aby wewnątrz funkcji zmodyfikować zmienną, nie znajdującą się w zakresie tej funkcji. W języku C++ można w tym celu zastosować referencje, ale mechanizm przekazywania argumentu przez wskaźniki został zachowany.

Przeanalizujmy kolejny program, traktujący o tzw. dynamicznej alokacji pamięci oraz użytkowaniu struktur za pomocą wskaźników (plik *wsk_struct.cpp*):

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

struct karta
{
   int figura;
   int kolor;
};

void wypisz(karta *wsk);

int main()
{
karta k1 = {1, 2};
karta *wskart;

   cout<<"Program demonstruje obsluge struktur za pomoca wskaznika"<<endl<<endl;

   cout<<"Utworzymy dynamicznie zmienna typu karta... ";

   // Operator new przydziela pamiec na nowa zmienna
   try {
         wskart = new karta;
   }
   catch(bad_alloc) {
         cout<<"blad alokacji pamieci"<<endl;
         return 0;
   }

   cout<<"Dokonane zostanie przypisanie struktury w calosci"<<endl<<endl;
   *wskart = k1;

   cout<<"Struktura zaalokowana dynamicznie ma wartosci:"<<endl;
   wypisz(wskart);
   cout<<endl;

   // Na koniec musimy zniszczyc zmienna zaalokowana dynamicznie
   delete wskart;
}

void wypisz(karta *wsk)
{
   cout<< "Karta ma wartosci pol: figura=";
   cout<< wsk->figura ;

   cout<<", kolor=";
   cout<< wsk->kolor ;
}
```

Spójrzmy na funkcję *wypisz*. Jest tam użyty specjalny operator -> służący do wybrania pola ze struktury, do której mamy dostęp poprzez wskaźnik. Wyrażenie **wsk->kolor** jest równoważne wyrażeniu **(*wsk).kolor** – oba powodują dereferencję składowej struktury o nazwie kolor, wskazywanej przez *wsk*, choć pierwszy zapis jest krótszy i bardziej czytelny.

W programie tym jest użyty mechanizm *dynamicznej alokacji pamięci*. Termin ten oznacza, że na żądanie napisanego przez nas programu system operacyjny przyznaje nam obszar pamięci, aby można było przechowywać w nim dane. W jakim celu używa się tego mechanizmu? Pisząc program nie zawsze jest możliwość przewidzenia, ile informacji będzie on przechowywał. Na przykład, pisząc w edytorze tekstu, można utworzyć dokument zajmujący kilka linijek tekstu, ale również wieleset stron. Nie da się w takim wypadku utworzyć sensownie zmiennych dla każdego przypadku: jeśli przyjmiemy objętość kilku linijek, nie zmieściłby nam się duży dokument w edytorze, jeśli zarezerwujemy duże ilości pamięci na tekst, to pisząc krótką notatkę, marnujemy niepotrzebnie pamięć operacyjną komputera - dopóki nie zakończymy naszego „zachłannego“ programu, blokujemy duży obszar pamięci niepotrzebnie. Aby rozwiązać ten problem, możemy przechowywać dane programu w obszarze pamięci, jaki zostanie udostępniony przez system operacyjny, w ilości jaka jest w danej chwili potrzebna. Zyski z takiego rozwiązania są następujące: dopóki pamięć faktycznie nie jest potrzebna, możemy jej nie rezerwować (istotne zmienne w programie istnieją przez cały cały czas jego pracy) oraz możemy zwolnić ją, gdy tylko przestanie być potrzebna, ponadto mamy możliwość używać tylko tyle pamięci, ile jest w danej chwili potrzebne. Dynamiczna alokacja pamięci niesie jednak ze sobą również problemy. Pierwszym z nich jest to, że możemy nie dostać żądanej ilości pamięci np. z powodu jej chwilowego lub permanentnego braku – trzeba opracować strategie postępowania na taką okoliczność. Po drugie, dynamicznie alokowane obszary pamięci należy zwolnić. Jeśli się tego nie zrobi, pamięć przyznana, ale nie zwolniona i nieużywana jest nieużyteczna dla systemu (określa się taką sytuację żargonowo jako „wyciek pamięci”, ang. memory leak). W zależności od systemu operacyjnego, taka pamięć może być np. odzyskana automatycznie (po zakończeniu działania programu) lub niedostępna aż do ponownego uruchomienia systemu operacyjnego. Procedura tworzenia zmiennych dynamicznych zilustrowana jest w powyższym programie, skrótowo przebiega następująco:

1. alokacja pamięci operatorem *new* , po którym następuje nazwa typu (w szczególności dla tablic stosuje się wyrażenie *new typ[rozmiar]*). Jeśli alokacja się nie powiedzie, zostaje zrealizowane tzw. rzucenie wyjątku (ang. *exception throw*) należącego do klasy *bad_alloc*. Typowa procedura obsługi tzw. przechwycenia wyjątku *bad_alloc* przedstawiona jest w powyższym programie, zaś jej pominięcie spowoduje awaryjne zakończenie działania programu, co w prostych sytuacjach bywa również wystarczającym rozwiązaniem.
2. użytkowanie przyznanej pamięci za pomocą wskaźnika.
3. zwolnienie pamięci operatorem *delete* po którym podaje się wskaźnik do zwalnianego obszaru pamięci (dla tablic należy zastosować wyrażenie *delete [] wskaźnik*).

Uwaga! Nie wolno zwalniać wielokrotnie tego samego obszaru pamięci. Najbezpieczniejsze rozwiązanie polega na przypisywaniu nieużywanym wskaźnikom, służącym do pracy ze zmiennymi alokowanymi dynamicznie, wartości *NULL* przed ich użyciem i po każdym zwolnieniu wskazywanego przez nie obszaru pamięci. Wartość *NULL*, zawarta we wskaźniku gwarantuje, że spodziewany we wskaźniku adres potraktowany zostanie jako nieważny przy ewentualnej próbie zastosowania względem niego operatora *delete*.

Zadanie

Napisać program, który utworzy tablicę ze wskaźnikami do struktury, przechowującej dane z telefonami (patrz poprzedni skrypt), poprawnie będzie obsługiwał dynamiczną alokację pamięci na te struktury oraz będzie potrafił pobrać, jak również wypisać te dane.