Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki
Katedra Elektrotechniki Przemysłowej i Automatyki

Informatyka 2

Wskaźniki. Dynamiczna alokacja pamięci

Instrukcja laboratoryjna
(wersja robocza)

Paweł Strączyński
2020

1. Operacje na wskaźnikach

Wskaźnik (ang. pointer) to zmienna zawierająca adres innej zmiennej. Każda zmienna poza swą nazwą i wartością przechowywaną posiada również swój adres, pod którym jest przechowywana w pamięci komputera. Na listingu 1.1 przedstawiono sposób tworzenia zmiennych wskaźnikowych. Deklaracja wskaźnika składa się z:

- typu zmiennej na która wskaźnik wskazuje (wskaźnik może być typu void jeżeli wskaźnik nie określa typu wskazywanej zmiennej),
- operatora * oznaczającego deklaracje zmiennej typu wskaźnikowego,
- nazwy zmiennej.

```
typ_wskazywanej_zmiennej * nazwa_wskaźnika;
```

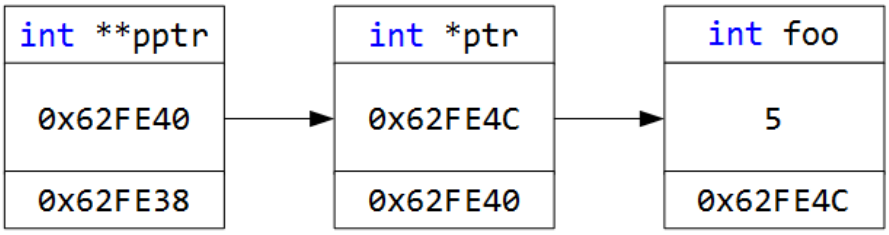
Listing 1.1. Tworzenie zmiennych wskaźnikowych

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

int main(int argc, char *argv[])
{
      int foo = 4; //Zmienna typu int
      int *ptr = &foo; //Wskaźnik na zmienna foo
      int **pptr = &ptr; //Wskaźnik na wskaźnik na zmienna foo
      printf("Wskaznik ptr zawiera adres zmiennej foo rowny: %p\n",ptr);
      printf("Wartosc zmiennej foo na która wskazuje ptr to: %d\n",*ptr);
      printf("Adres zmiennej wskaznikowej na która wskazuje pptr: %p\n",pptr);
      printf("Zawartosc zmiennej wskaznikowej ptr : %p\n",*pptr);
      printf("Zawartosc zmiennej foo: %d\n",**pptr);
      getchar();

      return 0;
}
```

Do utworzonej powyżej zmiennej wskaźnikowej ptr został przypisany adres zmiennej `foo` z wykorzystaniem operatora pobrania adresu `&`. Aby uzyskać dostęp do wartości zmiennej wskazywanej przez wskaźnik należy użyć operatora wyłuskania * przed nazwą zmiennej wskaźnikowej. Wskaźnik może wskazywać na inna zmienna wskaźnikową. Wskaźnik może być także zainicjowany wartością `0` lub `NULL` – wskaźnik pusty.



Rysunek 1.1. Wskaźniki w języku C

Deklarując zmienna wskaźnikową znak * można stawiać zarówno przy typie jak i nazwie zmiennej. Warto jednak stawiać znak * przy nazwie zmiennej gdyż pomoże to uniknąć ewentualnych pomyłek przy tworzeniu kilku zmiennych jednocześnie – listing 1.2.

Listing 1.2. Deklarowanie zmiennych wskaźnikowych

```
char *ptr1, *ptr2; //Deklaracja dwóch wskaźników na zmienną typu char
char* ptr3, ptr4; //ptr4 NIE JEST ZMIENNA WSKAŹNIKOWĄ!!!
```

2. Wskaźniki i tablice

W języku C istnieje silne powiązanie wskaźników z tablicami. Nazwa tablicy jest adresem pierwszego elementu tablicy. Zapis taki jak `&tab[0]` jest równoważny do `tab`. Podobnie zawartość elementu odczytywaną z użyciem operatora tablicowego `tab[0]` można równoważnie zapisać z wykorzystaniem wskaźników jako `*tab`. W tabeli 1 przedstawiono kilka przykładów równoważnych zapisów z wykorzystaniem operatora tablicowego `[]` i wskaźnika.

Tabela 2.1. Dostęp do elementów tablicy z użyciem wskaźników

Wskaźnik	**Operator []**
*(tab+1)	tab[1]
*(tab+i)	tab[i]
*tab+1	tab[0]+1
tab+5	&tab[5]

Przykładowe operacje na tablicach z użyciem wskaźników przedstawiono na listingu 2.1.

Listing 2.1. Dostęp do elementów tablicy z wykorzystaniem wskaźników

```
int i=0;
int tab[5] = {10, 20, 30, 40, 50};
```

```
printf("Zawartosc tablicy to:\n);
for(i=0;i<5; i++) {printf("->  %d\n:",*(tab+i))};
printf("Pierwszy element tablicy powiększony o 1 to:%d\n",*tab+1);
printf("Drugi element tablicy to:%d\n",*(tab+1));
```

3. Wskaźniki jako argumenty funkcji

Wskaźniki okazują się niezwykle przydatne jako argumenty funkcji. Podczas gdy do funkcji przekazywana jest zwykła zmienna to funkcja ta operuje jedynie na jej lokalnej kopii – wszelkie modyfikacje odbywają się na tej kopii właśnie. Przekazując zamiast zmiennych wskaźniki, funkcja również robi kopie ale tej zmiennej wskaźnikowej. Ta lokalna kopia wskaźnika wskazuje na oryginalną zmienną która można w ten sposób modyfikować. Po zakończeniu działania funkcji zatem zawartość zmiennej na która wskaźnik został przekazany jako argument funkcji zostanie zmodyfikowana – listing 3.1.

Listing 3.1. Wskaźnik jako argument funkcji

```
#include <stdio.h>

void fun1(int var1)
{
      var1=12;
}

void fun2(int *var1)
{
      (*var1)=12;
}

int main()
{
      int foo=1,bar=1;
      fun1(foo);
      fun2(&bar);
      printf("foo = %d",foo);
      printf("bar = %d",bar);
}
```

4. Dynamiczna alokacja pamięci

W język C przydzielanie pamięci odbywać się może na dwa sposoby:

- statycznie –stała ilość pamięci przydzielana przy deklaracji zmiennych, – zmienne istnieją od uruchomienia do zakończenia programu,

- dynamicznie – program alokuje i zwalnia dowolną ilość pamięci podczas swojej pracy.

Tworząc tablice statyczną na etapie jej deklaracji należy określić rozmiar, gdyż kompilator musi wygenerować program rezerwujący określona ilość pamięci. Programista podczas pisania programu musi przewidzieć jaki maksymalny rozmiar tablicy będzie potrzebny. Gdy okaże się, że zachodzi potrzeba przechowywania mniejszej liczby elementów to cześć statycznie alokowanej pamięci jest marnowana. Rozwiązaniem tego problemu jest wykorzystanie dynamicznej alokacji pamięci. W języku C istnieją w bibliotece `stdlib.h` istnieją dwie podstawowe funkcje służące do dynamicznego alokowania zadanego pamięci: `malloc()` oraz `calloc()`. Ich deklaracje przedstawiono na listingach 4.1 oraz 4.2.

Listing 4.1 Deklaracja funkcji calloc()

```
void *calloc(size_t n, size_t size );
```

Funkcja `calloc()` alokuje obszar pamięci o rozmiarze `n*size` i inicjuje go zerami. Funkcja zwraca wskaźnik do przydzielonego obszaru pamięci, natomiast gdy obszar pamięci nie może zostać przydzielony funkcja zwraca wartość `NULL`.

Listing 4.2 Deklaracja funkcji malloc()

```
void *malloc(size_t size );
```

Funkcja `malloc()` alokuje obszar pamięci o rozmiarze `size`. Pamięć nie zostaje zainicjowana.. Funkcja zwraca wskaźnik do przydzielonego obszaru pamięci, natomiast gdy obszar pamięci nie może zostać przydzielony funkcja zwraca wartość `NULL`.

Do dealokacji (zwalniania) obszaru pamięci służy funkcja `free()`. Funkcja zwalnia obszar pamięci wskazywany przez wskaźnik będący jej argumentem. Należy pamiętać o zwalnianiu nieużywanego obszaru pamięci. W przeciwnym przypadku może dojść do wycieku pamięci – nowa pamięć jest rezerwowana, a stara nie zostaje zwolniona. W rezultacie może dojść do braku wolnej pamięci. Należy również pamiętać aby nie zwalniać wielokrotnie tego samego obszaru pamięci. Przykładowy program tworzący tablicę dynamiczną przedstawiono na listingu poniżej.

Listing 4.3 Program operujący na dynamicznej tablicy

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

int main(int argc, char *argv[])
```

```
{
	int *tab;
	int i=0, n, sum=0;

	printf("Podaj ilosc liczb ktorych suma ma zostac wyznaczona:\n");
	scanf("%d",&n);
	tab = (int*)malloc(n*sizeof(*tab)); //lub sizeof(int)
	for(i=0; i<n; i++)
	{
		printf("\nPodaj %d liczbe:",i+1);
		scanf("%d",tab+i); //lub równoznacznie &tab[i]
	}
	for(i=0; i<n; i++)
	{
		sum+=tab[i]; //lub sum+=*(tab+i)
	}
	printf("Suma wprowadzonych liczb to %d",sum);
	free(tab);
	getchar();

	return 0;
}
```

Zadania do wykonania

Zadanie 1

Uruchomić i przeanalizować program z listingu 1.1. Utworzyć zmienną wskaźnikową `int ***ppptr` wskazującą na wskaźnik `int **pptr`. Przy pomocy wskaźnika `ppptr` wyświetlić na ekranie zawartość zmiennej `foo`.

Zadanie 2

Zmodyfikować przykład z instrukcji (listing 4.3) tak, aby wyznaczanie sumy elementów wykonywane było w postaci funkcji, która zwraca wynik operacji przez wskaźnik będący jej argumentem. Deklaracja funkcji powinna wyglądać następująco:

```
void SumTab(int *suma);
```

Wyświetlanie wyniku powinno znajdować poza funkcją `SumTab()`.

Zadanie 3

Napisać funkcję który stworzy n-elementowy wektor (dynamiczna tablica) liczb losowych, a następnie wyznaczy:

- wartość maksymalną,
- wartość minimalną.

Funkcja powinna zwracać wartość 0 lub 1 w zależności od tego czy alokacja pamięci zakończyła się powodzeniem. Wartości maksymalnego i minimalnego oraz elementu powinny być przekazywane przez wskaźnik. Deklaracja funkcji powinna wyglądać jak poniżej:

```
int MaxMin(int *max, int min);
```

Zadanie 4

Napisać funkcję która utworzy i wyświetli n-elementowy wektor (dynamiczna tablica) liczb z przedziału $< x1, x2 >$. Algorytm wyznaczania kolejnych elementów wektora jest następujący:

`x[k]=(k*(x2-x1) / (floor(n) - 1)) + x1` dla $k \in < 0, n-2 >$

`x[k] = x2` dla $k = n - 1$

Wskazówka:

Funckja `floor()` *znajduje się w bibliotece* `math.h`