Podstawy Programowania
Zajęcia laboratoryjne 9
I rok Bioinformatyki Politechniki Poznańskiej

Wskaźniki

1 Wskaźniki

Wskaźnik to specjalny rodzaj zmiennej, w której zapisany jest adres w pamięci komputera. Oznacza to, że wskaźnik wskazuje miejsce, gdzie zapisana jest jakaś informacja (np. zmienna typu liczbowego czy struktura).

Adres pamięci to pewna liczba całkowita, jednoznacznie definiująca położenie pewnego obiektu w pamięci komputera. Tymi obiektami mogą być np. zmienne, elementy tablic czy nawet funkcje. Jeżeli str jest obiektem typu char, a pstr ma być wskaźnikiem, który wskazuje na ten obiekt, to taki wskaźnik można zdefiniować następujący sposób:

```
1 char str;
2 char *pstr; //* operator odwolania sie do danych wskazywanych przez wskaznik
3 pstr = &str; //& operator pobrania adresu (zmiennej, struktury, tablicy itp.)
```

Deklaracja wskaźnika pojawia się już w powyższym przykładzie, poprzez podanie typu, operatora dereferencji * i nazwy zmiennej wskaźnikowej. W poniższym przykładzie zarówno wskaźnik jak i tablica są zmiennymi typu wskaźnikowego.

```
1 typ *wskaznik;
2 typ tablica[rozmiar];
```

Przypisanie adresu do wkaźnika:

```
1 wskaznik = inny_wskaznik;
2 wskaznik = &zmienna;
```

Odczyt wartości z adresu / zapis pod adresem:

```
1 zmienna = *wskaznik;
2 *wskaznik = wartosc;
```

Działania na wskaźnikach:

```
1 wskaznik++;
2 wskaznik--;
3 wskaznik + indeks;
4 wskaznik[indeks];
```

Operator wyłuskania (*zmienna) ma wysoki priorytet, szczególnie z operatorami ++ i --. Zwróć uwagę, że poniższe działania można łączyć:

```
1 wskaznik = inny_wskaznik + indeks;
2 zmienna = *(wskaznik + indeks);
3 zmienna = *wskaznik++;
4 wskaznik[indeks];
```

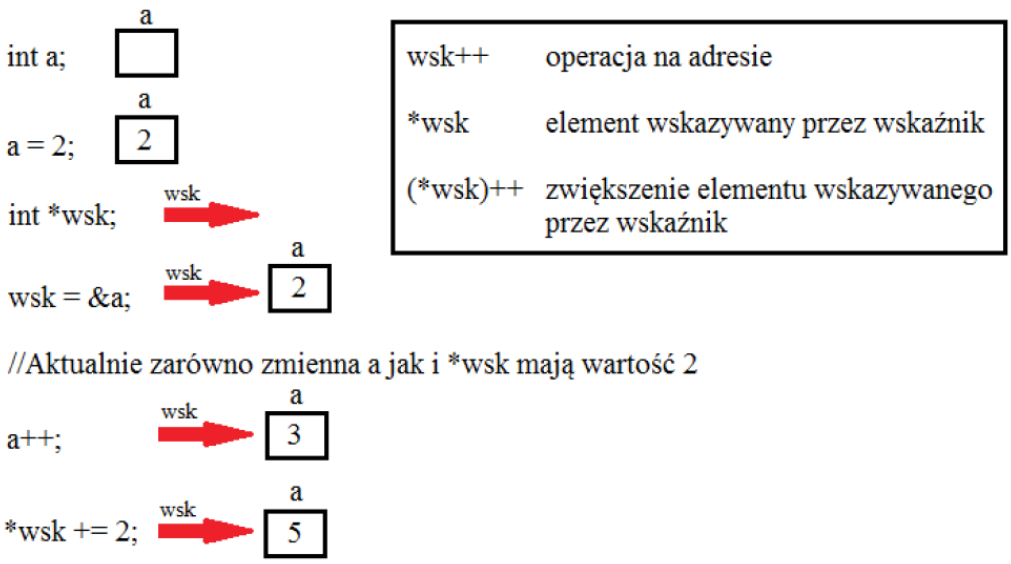

Figure 1: Operacje na wskaźnikach

Wskaźniki typu void: w C wskaźniki dowolnego typu mogą być przydzielane do wskaźników typu void i odwrotnie, natomiast w C++ wymagana jest jawna konwersja:

```
1 void *wsk;
2 char *str = "Tekst";
3 wsk = (char*)str;
```

2 Przykłady do przetestowania i zrozumienia

Wskaźniki, zapoznanie się z użyciem wskaźników na poniższych przykładach (kod 1.1 w materiałach na stronie):

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int temp;
6      int *wskaznik;
7      temp = 100;
8
9      printf("Liczba w zmiennej TEMP: %d\n",temp);
10     wskaznik = &temp;     // przypisujemy wskaznikowi
11                           // adres zmiennej TEMP
12     *wskaznik = 200;      // pod adres wskaznika
13                           // przypisujemy wartosc 200
14     printf("Liczba wskazywana przez wskaznik: %d\n",*wskaznik);
15     printf("Liczba w zmiennej TEMP: %d\n",temp);
16
17     return 0;
18 }
```

Kolejny przykład (kod 1.2 w materiałach na stronie):

```c
#include <stdio.h>

int main()
{
    int liczba = 65;
    printf("Wartosc zmiennej: %d\n", liczba);
    printf("Adres zmiennej: %p\n", &liczba);

    return 0;
}
```

Wskaźniki i tablice, w języku C wskaźniki i tablice w wielu przypadkach traktowane są tak samo. Elementy tablicy podczas uruchamiania programu przypisywane są do kolejnych adresów w pamięci. Taki sposób alokacji pamięci dla tablic pozwala na odnalezienie danego elementu tablicy za pomocą indeksów lub za pomocą wskaźników. Przykład (kod 2 w materiałach na stronie):

```c
#include <stdio.h>

int main()
{
    int tab[4]={20,40,50,100};
    int *wsk; //wskazuje na miejsce w pamieci
    wsk = &tab[0]; //przypisujemy adres na pierwszy element tab

    printf("Element pierwszy tablicy --> %d\n",*wsk); // i wyswietl zawartosc *wsk

    wsk = wsk + 1; // wez adres aktualny + 1
    printf("Element drugi tablicy --> %d\n",*wsk);

    wsk++;
    printf("Element trzeci tablicy --> %d\n",*wsk);

    wsk++;
    *wsk = 200;
    printf("Element czwarty tablicy --> %d\n",*wsk);

    return 0;
}
```

Zrealizuj poniższe zadania:

Zad 1. Początkowo zadeklaruj tablicę kilku elementów, niech rozmiar będzie stałą (np. const int size = 3), wyświetl dla każdego elementu: numer indeksu i wartość tablicy. Następnie chcemy mieć tablicę, która może przechowywać wskaźniki dla typu int. Zatem deklarujemy zmienna wskaźnikową ptr jako tablicę wskaźników dla typu int, o rozmiarze size (tak aby każdy element w ptr zawierał wskaźnik do wartości int). Przypisujemy adresy i wyświetlamy dla całej tablicy: numer indeksu i wartość elementu wskazywanego przez wskaźnik.

Zad 2. Wykorzystaj tablice z poprzedniego zadania, zadeklaruj wskaźnik i przypisz adres na pierwszy element tablicy. Następnie w pętli dla całej tablicy wyświetl adres i wartość elementu wskazywanego przez wskaźnik (wykorzystaj inkrementację wskaźnika ptr++).

Wskaźniki i funkcje, w języku C parametry do funkcji przekazywane są zawsze przez wartość. Co znaczy, że wewnątrz funkcji operujemy jedynie na kopiach zmiennych. Przekazując do funkcji zamiast zwykłych zmiennych wskaźniki do tych zmiennych, mamy możliwość modyfikacji oryginalnych wartości przechowywanych przez zmienne w ciele funkcji. Przykład (kod 3 w materiałach na stronie):

```c
#include <stdio.h>

void sum(int *k, int *l, int *s);

int main()
{
    int a,b;
    int suma;

    printf("Podaj liczbe a: ");
    scanf("%d",&a);

    printf("Podaj liczbe b: ");
    scanf("%d",&b);

    sum(&a,&b,&suma);
    printf("Suma: (%d + %d) = %d\n",a,b,suma);

    return 0;
}

void sum(int *k, int *l, int *s)
{
    *s = (*k + *l);
}
```

Zrealizuj poniższe zadanie:

Zad 3. Napisz funkcję swap, która zamienia element a i b (a na b, b na a).

Funkcja zwracająca wskaźnik, czyli funkcja zwracająca wskaźnik do funkcji wywołującej. W przypadku takich funkcji należy zachować szczególną ostrożność ponieważ zmienne lokalne funkcji NIE działają poza funkcją. Mają zasięg tylko wewnątrz funkcji. Przykład (kod 4 w materiałach na stronie):

```c
#include <stdio.h>

int *wieksza(int *a, int *b);

void main()
{
    int a = 7;
    int b = 77;
    int *p = wieksza(&a, &b);
    printf("%d jest wieksza",*p);
}

int *wieksza(int *a, int *b)
{
    if(*a > *b)
        return a;
    else
        return b;
}
```

Wskaźnik na wskaźnik, normalnie wskaźnik zawiera adres zmiennej. Gdy definiujemy wskaźnik na wskaźnik to pierwszy wskaźnik, zawiera adres drugiego wskaźnika, który wskazuje na adres zmiennej. Przykład (kod 5 w materiałach na stronie):

```
1
2  #include <stdio.h>
3
4  int main () {
5
6      int zmienna;
7      int *ptr;
8      int **pptr;
9
10     zmienna = 66;
11     ptr = &zmienna; //wskaznik wskazuje na adres zmiennej
12     pptr = &ptr; //wskaznik wskazuje na adres wskaznika, wskazujacego na adres zmiennej
13
14     printf("wartosc zmiennej: %d\n", zmienna );
15     printf("wartosc zmiennej na ktora wskazuje wskaznik *ptr: %d\n", *ptr );
16     printf("wartosc zmiennej na ktora wskazuje wskaznik na wskaznik **ptr %d\n", **pptr);
17
18     return 0;
19 }
```

3 Dynamiczna alokacja pamięci

Czasami z góry nie wiadomo ilu elementowa tablica będzie potrzebna. Deklaruje się wówczas zmienną wskaźnikową pożądanego typu i wywołuje funkcje alokacji pamięci (o żądanym rozmiarze). Funkcja ta zwraca adres pamięci. Należy to wpisać do wskaźnika na którym można później już normalnie działać (np. wskaznik[indeks] = wartosc;). Istotne jest aby pamiętać, że zaalokowaną pamięć należy zwolnić jeśli nie jest już potrzebna. Niezwolnione obszary pamięci, czytanie i zapisywanie pod niedozwolone adresy to częste i poważne błędy. Funkcje alokujące i zwalniające pamięć malloc() i free() w C zostały zastąpione w C++ przez operatory new oraz delete.

Dynamiczna alokacja pamięci, użycie funkcji malloc() pozwala bezpośrednio przydzielenie pamięci dla wskaźnika, deklaracja:

```
1  int *ptr;
2  ptr = (int*)malloc(n * sizeof(int));
```

Powyższy zapis może być jednolinijkowy:

```
1  int *ptr = (int*)malloc(n * sizeof(int));
```

gdzie:

- (int*) - rzutowanie na typ wskaźnika dla jakiego przydzielmy pamięć, ponieważ funkcja malloc zwraca zawsze void*
- (sizeof(int)) - argumentem dla funkcji malloc jest wielkość obszaru pamięci jaki chcemy przydzielić. Tu jest to rozmiar jednego inta, dlatego aby poprawnie przydzielić pamięć mnożymy przez n-elementów.

Przykład dynamicznej alokacji pamięci dla liczby całkowitej (kod 6 w materiałach na stronie):

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *ptr;
    ptr = (int*)malloc(sizeof(int)); //alokacja dla jednej liczby calkowitej

    if(ptr==NULL)
    {
        printf("\n Przydzielenie pamieci nie jest mozliwe");
        return 0;
    }

    printf("Podaj wartosc zmiennej: ");
    scanf("%d", ptr);
    printf("\nWartosc to: %d", *ptr);
    free(ptr);
    printf("\nWartosc po zwolnieniu pamieci: %d", *ptr);

    return 0;
}
```

Przykład dynamicznej alokacji pamięci dla n liczb całkowitych (kod 7 w materiałach na stronie):

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n;
    int sum = 0;
    printf("Ile elementow chcesz sumowac: ");
    scanf("%d", &n); //alokacja dla n liczb, wielkosc zdefiniowana przez uzytkownika

    int *ptr = (int*)malloc(n * sizeof(int)); //dynamiczna alokacja pamieci

    if(ptr == NULL) // zabezpieczenie
    {
        printf("Nie doszlo do alokacji pamieci.");
        return 0;
    }

    printf("Podaj elementy: "); //wczytanie elementow i sumowanie
    for(int i = 0; i < n; ++i)
    {
        scanf("%d", ptr + i);
        sum += *(ptr + i);
    }

    printf("Suma = %d", sum);

    free(ptr); //zwolnienie pamieci!


    return 0;
}
```

4 Zadania

Zad 1. Napisz program definiujący zmienną typu int oraz wskaźnik do zmiennej typu int. Program powinien wczytać z klawiatury wartość i podstawić ją do zmiennej stosując wskaźnik i operator adresu.

Zad 2. Na podstawie poniższego fragmentu kodu (fragment kodu w osobnym pliku) napisz program, który wyświetli tablicę "jakas_tablica". A następnie poprzez wskaźnik wsk element tablicy o indeksie 7 zostanie zmieniony na 77 i ponownie wyświetli tablicę.

```
1     int jakas_tablica[10];  //deklaracja tablicy
2     int *wsk;               // deklaracja wskaznika typu int
3     wsk = jakas_tablica;    // wsk wskazuje 1-szy element
4     wsk = &jakas_tablica[0]; // to jest dokladnie to samo
5                              // co instrukcja wczesniej
6     wsk++;      // teraz wskazujemy na 2-gi element tablicy
7     (*wsk)++; // a teraz ten 2-gi element zwiekszamy o 1
```

Zad 3. Zadeklaruj tablice 10-elementów i wyświetl dla każdego elementu jego adres (*ptr*) oraz wartość elementu wskazywanego przez wskaźnik (*ptr). Wartości wyświetlaj od największego indeksu do najmniejszego (wykorzystaj dekrementację wskaźnika ptr−−).

Zad 4. Wskaźniki można porównywać wykorzystując operatory relacyjne. Wykorzystaj zadanie 2 z punktu wskaźniki i tablice i zmodyfikuj je następująco: dopóki adres, na który wskazuje zmienna wskaźnikowa jest mniejszy lub równy ostatniemu elementowi tablicy to: wyświetlaj adres oraz wartość elementu wskazywanego przez wskaźnik a następnie zwiększ zmienną wskaźnikową.

Zad 5. Utwórz funkcję, która zwraca średnią arytmetyczną liczb. Funkcja ta jako parametry przyjmuje wskaźnik na tablicę i rozmiar tablicy.

Zad 6. Napisz program, który realizuje następujące funkcje dla tablic jednowymiarowych (alokowanych dynamicznie):

- Dodaje odpowiadające sobie elementy tablic – suma dwóch wektorów
- Mnoży odpowiadające sobie elementy tablic
- Wyznacza różnicę pomiędzy maksymalnym a minimalnym elementem tablicy