Wskaźniki, tablice, funkcje...

Wojciech Myszka

19 kwietnia 2020

Streszczenie

To już piąty odcinek. I końca nie widać. Trzymajcie się zdrowo.
Dziś Instrukcja Laboratoryjna Nr 9 o tematyce zbliżonej do tego wpisu.

Spis treści

Wstęp

Wskaźniki są wredne. Wierzcie mi.

Oto prototyp prostej funkcji:

```
int funkcja(int, int *)
```

Co można o niej powiedzieć?

- zwraca wartości całkowite;
- ma dwa parametry:
  - pierwszy to wyrażenie całkowite,
  - drugi to wskaźnik na `int`.

Natomiast można sobie wyobrazić dwie zupełnie różne realizacje funkcji **zgodne** z tym prototypem:

Funkcja operująca na tablicy

```
int funkcja(int n, int * t)
{
    int suma = 0;
    for (int i = 0; i < n; i++)
        suma += t[i];
    return suma;
}
```

Bardzo prosta funkcja, wyliczająca sumę wszystkich elementów `n`-elementowej tablicy całkowitej. Dla przypomnienia `suma += t[i];` jest równoważne `suma = suma + t[i];`.

Użycie funkcji

```
int main()
{
    int x[10] = {1, 3, 12, 7, 128, -15, -1, 0, -22, 1000};
    printf("Suma elementów tablicy: %d\n", funkcja(10, x));
    return 0;
}
```

Funkcja zwracająca dwie wartości

Jak wiadomo, polecenie `return` może być użyte do „wyprowadzenia” z wnętrza funkcji jedynie wartości jednej zmiennej.

```
int funkcja(int x, int * k)
{
    *k = 0;
    if ( x <= 0 )
        return 0;
    else
        while ( x != 1 )
        {
            ( *k )++;
```

```
            if ( x % 2 == 0 )
                x = x / 2;
            else
                x = 3 * x + 1;
        }
    return 1;
}
```

Powyższy program to realizacja interesującego zagadnienia zwanego problemem Collatza. Był on omawiany przeze mnie na zajęciach z Technologii informacyjnych (ósmy zestaw slajdów: Zapis algorytmów (Algorytmy, Część II)). Program poprawnie działa tylko dla wartości `x` dodatnich i zwraca wartość 0 (fałsz) gdy `x` jest niedodatnie; 1 gdy argument jest poprawny. Drugi argument to liczba iteracji, którą trzeba wykonać aby program się zakończył.

Użycie funkcji

```
int main ()
{
    int n;
    if (funkcja(12135, &n))
        printf("Liczba iteracji: %d\n", n);
    else
        printf("Złe parametry!\n");
    return 0;
}
```

(Tak na marginesie: trzeba było 249 iteracji...)

Podsumowanie

Celem tych przykładów było uświadomienie, że wskaźnik może być użyty do przekazania adresu tablicy albo jakiejś zwykłej zmiennej: wskaźnik jest taki sam, ale jego sens i użycie zupełnie inne.

Tablice wielowymiarowe

Język C pozwala na deklarowanie tablic wielowymiarowych:

```
int T[10][20];
```

Tablica (dwuwymiarowa) `T` ma 10 wierszy i 20 kolumn.

Podobnie zadeklarujemy tablicę trójwymiarową:

```
int Tablica[10][20][30];
```

Przekazanie tablicy do funkcji

Tablica jednowymiarowa

```
int funkcja (int, int *);
int main()
{
    int t[100]
//  ...
    funkcja(100, t)
}
```

Możemy również zorganizować to inaczej:

```
int funkcja (int, int []);
int main()
{
    int t[100]
//  ...
    funkcja(100, t)
}
```

Wywołując funkcję wpisujemy **nazwę** tablicy, czyli **adres** jej początku.

Zapis `int []` jest poprawny, sugeruje, że drugim parametrem funkcji jest tablica czyli wskaźnik...

Ostatecznie mamy następujące możliwości zadeklarowania funkcji mającej tablicę jako jeden z parametrów:

1. `int funkcja(int n, int t[n])`
2. `int funkcja(int n, int t[ ])`
3. `int funkcja(int n, int * t)`

Tablica wielowymiarowa

Skupimy się na dwu wymiarach. Pamiętamy cały czas, że

Najprawdopodobniej powinna zadziałać taka konstrukcja:

```
int Funkcja(int m, int n, int T[m][n]);
```

Trzecim jej argumentem jest tablica `T` o `m` wierszach i `n` kolumnach. Do funkcji **musimy** przekazać informację o rozmiarach tablicy.

W (opisanym wcześniej) przypadku tablic jednowymiarowych mieliśmy dwa równoważne (niewskaźnikowe) zapisy `int t[n]` i `int t[ ]`. Czy coś takiego jest możliwe w przypadku tablic dwuwymiarowych? I tak i nie. Pominięcie rozmiaru tablicy w przypadku tablic jednowymiarowych nie jest problemem — język C nie sprawdza czy granice tablicy nie zostały przekroczone. Wystarczy znajomość początku (i zaufanie do programisty, że granic tablicy nie przekroczy).

W przypadku tablicy dwuwymiarowej, organizacja dostępu do poszczególnych elementów tablicy **wymaga** znajomości długości wiersza, bo miejsce elementu w pamięci można wyliczyć tylko gdy zna się długość wiersza. Tablica (4 wiersze, 4 kolumny) o następujących wartościach:

	0	1	2	3
0	00	01	02	03
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33

zapisana będzie w pamięci tak:

0	1	2	3	4	5	6	7	8	9	10	...	15
00	01	02	03	10	11	12	13	20	21	22	...	33

Zatem, jedynie dopuszczalna, deklaracja tablicy dwuwymiarowej będącej parametrem funkcji wygląda tak:

```
int Funkcja(int m, int n, int T[ ][n])
```

(możemy pominąć jedynie pierwszy wymiar).

Skrajnie uproszczony prototyp będzie wyglądał tak:

```
int Funkcja(int, int, int [][5]);
```

Należy podać liczbę kolumn. Można to zrobić też tak:

```
int Funkcja(int , int n, int [ ][n])
```

(Jeżeli chcemy, żeby liczba kolumn mogła być **zmienną** — należy **wcześniej** tę zmienną zadeklarować...)

Przeglądając Internet można znaleźć różne porady jak zadeklarować tablicę dwuwymiarową. Pewna ich część sugeruje, żeby samemu zorganizować obsługę tablicy dwuwymiarowej używając jednowymiarowej i (niezbyt skomplikowanych, ale jednak) dodatkowych przeliczeń na indeksach.

Aby skorzystać z funkcji `Funkcja()` program może wyglądać tak:

```
int main ()
{
    int Tablica[20][30];
//  ...
    x = Funkcja(20, 30, Tablica);
//  ...
}
```

Wywołując funkcję wpisujemy jedynie **nazwę** tablicy, czyli do funkcji przekazujemy **adres** jej początku.

Dynamiczna alokacja pamięci

Tablica jednowymiarowa

Jak wszyscy wiedzą, na potrzeby organizacji tablic, istnieje możliwość przydziału pamięci „od systemu operacyjnego". Działą to jakoś tak:

```
#include<sdtlib.h>
// ...
int main()
{
    int * T;
    T = malloc(200);
}
```

Deklarujemy zmienną `T` typu wskaźnik na `int`. Funkcja `malloc()` przydziela 20 **bajtów** z zasobów pamięci operacyjnej komputera (czyli tworzymy tablicę o 50 elementach, gdyż każdy element tablicy `int` to 4 bajty.)

Później z tablicy korzystamy „w sposób normalny".

Czyli gdy chcemy zadeklarować tablicę o `N` elementach, bardziej kompletny program będzie wyglądał tak:

```
#include <stdlib.h>
#include <stdio.h>
int main()
{
    int N = 50; // Wartość zmiennej N można nadać w dowolny sposób
    int * T;
    T = malloc(N * sizeof(int));
    if (T == NULL)
    {
        printf("Nie mogłem zadeklarować tablicy!\n");
        return 1;
    }
// tu dalsze obliczenia z użyciem tablicy
// skorzystamy ze zdefiniowanej wcześniej funkcji
    x = funkcja(N, T);
}
```

Sprawdzenie czy przypadkiem `T` nie jest równe `NULL` (wskaźnik o wartości zero) ma nas ochronić przed sytuacją, w której system operacyjny nie mógł przydzielić pamięci. Funkcja `malloc()`

informuje nas o tym zwracając wartość zero.

Tablica dwuwymiarowa

Można podobnie również w przypadku tablicy dwuwymiarowej. Ale wróćmy najpierw do „klasycznej" definicji. Co znaczy `int t[20]`? rezerwujemy obszar pamięci do przechowania 20 wartości `int`

Co zatem znaczy `int T[20][30]`? Czy można temu nadać taką interpretację, że deklarujemy najpierw tablicę o 20 elementach przechowującą informacje o adresach początków dwudziestu, trzydziesto-elementowych tablic jednowymiarowych?

Tym w istocie jest tablica dwuwymiarowa: każdy jej wiersz to tablica jednowymarowa...

Czy zatem można odpytać o adres drugiego wiersza tablicy?

```
int main()
{
    int T[20][30];
//  ...
    printf("T[2]=%ld\n", (long int)T[2]);
    printf("T[2]=%p\n", T[2]);
}
```

1. Deklaruję tablicę
2. Drukuję wartość `T[2]` dokonując konwersji (rzutowania) na `long int`. (Adresy w komputerze są liczbami 64. bitowymi).
3. Drukuję tę wartość jako wskaźnik (`%p`).

Wynik programu wygląda jakoś tak:

```
T[2]=140732971692768
T[2]=0x7ffef2c932e0
```

(Każde uruchomienie generuje inne wartości. Pierwsza liczba to wartosć adresu zapisana dziesiętnie, druga szesnastkowo.)

Czy zatem można zmajstrować sobie tablicę dwuwymiarową korzystając z funkcji `malloc()`?

Najpierw musimy *zmajstrować* **jednowymiarową** tablicę wskaźników (o dwudziestu elementach[1])

```
int * T[20];
```

(tablica ma dwadzieścia elementów (`[20]`), a jej wartościami są wskaźniki na `int` (`int *`)).

Teraz musimy poprosić system operacyjny, aby utworzy dla nas dwadzieścia tablic jednowymiarowych, po trzydzieści elementów; ich adresy wpiszemy do kolejnych komórek `T`:

```
for (int i = 0; i < 20; i++)
```

[1] Konsekwentnie nawiązuję do przykładu

```
    T[i] = malloc(30 * sizeof(int));
```

Teraz trudne pytanie pomocnicze o typ wskaźnika wskazującego na tablicę wskaźników?

Gdy tablica jest typu `int` to jest łatwo: `int *`. Teraz tablica jest typu `int *` zatem wskaźnik na jej początek to `int **`.

Czyli jeżeli dodatkowo zadeklarujemy:

```
int ** Tprim;
```

to będziemy mogli napisać

```
Tprim = T;
```

czyli do `Tprim` wstawić adres początku `T` i powinno być, tak, że `Tprim[i] == T[i]`.

Jeżeli pójdziemy krok dalej i użyjemy funkcji `malloc()` do stworzenie dwudziestoelementowej tablicy `T`, to dostaniemy gotowca do deklaracji tablic dwuwymairowych:

```
int M = 20;
int N = 30;
int ** T;
T = malloc(M * sizeof(int *)); // przydzielamy pamięć na wskaźnik!
for(int i = 0; i < 20)
    T[i] = malloc(N * sizeof(int)) // teraz przydzielamy pamięć na liczbę
```

Dodatkowo po każdym użyciu funkcji `malloc` sprawdzić trzeba czy nie zwróciła on a wartości zero! Gdy tak się zdarzy — program nie może być kontynuowany.

Tablica dwuwymiarowa i funkcja

Używając tak utworzonej tablicy w funkcji musimy zadbać o to żeby była zgodność typów parametrów w definicji funkcji i wywołaniu funkcji.

W funkcji o prototypie

```
inf Funkcja(int, int, int [][30]);
```

trzeci parametr jest typu `int (*)[30]` stworzona powyższą metodą tablica/wskaźnik `T` jest typu `int **`. Jedno nie pasuje do drugiego. Dobra wiadomość jest taka, że można sobie zmajstrować taką funkcję:

```
int FunkcjaPrim(int, int, int **);
```

i wszystko powinno być ok.

Zła wiadomość jest taka, że trzeba się zdecydować na jeden sposób deklaracji tablic wielowymiarowych:

- albo automatyczny (`int T[20][30]`) ,
- albo dynamiczny (znacznie dłuższy i bardziej zagmatwany).

Zaleta dynamicznego deklarowania pamięci

Wyobraźmy sobie taki prosty bardzo program:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv)
{
    for (long int i = 1; ; i = i * 2)
    {
        printf("i=%ld\n", i);
        double  t[i];
        t[i - 1] = 0.;
    }
    return 0;
}
```

Nie robi on nic specjalnego tylko w każdym obrocie pętli deklaruje coraz to większą tablicę i nadaje jej ostatniemu elementowi wartość zero.

Uruchomienie tego programu kończy się tak:

```
i=1
i=2
i=4
i=8
i=16
i=32
i=64
i=128
i=256
i=512
i=1024
i=2048
i=4096
i=8192
i=16384
i=32768
i=65536
i=131072
i=262144
i=524288
i=1048576
Segmentation fault (core dumped)
```

Jak widać największa tablica, którą udało się zadeklarować miała 524288 elementów. Całkiem to

sporo, ale...

Gdy nasz program będzie wyglądał tak:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv)
{
    for (long int i = 1; ; i = i * 2)
    {
        double * t = malloc(sizeof(double) * i);
        if (t == NULL)
        {
            printf("brak pamieci, i=%ld\n", i);
            return 1;
        }
        t[i - 1] = 0.;
        free(t);
    }
    return 0;
}
```

to znaczy za każdym razem tablica deklarowana jest dynamicznie, a następnie pamięć jest zwalniana, program kończy się tak:

```
brak pamieci, i=2147483648
```

Zadeklarowana tabela może być znaaaaacznie dłuższa.

Zadanie na ten tydzień

Zadanie na ten tydzień jest dosyć proste: macie Państwo stworzyć program, który:

1. Tworzy tablicę dwuwymiarowa o zadanej liczbie kolumn i wierszy;
2. Wypełnia ją danymi (odradzam wczytywanie — będzie to męczące; już lepiej wypełniać ją automatycznie)
3. Program uzupełniają trzy funkcje:
    1. wyliczająca średnią z **całej** tablicy,
    2. wyliczająca średnią z zadanego wiersza,
    3. wyliczająca średnią z zadanej kolumny.
4. Program powinien też wydrukować tablicę wejściową oraz wyniki. Te wydruki są po to, żeby można było sprawdzić działanie...

Podstawą jest instrukcja laboratoryjne numer 9: Tablice i funkcje.

Tekst w postaci pliku PDF...

...jest również dostępny.