Metody Numeryczne II A
Porady dotyczące programowania

mgr Jakub Narębski

2 kwietnia 2001 roku

1 Stosowanie tablic

1.1 Przekazywanie tablic jako parametrów funkcji

W języku C tablice to właściwie wskaźniki. Tablice wielowymiarowe to tak naprawdę tablice tablic. Jedyne co C potrzebuje by indeksować elementy tablicy (móc się odnosić do jej poszczególnych elementów), to znajomość typu elementów tablicy. Powoduje to, że przy przekazywaniu tablic do funkcji możemy nie podawać ostatniego wymiaru macierzy. W wywołaniach funkcji argument przekazywany jest jako wskaźnik. Możemy zatem pisać

```
void func(int n, float a[],...)
```

i przekazywać do funkcji tablice o dowolnej wielkości. Co więcej, możemy do funkcji przekazywać tablice *alokowane dynamicznie*, czyli odpowiedni wskaźnik (patrz rozdział 1.2).

Przekazując tablice wielowymiarowe do funkcji musimy podawać wszystkie wymiary z wyjątkiem pierwszego, np.

```
void func(float a[][N],...)
```

Typy **float** arr[M][N] i **float** **ppf *nie są* równoważne, pomimo że do elementów, po odpowiedniej inicjalizacji (patrz rozdział 1.2) można się odwoływać arr[i][j] i ppf[i][j]. W szczególności tak zdefiniowana zmienna arr jest tablicą tablic. arr[i] jest N–elementową tablicą, zaś ppf[i] jest wskaźnikiem.

Jeśli zatem chcemy przekazywać statyczne tablice dwuwymiarowe (macierze) o różnym rozmiarze do funkcji musimy się uciec do pewnego tricku. (Trick ten jest często stosowany w procedurach napisanych Fortranie.) Należy jako argument przekazywać macierz o drugim wymiarze będącym maksymalnym, jakiego będziemy używać w programie, t.j. np.

```
void func(int m, int n, float arr[][MAXN],...).
```

Oczywiście wypełniać będziemy tylko interesującą nas część macierzy.

1.2 Tablice alokowane dynamicznie

1.2.1 Alokowanie tablic jednowymiarowych

Pamięć na tablice jednowymiarowe alokujemy za pomocą funkcji calloc bądź malloc, a zwalniamy za pomocą funkcji free. Deklaracje tych funkcji znajdują się w `stdlib.h`. Powinno się zawsze sprawdzać, czy przydzielenie pamięci się powiodło. Pisanie po nie przydzielonej pamięci może powodować naprawdę dziwne efekty: od zrzucania core po losowe zachowanie programu!

Przykładowy szablon do alokowania tablicy jednowymiarowej liczb zmiennoprzecinkowych wygląda następująco

```
#include <stdlib.h>
...
void main(int argc, char *argv[])
{
  float *vect;
  ...
  N = 1024; /* czy jakikolwiek sposób ustalenia rozmiaru */
  ...
  vect = (float *) calloc (N, sizeof(float ));
  if(vect == NULL) {
    printf("Nie udało się zaalokować pamięci na wektor vect.\n");
    /* tutaj zwalniamy wszystkie dotychczas zaalokowane wektory */

    exit (1); /* kończymy program */
  }
  ...
  free (vect );

  return 0;
}
```

Zamiast `calloc (N, sizeof(float))` do przydzielenia (alokacji) pamięci można użyć `malloc(N*sizeof(float))`. Ta druga funkcja *nie zeruje* przydzielanej pamięci, stąd może być szybsza (o ile nie potrzebujemy wyzerować przydzielonej pamięci).

Istnieje także inna forma zapisu operacji przydzielenia pamięci. Zamiast pisać

```
vect = (float *) calloc (n, sizeof(float ));
```

stosując jawne rzutowanie i korzystając z typu elementów wektora można zapisać

```
vect = calloc (n, sizeof(*vect ));
```

korzystając z niejawnego rzutowania. Zapis ten ma tę zaletę, że przy zmianie typu elementów wektora nie trzeba zmieniać kodu programu. Oczywiście można by to samo uzyskać definiując nazywając typ danych, np. pisząc

```
typedef float real;
```

i deklarując wektor o elementach typu `real`. Stosowanie jednego z tych dwóch sposobów zapisu jest kwestią gustu. Można stosować dowolną z konwencji, byle *konsekwentnie*.

1.2.2 Przechowywanie macierzy w jednowymiarowym wektorze

Najprostszym sposobem *zaalokowania* tablicy wielowymiarowej jest zrobić ją jako tablice jednowymiarową o liczbie elementów będącej iloczynem wymiarów macierzy wielowymiarowej. Jeśli zatem chcemy mieć tablicę dynamiczną odpowiadającą macierzy $M \times N$, to należy wykonać

```
arr = (float *) calloc (M*N, sizeof(float));
```

(Oczywiście, tak samo jak w poprzednim przypadku, należy sprawdzić, czy przydzielenie pamięci się powiodło.)

Prostota przydzielania i zwalniania pamięci okupiona jest trudniejszym niż w przypadku tablic statycznych dostępem do elementu tablicy. Stosując konwencję C, t.j. ostatni indeks zmienia się najszybciej, do elementu arr_{ij} odwołujemy się następująco: arr[i*N + j].

1.2.3 Tablica wskaźników na tablice

Aby móc się posługiwać notacją jak dla wielowymiarowych tablic statycznych musimy postępować inaczej. Deklarujemy zmienną jako wskaźnik na wskaźnik na liczbę rzeczywistą, a dokładniej jako tablicę wskaźników na wiersze. Za cenę bardziej skomplikowanego przydzielania i zwalniania pamieci możemy się odwoływać do elementu arr_{ij} poprzez arr[i][j]. Tak zaalokowana tablica zajmuje więcej miejsca niż wektor jednowymiarowy o odpowiedniej liczbie elementów.

Dodatkową zaletą tego sposobu jest możliwośc stosowania bardzo dużych tablic; tak dużych że nie można przydzielić na nie pamięć w jednym kawałku.

Przykładowy szablon do alokowania tablicy dwuwymiarowej liczb zmiennoprzecinkowych wygląda następująco

```
#include <stdlib.h>
...
void main(int argc, char *argv[])
{
  float **arr;  /* będzie to odpowiednik tablicy m na n */
  int i, j;
  ...
  M = N = 1024; /* czy jakikolwiek sposób ustalenia rozmiaru */
  ...
  arr = (float **) calloc (M, sizeof(float *));
  if(arr == NULL) {
    printf("Nie udało się przydzielić pamięci na macierz arr.\n");
    exit(1);
  }

  for(i = 0; i < M; i++) {
    arr[i] = (float *) calloc (N, sizeof(float));
    if(arr[i] == NULL) {
      printf("Nie udało się przydzielić pamięci na wiersz %i macierzy arr.\n", i);
      /* zwalniamy dotychczas zaalokowane wiersze */
      /* oraz wskaźnik na tablicę wkaźników na wiersze */
      for(j = i-1; j >=0 ; j--)
        free(arr[i]);
      free(arr);
      exit(1);
    }
  }
  ...
  /* najpierw zwalniamy wiersze, później samą macierz */
  for(i = 0; i < M; i++)
    free(arr[i]);
  free(arr);
```

```
  return 0;
}
```

Oczywiście jak zwykle przed wyjściem z programu należy zwolnić *wszystkie* dotyczas zaalokowane tablice.

Dodatkową zaletą tego sposobu przechowywania tablic wielowymiarowych jest prostota zamiany miejscami dwu wierszy tablicy. Wystarczy napisać

```
  float *tmp;

  tmp    = arr[i];
  arr[i] = arr[j];
  arr[j] = tmp;
```

1.2.4 Wektor danych i tablica „indeksująca”

Kosztem dodatkowej zmiennej (i dodatkowej pamięci, podobnie jak w poprzednim podrozdziale) możemy mieć jednocześnie łatwe przydzielanie i zwalnianie pamięci oraz łatwy dostęp do elementów. Dokonujemy tego trzymając dane w jednowymiarowym wektorze i trzymając dodatkowo wektor wskaźników na początkowe elementy wierszy.

Szablon postepowania wygląda nastepująco:

```
#include <stdlib.h>
...
void main(int argc, char *argv[])
{
  float **arr; /* tutaj trzymamy wskaźniki na wiersze */
  float *arr_data; /* tutaj trzymamy elementy tablicy */
  ...
  M = N = 1024; /* czy jakikolwiek sposób ustalenia rozmiaru */
  ...
  arr = (float **) calloc (M, sizeof(float *));
  if(arr == NULL) {
    printf("Nie udało się zaalokować pamięci na wektor indeksujący dla arr.\n");
    exit(1);
  }

  arr_data = (float *)calloc(m*n, sizeof(float));
  if(arr_data == NULL) {
    printf("Nie udało się zaalokować pamięci na dane na macierz arr.\n");

    free(arr); /* zwalniamy przydzieloną do tej pory pamięć */
    exit(1);
  }

  /* arr[i] ma wskazywać na i-ty wiersz macierzy */
  for(i = 0; i < m; i++)
     arr[i] = &arr_data[i*n]; /* arr_data + i*n */
  ...
  /* zwalniamy wektor danych */
  free(data);
  /* zwalniamy wektor indeksujący */
```

```
    free(arr);

  return 0;
}
```

Dzięki pomocniczemu wektorowi wskaźników arr możemy się do elementu arr_{ij} odwoływać za pomocą arr[i][j], jak również za pomocą arr_data[i*n + j] (niektóre funkcje w bibliotekach numerycznych wymagają takiego sposobu przekazywania tablic).

Zamiany wierszy miejscami nie można jednak robić tak prosto, jak przy stosowaniu sposobu z poprzedniego rozdziału.

1.3 Przekazywanie tablic dynamicznych do funkcji

Uwaga: w rozdziale tym przyjmujemy w przykładach deklaracji funkcji, że funkcja nie zwraca żadnych wyników jako swojej wartości (czyli jest typu **void**). Często jest tak, że nawet jeśli funkcja zwraca wynik za pomocą swoich argumentów to jako swój wynik daje kod błędu, zwracając zero w razie powodzenia, a wartość różną od zera (identyfikującą przyczynę niepowodzenia) w przeciwnym przypadku. Do funkcji przekazujemy także wymiary tablicy. Przyjmujemy że wymiary i tablica przekazywane są w przykładach jako pierwsze argumenty; symbol '...' oznacza pozostałe argumenty funkcji.

Jeśli chcemy przekazać do funkcji tablicę dynamiczną *jednowymiarową* (o pamięci przydzielonej dynamicznie jak w rozdziale 1.2.1), stosujemy następującą deklarację funkcji

```
void func1(int n, float vect[],...)
```

Zamiast pisać **float** vect[] moglibyśmy używać **float** *vect. Ten pierwszy sposób jest lepszy ponieważ ukazuje nasze intencje: vect ma być jednowymiarową tablicą, do elementów której będziemy się odwoływać za pomocą notacji indeksowej, tzn. do element $vect_i$ wektora poprzez vect[i]. Do tak zadeklarowanej funkcji możemy także przekazywać wektory statyczne (zadeklarowane jako **float** v[N]) o dowolnym rozmiarze.

Jeżeli chcemy przekazać do funkcji tablicę dwuwymiarową przechowywaną w wektorze jednowymiarowym (jak w rozdziale 1.2.2), to piszemy

```
void func2(int m, int n, float *arr,...)
```

Jeżeli do elementu macierzy arr_{ij} odwołujemy się za pomocą arr[i*n+j] (tzn. wiersz po wierszu, tak jak przechowywane są tablice w C), to do funkcji możemy przekazywać tablice dwuwymiarowe statyczne. Jeśli mamy w programie macierz zadeklarowaną jako **float** arr[M][N] to w wywołaniu funkcji piszemy

```
func2(M, N, &arr[0][0], ...);
```

Przy stosowaniu sposobu z rozdziału 1.2.4 możemy przekazywać do takiej funkcji wektor zawierający elementy tablicy arr_data.

Jeśli chcemy przekazać do funkcji tablicę dwuwymiarową, przechowywaną jako tablica wskaźników na wiersze (jak w rozdziale 1.2.3), stosujemy następującą deklarację argumentów funkcji

```
void func3(int m, int n, float **arr,...)
```

Do tak zadeklarowanej funkcji *nie można* przekazywać tablic wielowymiarowych zadeklarowanych statycznie. Przy stosowaniu sposobu z rozdziału 1.2.4 możemy do takiej funkcji przekazywać wektor „indeksujący” arr, zawierający wskaźniki do wierszy.

2 Wskaźniki do funkcji

W języku C istnieją wskaźniki na funkcje. Warto korzystać z tej właściwości, gdyż przyczynia się ona do większej czytelności kodu i umożliwia jego łatwiejszą modyfikację.

2.1 Przekazywanie funkcji jako argumentu

Czasami potrzebujemy przekazać funkcję jako jeden z argumentów. Chcemy np. wypełnić tablicę wartościami zadanej funkcji czy rozwiązać równanie nieliniowe zadane jakąś funkcją. Możemy to zrobić za pomocą wskaźników do funkcji. Niech przykładowa funkcja wypełnia tablicę wartościami innej funkcji. Można to zrobić w następujący sposób:

```
void filltable (int n, double v[],
                double (*func)(double x),
                double min, double max)
{
  int i;
  double scale = (max - min)/n;

  for(i = 0; i < n; i++)
    v[i] = (*func)(i*scale - min);
}
```

Następnie taką funkcję można wywołać np. w ten sposób

```
  filltable (N, x, &sin, 0, 2*M_PI);
```

wypełniając wektor x o rozmiarze N wartościami funkcji sinus dla wartości argumentu od 0 do 2π. Ponieważ nazwa funkcji jest równoważna wskaźnikowi do niej, w wywołaniu można pominąć operator pobrania adresu & (część kompilatorów pisze nawet, że operator & jest zbędny).

Można także zdefiniować typ, będący wskaźnikiem do funkcji określinego rodzaju, a następnie używać go w deklaracjach argumentów funkcji. Przykładowo definicja typu funkcji będącej prawą stroną równania (układu równań) różniczkowego zwyczajnego wygląda następująco

```
typedef void (*ode_rhs)(float t, float u[], float dudt[]);
```

(nie przekazujemy tutaj rozmiaru tablicy gdyż dla konkretnych zagadnień jest to zazwyczaj niepotrzebne: rozmiar jest zdefiniowany przez zadane zagadnienie). Używamy go następnie w definicji funkcji obliczającej następny krok metody różnicowej rozwiązywania RRZ. Przykładowo nagłówek funkcji dla metody Eulera mógłby wyglądać następująco

```
void Euler_step(float t, float dt,
                int n, float u[], float unew[],
                ode_rhs derivs)
```

2.2 Tablica wskaźników na funkcje

Jeżeli chcemy zgrupować kilka funkcji w jedną całość (np. kolejne metody wielokrokowe rozwiązywania równań różniczkowych zwyczajnych czy też kilka różnych równań), możemy to zrobić za pomocą tablicy wskaźników do funkcji. W najprostszym przypadku możemy to zrobić w następujący sposób (zakładając, że chcmy mieć w tablicy funkcje o jednym argumencie rzeczywistym, zwracające liczbę rzeczywistą)

```
float (*funcvect[N])(float x);
```

Zazwyczaj jednak chcemy np. razem z funkcją zgrupować dane o niej, np. rząd metody czy opis funkcji. Można to zrobić w następujący sposób:

```
/* najpierw definiujemy jakie funkcje chcemy przechowywać w tablicy */
typedef double (*FUNCTION)(double x);

/* następnie definiujemy strukturę zawierającą funkcje i informacje o niej */
typedef struct FunctionInfoTag
{
  FUNCTION func; /* funkcja */
  char *desc; /* jej opis */
  ...
} FUNCINFO;

/* definiujemy tablicę funkcji i informacji o nich, od razu ją inicjalizując */
/* korzystamy z tego, że jeśli inicjalizujemy w definicji tablicy, */
/* to nie musimy podawać jej rozmiaru: kompilator potrafi go sam wyznaczyć */
FUNCINFO table_of_functions[] =
{{ sin, "sin(x)" },
 { cos, "cos(x)" }};

/* stała określająca rozmiar tej tablicy */
#define ToF_SIZE (sizeof(table_of_functions)/sizeof(table_of_functions[0]))

/* makrodefinicja do wyznaczania rozmiaru tablicy jednowymiarowej */
#define ARRAY_SIZE(arr) (sizeof(arr)/sizeof(arr[0]))
```

Pominęliśmy tutaj operatory pobrania adresu &, zgodnie z faktem, że „goła” nazwa funkcji oznacza wskaźnik do niej.

Jeżeli chcemy wybrać np. pierwszą funkcję z tej tablicy, piszemy

```
table_of_functions[0].func
```

(stosujemy operator wybrania składowej struktury).

3 Funkcje o wartościach wektorowych

Jeżeli chcemy napisać funkcję, która zwraca wektor (np. przy rozwiązywaniu równań rózniczkowych zwyczajnych, czy też licząc dyskretną transformatę Fouriera), to chyba najwygodniej jest zwrócić wektor wynikowy jako jeden z argumentów. Przykładowa deklaracja funkcji wyglądała by wtedy następująco:

```
void func(int n, float v[], float res[]);
```

Jeśli funkcja potrzebuje wektora pomocniczego do wyliczenia wektora wynikowego z wektora wejściowego, można skorzystać z argumentu wynikowego (np. res w powyższym przykładzie). Jest to zazwyczaj lepsze rozwiązanie niż napisanie funkcji, która zwraca wektor wynikowy w wektorze wejściowym (nadpisując wektor wejściowy) i alokuje dodatkową pamięć na wektor pomocniczy.