Piękno programowania

Wskaźniki, referencje, głęboka depresja

Każda zmienna przy inicjalizacji ma przypisywany swój adres w pamięci. Gdy w programie umieścimy definicję:

```
int i=200;
```

to tak naprawdę, gdzieś w pamięci RAM umieszczona zostanie wartość 200, a jej adres zostanie zapamiętany. Zawsze, gdy będziemy chcieli się do wartości zlokalizowanej w pamięci odwołać, to kompilator nie będzie widział nazwy tej zmiennej, a jej konkretny adres, nazwa jest tylko dla nas.

Sprawdzić, pod jakim adresem znajduje się konkretna zmienna możemy używając operatora '&' ustawionego przed nazwą zmiennej.

```
int i=200;
int j=100;
cout<<i<<" "<<&i<<endl;
cout<<j<<" "<<&j<<endl;
```

Możemy teraz w terminalu zobaczyć, gdzie te zmienne w pamięci wylądowały:

```
200 0x7ff7b47cf6b8
100 0x7ff7b47cf6b4
```

Możemy też zobaczyć, że te zmienne umieszczone zostały obok siebie, jedna po drugiej (konkretnie jedna przed drugą).

Wskaźnikiem nazywamy zmienną, której wartością jest adres innej zmiennej. Dokładnie tak samo, jak w wypadku każdej innej zmiennej, najpierw musimy ten wskaźnik **zadeklarować** i dokładnie tak samo jak z każdą inną zmienną, **wskaźnik musi mieć swój typ**

```
int* ptr;
```

Różnica pomiędzy wskaźnikiem a zmienną jest dość spora, ponieważ każdy wskaźnik, mimo że musimy określić jakiego będzie typu, to tak naprawdę zawsze będzie długą liczbą heksadecymalną. Określenie typu służy do określenia, na jakiego typu zmienną taki wskaźnik będzie wskazywał.

```
int i=200;
int* ptr;
ptr = &i;
cout<<i<<" "<<&i<<endl;
cout<<ptr<<" "<<*ptr<<endl;
```

Do zadeklarowanego wskaźnika przypisywać możemy w zasadzie jedynie adres zmiennej, pamiętając o zgodności typów danych. I teraz ważne. Adres zmiennej uzyskamy stawiając przed nią '&', natomiast wartość zmiennej, której adres przechowuje wskaźnik uzyskamy stawiając przed nim '*'.

Tablice

Śmieszna sprawa, wyobraźcie sobie, bo powiązanie wskaźników z tablicami jest dość silne. Na przykład, wskaźnikiem wskazującym na początek tablicy możemy manipulować i w ten sposób uzyskiwać dostęp do kolejnych elementów tablicy nie używając standardowych indeksów:

```
int arr[]={2, 1, 37};
int *ptr = arr;
for(int i=0; i<sizeof(arr)/sizeof(*ptr);i++){
     cout<<"Wartość zmiennej pod adresem:"<<ptr+i
     <<": "<<*(ptr+i)<<endl;
}
```

Wykonanie powyższego kodu wydrukuje nam wszystkie elementy tablicy wraz z ich adresami. Można tu zauważyć ciekawą właściwość wskaźników, bowiem w sytuacji, gdy ptr ma wartość `0x7ff7ba4ae6b0`, to ptr+1 nie będzie miało wartości …e6b1, a …e6b4, czyli zwiększenie wskaźnika o 1 tak naprawdę zwiększyło go o 4. Bierze się to z tego, że inkrementując wskaźnik nie zwiększamy zawartego w nim adresu o 1, a o rozmiar typu danego wskaźnika. W powyższym wypadku typem jest int, który ma rozmiar 4 bajtów, dlatego wskaźnik zwiększany jest o 4.

Funkcje

Gdy przekazujemy do funkcji jakąś zmienną, przy założeniu nie jest ona globalna, to przy wywołaniu tejże funkcji stworzona zostanie kopia tejże zmiennej, więc zmienna podana w wywołaniu nie jest w żaden sposób modyfikowana, wewnątrz funkcji pracujemy bowiem na jej kopii.

```
void test(int param);
int main (){
    int a=10;
    test(a);
    cout<<a;
    return 0;
}
void test(int param){
    param=param+8;
}
```

W powyższym, raczej głupim, przykładzie wywołamy funkcję test na zmiennej a, ale jeśli sprawdzimy sobie adresy zmiennych `a` i `param`, to okaże się, że są to zupełnie inne zmienne,

znajdujące się w różnych lokalizacjach, a jedynie z takimi samymi wartościami. Co za tym idzie, modyfikując zmienną `param` nie tykamy zmiennej `a`.

Inaczej mieć się będzie sytuacja, w której do funkcji podany zostanie wskaźnik.

```
void test(int *param);
int main (){
    int a=10;
    test(&a);
    cout<<&a<<" "<<a;
    return 0;
}
void test(int *param){
    cout<<param<<" "<<*param<<endl;
    *param=*param+8;
}
```

Z technicznego punktu widzenia, do funkcji podany będzie podany adres zmiennej. W funkcji wyciągamy zmienną, która znajduje się na tym adresie, modyfikujemy ją i zapisujemy na tym samym adresie. Dzięki temu z wnętrza funkcji test zmodyfikowaliśmy zmienną z funkcji main, do której w tradycyjnym przypadku nie mieliśmy dostępu.

Zadanie

Święta za pasem, więc lekkie i przyjemne. Rozwiąż zadanie 3 z Lab9, ale tak, aby uniknąć kopiowania tablicy do każdej z funkcji. Fajnie by było, gdyby w podpunkcie f) zamiana następowała na głównej tablicy i miała możliwość wpłynięcia na realizację pozostałych funkcji.