Увод в програмирането

Лекция 6:

Масиви

Мотивация

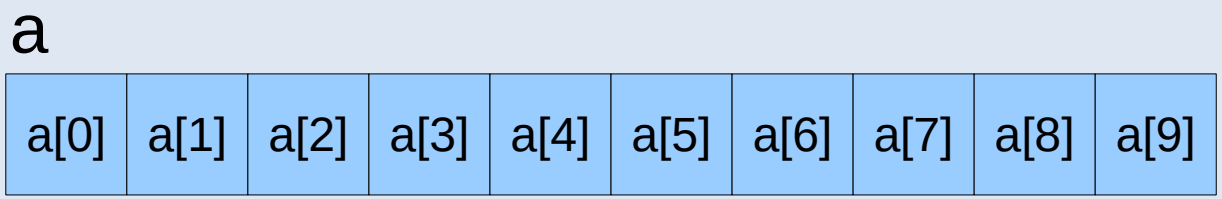
- Искаме в променлива/и да запишем редица от стойности (от еднакъв тип)
  - Напр. списък с факултетни номера в дадена група (ако приемем, че са 5-цифрени)
- С познатите средства това е почти невъзможно:
  - По една променлива за всеки ф.н. - **`int fn1, fn2, fn3, fn4, fn5, fn6, fn7;`** – не можем да ги обходим с цикъл, трябва всяка поотделно
  - Да използваме отделните цифри на един int – можем да запишем само най-кратки редици

Решение

- Едно от най-често използваните решения на описания проблем са *масивите*
- Масивът е крайна редица от елементи от един и същ тип
- Всеки елемент има пореден номер (*индекс*)
- Отделните елементи на даден масив се достъпват по индекс

Масиви в C/C++ - деклариране

- Основен синтаксис: <тип> <име>[<брой>];
- Квадратните скоби този път не са от метаезика на Бекус-Наур, а се изписват в програмата
- Пример: **`int a[10];`** - масив **`a`** с 10 целочислени елемента



Деклариране на масив – брой на елементите

- Броят на елементите, който се указва в квадратните скоби, трябва да бъде константен израз
  - т.е. не може да зависи от стойностите на променливи
- Примери:

```
int array[16];
const int COUNT = 5;
double pairs[COUNT * 2];
```

- Допълнителен материал: компилаторът gcc позволява използването и на променливи, но това е друг стандарт и няма да се приема на контролните

Представяне в паметта

- Заеманата от един масив памет е равна на размера на един елемент, умножен по броя на елементите
  - Понеже са от един и същ тип
- Пример: **`double grades[10];`** → 80 байта
- Съществува оператор sizeof,
  - с който може да се провери колко байта заема даден тип или дадена променлива
  - **`cout << sizeof(short) << " " // 2`**
    `<< sizeof(grades); // 80`

Достъпване на елемент на масив (1)

- С **<име>[индекс]** достъпваме конкретен елемент на даден масив
- При масив с *n* елемента първият е с индекс **0**, а последният – с **(*n* – 1)**
  - т.е. за **`int a[5]`** индексите са 0, 1, 2, 3 и 4
- В квадратните скоби може да се посочи произволен целочислен израз

Достъпване на елемент на масив (2)

- С получения елемент можем да правим всичко, каквото можем и с познатите ни от преди това променливи от същия тип, напр.:

```
int a[5];
cin >> a[0];
int i = 1;
cout << a[i – 1];
a[2] = 10 + a[0] * 3;
a[1] = 0;
cout << a[a[1] + 2];
```

- Съвсем естествено е да се използват цикли за обхождането на всички или част от елементите на масив

Достъпване на елемент на масив (3)

- Ако се опитаме да достъпим елемент на позиция извън границите на масива,
  - Напр. ако на упражнения ровим във Фейсбук, вместо да слушаме асистента, може да не разберем, че индексите започват от 0 и т.н.
- Ще се получи грешка (бъг) – грешен резултат или направо crash на програмата
- ```
  int array[10];
  cout << array[10];
  cout << array[9999999];
  ```
  ```

Основни действия с масиви

Действие	**double d**	**double array[3]**
Инициализация	**double d = 3.14159;**	**double array[3] = {1, 2, 3};**
Присвояване на стойност	**d = 0.5;**	~~**array = {9, 8, 7};**~~ ~~**array = array2;**~~ **// не може**
Отпечатване на екрана	**cout << d;**	**cout << array; // отпечатва странна стойност**
Прочитане от клавиатурата	**cin >> d;**	~~**cin >> array;**~~ **// не може**

- Каква беше разликата между инициализация и присвояване?
- По-късно (в лекцията за указатели) ще разгледаме защо не може да се извършва присвояване на стойност (след като вече е деклариран масивът), отпечатване със cout и т.н.

Деклариране на масив – подробен синтаксис

<тип> <идентификатор>[[<константа>]]
[= { <константа> {, <константа> } }] ;

Примери:

- `bool b[10], c[11], d; // два масива b и c;`
  `// стойностите им са неопределени`
- `double x[3] = {0.5, 1.5, 2}; //нач. стойност`
- `int a[] = {3 + 2, 2 * 4}; //щом има начална`
  `// стойност, може да се пропусне броят`
  `↔ int a[2] = {5, 8};`
- `float f[4] = {2.3, 4.5}; // даваме стойност само на част от елементите – другите са нули`
  `↔ float f[4] = {2.3, 4.5, 0, 0};`

Масиви – често извършвани операции (примери)

- Въвеждане от клавиатурата на масив с *n* стойности – дробни числа, където $0 \leq n \leq 100$

```
double array[100]; int count;
cin >> count;
for (int i = 0; i < count; i++)
    cin >> array[i];
```

  - Допълнителна променлива за реалния брой елементи
- Отпечатване на въведения масив
- Копиране на масив
- Проверка дали два масива имат едни и същи елементи

Масиви – още операции (1)

- Сума на елементите на масив
- Сума само на положителните елементи
- Да се провери дали масивът съдържа отрицателен елемент
- Да се провери дали всички елементи в масива са неотрицателни
- Да се намери индексът, на който се намира елемент с дадена стойност. Ако не се намери такъв, да се отпечата -1. Ако се съдържа на няколко места, да се намери първото срещане
  - Втори вариант: последното срещане

Масиви – още операции (2)

- Да се изтрие елемент от масив по зададен индекс (елементите след него да се избутат наляво)
- Да се вмъкне елемент на указана позиция
- Да се намерят стойността и индексът на минималния елемент
- Да се сортира масивът
  - Допълнителен материал: има тонове алгоритми за сортиране; ние ще използваме сортиране с пряка селекция

Многомерни масиви

- Елементите на един масив могат да бъдат други масиви
  - (с еднаква дължина)
- Може да имаме двумерни, тримерни, …, 10-мерни и т.н. масиви
  - Не е нужно да можем да си ги представяме геометрично

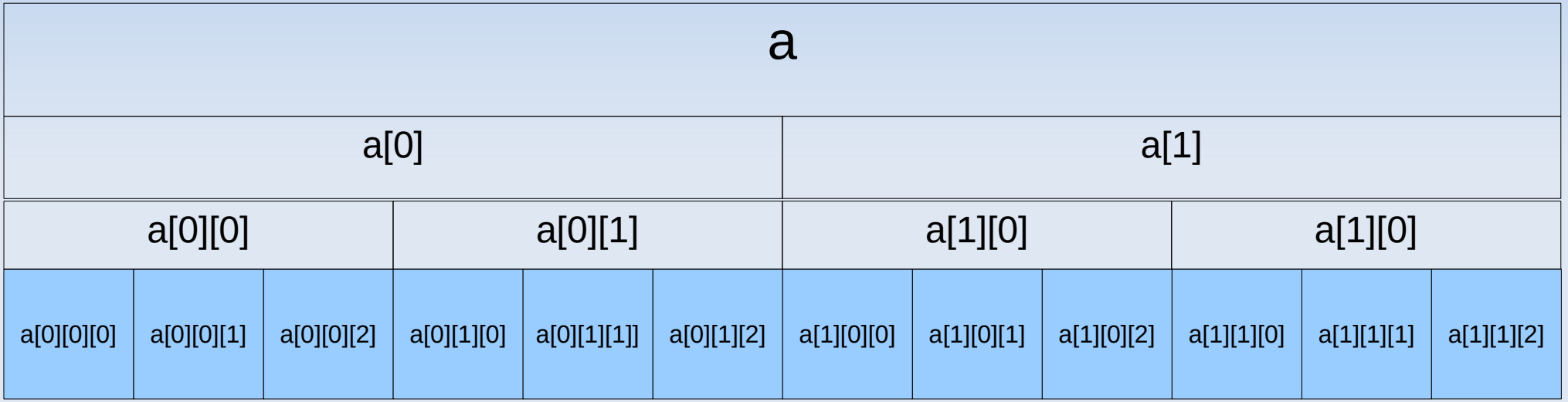
Многомерни масиви – дефиниране

- <тип> <идентификатор>[<константа>] {[<константа>]} = {<константа> {,<константа>}}
- Примери:

```
int a[2][3] = {{1, 2, 3}, {4, 5, 6}};
double b[5][6] = {0.1, 0.2, 0.3, 0.4}; // вътрешните
                                       // скоби не са задължителни
int c[4][5] = {{1, 2}, {3, 4, 5, 6}, {7, 8, 9}, {10}};
float f[][2][3] = {{{1.2, 2.3, 3.4}, {4.5, 5.6, 6.7}},
                   {{7.8, 8.9, 9.1}, {1.2, 2.3, 3.4}},
                   {{5.6}, {6.7, 7.8}}};
```

- Грешен пример: double b[5, 6];

Физическо представяне



Задачи за двумерни масиви (матрици)

- Въвеждане и извеждане на матрица
- Транспониране на квадратна матрица
- Намиране на еднакви стълбове
- Различни обхождания на квадратна матрица – елементи над главния диагонал и т.н.
- Търсене на седловидна точка

Допълнителен материал

- Масивът е линейна структура. Как да представим нелинейна структура, например дърво (като структурата от директориите в една файлова система) или мрежа от градове и пътища между тях?
- Добавянето и изтриването на елемент в масив може да е бавно, няма ли по-хубав начин?
- Има ли начин да представим множество от елементи така, че търсенето (а също и добавянето и изтриването) да е по-бързо от двоичното търсене?
- Отговори на тези въпроси може да получите в курса по *структури от данни* и програмиране (СДП)

Обобщение

- Масив: крайна редица от елементи от един и същ тип
- Всеки елемент има индекс; при масив с *n* елемента първият е с индекс 0, а последният – с (*n* – 1)
- <тип> <име>[<размер>]; размерът е константа
- Достъп до елемент - <име>[<индекс>]
- Не може присвояване, cout, cin – трябва цикъл
- Многомерни масиви – напр. int m[10][20][30];


KEEP
CALM
AND ASK
STUPID
QUESTIONS


DON'T
WORRY,
OUR STAFF IS
ACCUSTOMED TO
DUMB QUESTIONS