Politechnika Świętokrzyska
w Kielcach
Wydział Elektrotechniki, Automatyki i Informatyki

Podstawy programowania 2

Instrukcja laboratoryjna 1

„Wskaźniki
i zmienne dynamiczne”

Przygotowali:
mgr inż. Paweł Pięta
dr inż. Arkadiusz Chrobot

Kielce, 2023

1 Wstęp

Celem zajęć jest powtórzenie wiadomości z przedmiotu Podstawy programowania 1 na temat wskaźników oraz rozszerzenie informacji na ich temat o podwójne wskaźniki, wskaźniki na funkcje i zmienne dynamiczne.

2 Wskaźniki

Zmienne wskaźnikowe, nazywane krótko wskaźnikami (ang. *pointers*), przechowują adresy zmiennych. Ich wartościami nie są zatem bezpośrednio dane, ale adres pojedynczej komórki pamięci lub adres pierwszej komórki z grupy przyległych komórek pamięci (ciągłego bloku pamięci). Są one deklarowane w języku C w następujący sposób:

```
typ_danych *nazwa_wskaźnika;
```

przy czym `typ_danych` określa typ zmiennych, które może wskazywać zmienna wskaźnikowa. Przykładowo, wskaźnik na zmienną typu `int` zostałaby utworzony tak:

```
int *integer_pointer;
```

Deklaracja zmiennej wskaźnikowej różni się zatem od deklaracji zwykłej zmiennej tym, że przed jej nazwą widnieje symbol `*` (asterysk). Wypisanie na ekran za pomocą funkcji `printf()` adresu przechowywanego we wskaźniku możliwe jest po zastosowaniu ciągu formatującego `"%p"`.

Wykonywanie podstawowych operacji na zmiennych wskaźnikowych odbywa się z użyciem dwóch jednoargumentowych operatorów:

- operatora *„wyłuskania”* adresu oznaczonego symbolem `&` – umożliwia on pobranie adresu komórki pamięci, w której przechowywana jest zmienna (w tym wskaźnikowa), lub w której rozpoczyna się ciągły obszar pamięci zajmowany przez zmienną,
- operatora dereferencji oznaczonego symbolem `*` – umożliwia on odczytanie wartości zmiennej wskazywanej przez wskaźnik lub zapisanie w niej nowej wartości,

oraz standardowego operatora przypisania `=`, przy czym:

- w celu modyfikacji wartości zmiennej wskaźnikowej, po lewej stronie znaku `=` należy umieścić nazwę wskaźnika, a po prawej adres zmiennej typu `typ_danych` (np. pobrany z użyciem operatora wyłuskania adresu),
- w celu modyfikacji wartości zmiennej wskazywanej przez wskaźnik, po lewej stronie znaku `=` należy wykonać dereferencję zmiennej wskaźnikowej, a po prawej umieścić nową wartość dla zmiennej wskazywanej.

Możliwe jest również zadeklarowanie zmiennej wskaźnikowej, która wskazuje na dane nieokreślonego typu:

```
void *void_pointer;
```

Danym wskazywanym przez taki wskaźnik można nadać konkretny typ stosując rzutowanie. Przykładowo, rzutowanie na typ `int` zostałoby wykonane tak:

```
(int *)void_pointer
```

Dereferencja tak zrzutowanego wskaźnika wyglądałaby następująco:

```
*(int *)void_pointer
```

Podczas deklaracji zmiennej wskaźnikowej będącej zmienną lokalną, dobrą praktyką jest nadanie jej wartości początkowej, aby zabezpieczyć się przed przypadkowym użyciem tzw. *dzikiego wskaźnika* (ang. *wild pointer*), czyli takiego, który zawiera adres przypadkowej komórki pamięci. W tym celu często wykorzystuje się stałą `NULL`, która oznacza wartość zerową wskaźnika – tak zainicjowany wskaźnik nosi miano pustego (ang. *null pointer*). Standard `ISO C99` języka C pozwala również na użycie w tej sytuacji bezpośrednio wartości 0. Drugim sposobem inicjacji wartości zmiennej wskaźnikowej jest oczywiście zapisanie w niej adresu jakiejś zmiennej, np. globalnej lub lokalnej. Wskaźniki zadeklarowane jako zmienne globalne są domyślnie wskaźnikami pustymi. Przykłady deklaracji i użycia zmiennych wskaźnikowych zostały przedstawione w kodzie źródłowym 1.

Zmienne wskaźnikowe, a właściwie zapisane w nich adresy, mogą być porównywane za pomocą tych samych operatorów, które używane są do porównywania np. liczb całkowitych. Wskaźniki można też od siebie odejmować, nie można ich natomiast dodawać. Dodatkowo język C umożliwia stosowanie tzw. *arytmetyki wskaźników* – wskaźnik można zinkrementować lub zdekrementować, co spowoduje przesunięcie adresu w bajtach o rozmiar typu danych zmiennej wskazywanej. Do zmiennej wskaźnikowej można też dodać lub odjąć liczbę całkowitą, co zadziała jak wielokrotna inkrementacja lub dekrementacja. Arytmetyka wskaźników może zostać wykorzystana np. w obsłudze tablic.

Jeśli w programie zaistnieje potrzeba, aby z użyciem zmiennej wskaźnikowej wskazywać na inną zmienną wskaźnikową, to wówczas należy skorzystać z tzw. *podwójnego wskaźnika* (ang. *double pointer*), czyli wskaźnika na wskaźnik. Sposób deklarowania takiej zmiennej wygląda następująco:

```
typ_danych **nazwa_wskaźnika;
```

Dereferencja takiego wskaźnika daje w rezultacie pojedynczy wskaźnik. Podwójna dereferencja zwróci natomiast wartość zmiennej wskazywanej. Tego typu zmienne znajdują zastosowanie w sytuacji, gdy konieczna jest modyfikacja wartości wskaźnika wewnątrz funkcji. Wówczas zmienna wskaźnikowa powinna zostać przekazana do funkcji przez podwójny wskaźnik. Przykłady deklaracji i użycia podwójnych wskaźników zostały przedstawione w kodzie źródłowym 2.

Kod źródłowy 1: Przykłady deklaracji i użycia zmiennych wskaźnikowych

```c
#include <stdio.h>

struct point_3d
{
    float x, y, z;
};

int main()
{
    int var = 1;
    int *ptr_1 = &var;
    int *ptr_2 = NULL;
    struct point_3d point = {1.0f, 2.0f, 3.0f};
    struct point_3d *ptr_3 = &point;
    void *ptr_4 = NULL;

    ptr_2 = &var;
    *ptr_2 = 2;

    printf("ptr_1 = %p\n", ptr_1);
    printf("var = %d\n", *ptr_2);
    printf("null pointer = %p\n", ptr_4);
    return 0;
}
```

Kod źródłowy 2: Przykłady deklaracji i użycia podwójnych wskaźników

```c
#include <stdio.h>

// Wyzerowanie wskaźnika wewnątrz funkcji.
// Przykład użycia wskaźnika na wskaźnik (podwójnego wskaźnika).
void null_int_ptr(int **ptr)
{
    *ptr = NULL;
}

int main()
{
    int var = 1;
    int *ptr_1 = &var;
    int *ptr_2 = &var;
    int **double_ptr = &ptr_2;

    printf("ptr_1 = %p\n", ptr_1);
    printf("ptr_2 = %p\n", ptr_2);

    null_int_ptr(&ptr_1);
    null_int_ptr(double_ptr);

    printf("ptr_1 = %p\n", ptr_1);
    printf("ptr_2 = %p\n", ptr_2);

    return 0;
}
```

3 Wskaźniki na funkcje

Funkcje, tak samo jak dane, są umieszczane w komórkach pamięci operacyjnej komputera, a zatem mogą być one wskazywane przez zmienne wskaźnikowe. Wskaźniki na funkcje deklarujemy w języku C w następujący sposób:

```
typ_zwracanych_danych (*nazwa_wskaźnika)(lista_parametrów);
```

Zapis ten różni się od deklaracji funkcji jedynie nawiasami okrągłymi występującymi przy nazwie zmiennej wskaźnikowej oraz obecnością przed nazwą symbolu `*`. Przykładowo, wskaźnik na funkcję, która przyjmuje dwa argumenty wywołania typu `int` i zwraca wartość tego samego typu, zostałby utworzony tak:

```
int (*function_pointer)(int, int);
```

Tak jak wcześniej omówione zmienne wskaźnikowe, również wskaźniki na funkcje mogą być inicjowane stałą `NULL`. Ponieważ nazwa funkcji jest traktowana w języku C jako wskaźnik, przypisanie do wskaźnika na funkcję adresu funkcji odbywa się w następujący sposób:

```
function_pointer = nazwa_funkcji;
```

Powyższy zapis można dosłownie interpretować jako przypisanie nazwy funkcji do zmiennej wskaźnikowej. Składniowo wywołanie funkcji z użyciem wskaźnika nie różni się niczym od bezpośredniego wywołania funkcji – należy bowiem skorzystać z operatora wywołania funkcji `()`:

```
function_pointer(lista_parametrów);
```

Więcej przykładów deklaracji i użycia wskaźników na funkcje zostało przedstawionych w kodzie źródłowym 3. Warto wspomnieć, że wskaźniki na funkcje mogą być przechowywane w strukturach, jak również w tablicach, a ponadto możliwe jest przekazywanie ich do funkcji jako argumentów wywołania.

Kod źródłowy 3: Przykłady deklaracji i użycia wskaźników na funkcje

```
1  #include <stdio.h>
2
3  int multiply(int a, int b)
4  {
5      return a*b;
6  }
7
8  void print(int value)
9  {
10     printf("Value = %d\n", value);
11 }
12
```

```c
int main()
{
    int a = 2, b = 3;
    int (*multiply_func_ptr)(int, int) = multiply;
    void (*print_func_ptr)(int) = NULL;
    print_func_ptr = print;

    int result = multiply_func_ptr(a, b);
    print_func_ptr(result);
    return 0;
}
```

4 Zmienne dynamiczne

Oprócz zmiennych globalnych i lokalnych, trzecim rodzajem zmiennych, które mogą zostać użyte w programie, są zmienne dynamiczne. Są one tworzone i niszczone dynamicznie podczas działania programu (zgodnie z intencją programisty), a przechowywane są w obszarze pamięci nazywanym stertą (ang. *heap*). Funkcje języka C umożliwiające przydzielanie (alokację) pamięci dla zmiennych dynamicznych i jej zwalnianie (dealokację) to:

```c
void *malloc(size_t size);
void *calloc(size_t num, size_t size);
void *realloc(void *ptr, size_t new_size);
void free(void *ptr);
```

Ich dokładny opis został przedstawiony w tablicy 1. Po zwolnieniu obszaru pamięci zmiennej dynamicznej, dobrą praktyką jest nadanie wskaźnikowi wartości `NULL`, aby zabezpieczyć się przed przypadkowym użyciem tzw. *wiszącego wskaźnika* (ang. *dangling pointer*), czyli takiego, który zawiera adres zdealokowanego bloku pamięci.

Przykłady wykorzystania zmiennych dynamicznych zostały przedstawione w kodzie źródłowym 4. Program prezentuje dynamiczną alokację pamięci, przekazywanie zmiennych dynamicznych do funkcji, obsługę dynamicznych tablic z zastosowaniem indeksów i arytmetyki wskaźników, a także zwalnianie pamięci i wykorzystanie wskaźników na wskaźniki (podwójnych wskaźników).

Kod źródłowy 4: Przykłady wykorzystania zmiennych dynamicznych

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

// Wypisanie adresu przechowywanego przez wskaźnik.
void print_address(void *ptr, const char *what, const char *name)
{
    printf("Address of dynamic %s '%s' = %p\n", what, name, ptr);
}
```

```
9
10 // Zwolnienie obszaru pamięci i wyzerowanie wskaźnika.
11 // Przykład użycia wskaźnika na wskaźnik (podwójnego wskaźnika).
12 void free_ptr(void **ptr)
13 {
14     free(*ptr);
15     *ptr = NULL;
16 }
17
18 // Wypisanie dynamicznej tablicy z użyciem indeksów.
19 void print_array(int *array, size_t elements)
20 {
21     size_t i;
22     printf("[");
23     for (i=0; i<elements-1; i++)
24         printf("%d, ", array[i]);
25     printf("%d]\n", array[i]);
26 }
27
28 // Wypisanie dynamicznej tablicy z użyciem
29 // pomocniczego wskaźnika i operatora dereferencji.
30 void print_array_alt(int *array, size_t elements)
31 {
32     int *ptr = NULL;
33     printf("[");
34     for (ptr=array; ptr<array+elements-1; ptr++)
35         printf("%d, ", *ptr);
36     printf("%d]\n", *ptr);
37 }
38
39 int main()
40 {
41     // Przydział pamięci dla zmiennej dynamicznej funkcją malloc().
42     int *var = (int *)malloc(sizeof(int));
43     if (NULL != var)
44     {
45         print_address((void *)var, "variable", "var");
46         printf("Variable 'var' = %d\n", *var);
47         free_ptr((void **)&var);
48     }
49     else
50         printf("Dynamic variable 'var' could not be allocated.");
51
52     // Przydział pamięci dla dynamicznej tablicy funkcją calloc().
53     const size_t elements = 5;
54     int *array_1 = (int *)calloc(elements, sizeof(int));
55     if (NULL != array_1)
56     {
57         print_address((void *)array_1, "array", "array_1");
58         printf("Dynamic array 'array_1':\n");
59         print_array(array_1, elements);
60
61         // Zmiana rozmiaru pamięci
62         // dynamicznej tablicy funkcją realloc().
63         const size_t elements_2 = 2*elements;
64         int *array_2 = (int *)realloc((void *)array_1,
65                                       elements_2*sizeof(int));
66         if (NULL != array_2)
```

```
67          {
68              // Zmiana rozmiaru obszaru pamięci powiodła się, dlatego
69              // oryginalny, przestarzały wskaźnik należy wyzerować.
70              array_1 = NULL;
71              print_address((void *)array_2, "array", "array_2");
72              printf("Dynamic array 'array_2':\n");
73              print_array_alt(array_2, elements_2);
74              free_ptr((void **)&array_2);
75          }
76          else
77          {
78              printf("Dynamic array 'array_1' could not be reallocated.");
79              free_ptr((void **)&array_1);
80          }
81      }
82      else
83          printf("Dynamic array 'array_1' could not be allocated.");
84
85      return 0;
86  }
```

5 Zadania

Uwaga! Programy należy napisać z podziałem na funkcje z parametrami oraz nie można w nich korzystać ze zmiennych globalnych.

1. Napisz program, który będzie liczył sumę ciągu arytmetycznego i geometrycznego. Obliczanie sum zaimplementuj w osobnych funkcjach, które będziesz wywoływał za pośrednictwem wskaźników.

2. Zaimplementuj funkcję `allocate_array()`, której zadaniem będzie przydzielenie w sposób dynamiczny pamięci na tablicę liczb typu `float`. Funkcja powinna:

   - przyjąć przez parametr żądaną wielkość tablicy (liczbę jej elementów),
   - zwrócić wartość typu `bool`, która będzie sygnalizowała, czy przydział pamięci powiódł się, czy nie,
   - zwrócić wskaźnik na utworzoną tablicę poprzez parametr – w tym celu zastosuj parametr, który jest wskaźnikiem na wskaźnik, czyli ma np. taką postać: `float **array`.

   Zaprezentuj użycie powyższej funkcji w przykładowym programie, jak również wykorzystanie utworzonych z jej pomocą dynamicznych tablic.

3. Napisz program, w którym utworzysz macierz dynamiczną, wypełnisz ją liczbami całkowitymi należącymi do przedziału $[-10, 10]$ oraz wypiszesz na ekran zawartość tej macierzy i sumę elementów leżących na jej przekątnej. Program przed zakończeniem pracy powinien usunąć macierz. Wskazówki znajdziesz w książce B. W. Kernighana i D. M. Ritchie'go.

Nazwa funkcji	Opis
`malloc()`	Przydziela niezainicjowany obszar pamięci o rozmiarze `size`-bajtów. Jeśli `size` ma wartość 0, zachowanie zależne jest od implementacji. W razie powodzenia zwraca wskaźnik do początku nowo zaalokowanego bloku pamięci. Przydzielony obszar pamięci musi zostać zwolniony za pomocą funkcji `free()`. W razie niepowodzenia zwraca `NULL`.
`calloc()`	Przydziela obszar pamięci dla tablicy `num`-elementów (każdy rozmiaru `size`-bajtów) i inicjuje go wartościami zerowymi. Jeśli `size` ma wartość 0, zachowanie zależne jest od implementacji. W razie powodzenia zwraca wskaźnik do początku nowo zaalokowanego bloku pamięci. Przydzielony obszar pamięci musi zostać zwolniony za pomocą funkcji `free()`. W razie niepowodzenia zwraca `NULL`.
`realloc()`	Zmienia rozmiar obszaru pamięci wskazywanego przez `ptr`, który został uprzednio przydzielony przez `malloc()`, `calloc()` lub `realloc()` i nie został jeszcze zwolniony przez `free()`, na `new_size`-bajtów. Realokacja wykonywana jest na dwa sposoby: 1) rozszerzenie/zmniejszenie bloku pamięci pod adresem `ptr` (jeśli jest to możliwe), 2) zaalokowanie nowego obszaru pamięci, skopiowanie do niego danych ze starego bloku, a następnie zwolnienie wskaźnika `ptr`. Jeśli nie ma wystarczającej ilości pamięci, stary blok nie jest zwalniany. Jeśli `ptr` ma wartość `NULL`, zachowanie jest takie samo jak wywołanie `malloc(new_size)`. Jeśli `new_size` ma wartość 0, zachowanie zależne jest od implementacji (funkcja może zadziałać jak `free()`, albo zwrócony zostanie `NULL`). W razie powodzenia zwraca wskaźnik do początku nowo zaalokowanego bloku pamięci. Przydzielony obszar pamięci musi zostać zwolniony za pomocą funkcji `free()`. Oryginalny wskaźnik `ptr` staje się przestarzały (ang. *deprecated*) i nie powinien być już używany (nawet jeśli realokacja pamięci nastąpiła w miejscu). W razie niepowodzenia zwraca `NULL`, a oryginalny wskaźnik `ptr` pozostaje sprawny.
`free()`	Zwalnia obszar pamięci wskazywany przez `ptr`, który został uprzednio przydzielony przez `malloc()`, `calloc()` lub `realloc()`. Jeśli `ptr` ma wartość `NULL`, nie zostaje wykonana żadna operacja. Jeśli `ptr` ma wartość różną od wartości zwróconej wcześniej przez `malloc()`, `calloc()` lub `realloc()`, albo blok pamięci został już wcześniej zdealokowany, zachowanie jest nieokreślone. Po zakończeniu działania wskaźnik `ptr` staje się przestarzały i nie powinien być już używany. Funkcja nie zeruje zwalnianego obszaru pamięci ani wskaźnika `ptr`.

Tablica 1: Funkcje języka C do obsługi zmiennych dynamicznych

4. Napisz program, który będzie wczytywał do dynamicznej tablicy liczby całkowite tak długo, aż użytkownik wprowadzi liczbę 0. Początkowo przyjmij, że tablica będzie mieściła jedynie 5 takich liczb. Jeśli użytkownik będzie próbował wprowadzić więcej liczb, niż może pomieścić tablica, to program powinien wykryć taką sytuację i zwiększyć jej rozmiar o kolejne 5 elementów. Program powinien powtarzać opisane w poprzednim zdaniu czynności zawsze wtedy, kiedy będzie kończyło się

miejsce w tablicy, oraz do momentu, aż użytkownik nie skończy wprowadzać liczb. Po tym wszystkim program powinien wypisać na ekran zawartość tablicy i zwolnić przydzieloną na nią pamięć.

5. Napisz program, który utworzy dwudziestoelementową tablicę liczb typu `float`. Tę tablicę powinien wypełniać użytkownik, wprowadzając wartości z klawiatury. Program powinien też zawierać funkcję, która będzie posiadała parametr typu wskaźnik na funkcję, oprócz innych parametrów, które uznasz za niezbędne. Funkcja ta będzie w pętli przeglądała tablicę i jeśli wspomniany parametr wskaźnikowy nie będzie pusty, to będzie wywoływała funkcję przez niego wskazywaną i przekazywała jej jako argument wywołania bieżący element tablicy. Wskaźnik ten, w zależności od wyboru użytkownika, powinien wskazywać albo na funkcję wypisującą wartość pojedynczego elementu na ekran, albo zapisującą ją do pliku (decyzja podejmowana jest tylko raz, przed wywołaniem funkcji).