درس «مبانی کامپیوتر و برنامه‌سازی»

آرایه

صادق علی‌اکبری

سرفصل مطالب

- مفهوم آرایه
- نحوه تعریف آرایه
- نحوه استفاده از آرایه‌ها
- کاربردها
- الگوریتم‌های مهم در زمینه آرایه

مفهوم آرایه (Array)

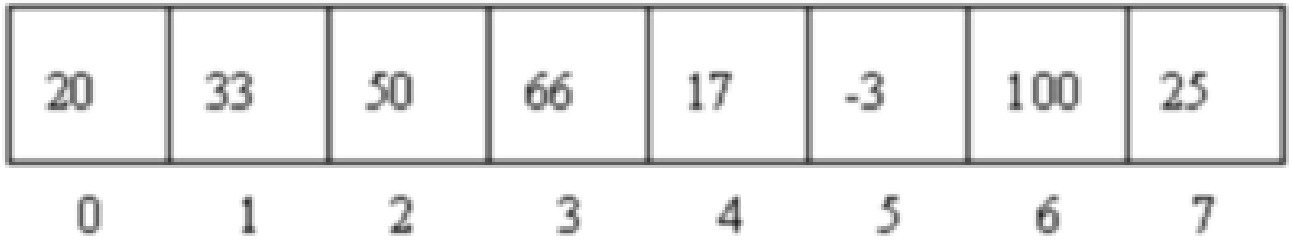
- گاهی به تعدادی متغیر نیازمندیم
  که همگی از یک نوع هستند و یک اطلاعات مشخص را نشان می‌دهند
- مثلاً مجموعه نمرات درس مبانی برنامه‌نویسی
  - تعدادی متغیر از نوع double که همگی در کنار هم مجموعه نمرات را نشان می‌دهند
- برای این کار، مفهوم **آرایه** (array) در زبان‌های برنامه‌نویسی وجود دارد
- با کمک آرایه، یک متغیر تعریف می‌کنیم که مانند ظرفی شامل تعداد زیادی متغیر است
- مثال:

```
double grades[38];
```

- یعنی grades یک آرایه شامل ۳۸ متغیر double است

آرایه (Array)

- **آرایه**: مجموعه‌ای از داده‌های مرتبط به هم
  - همه این داده‌ها، **از یک نوع داده** هستند
- **طول آرایه** ثابت است: طول آن یک بار و در زمان ایجاد، معین می‌شود
  - امکان کم و زیاد کردن اندازه (طول) آرایه وجود ندارد
- یک آرایه مثل مجموعه‌ای از متغیرها است
  - هر متغیر در آن یک عنصر (**element**) نامیده می‌شود



آرایه

- کل آرایه، یک **نام** دارد
- همه مقادیر موجود در آرایه، یک نوع واحد دارند
  - مثلاً همه int هستند یا همه double هستند
- عناصر آرایه به همان ترتیب در حافظه جای داده می‌شوند
  - همه خانه‌های آرایه (عناصر یا elements) پشت سر هم در حافظه جای می‌گیرند
- آرایه: گروهی از محل‌های متوالی حافظه که دارای نام و نوع یکسانی هستند

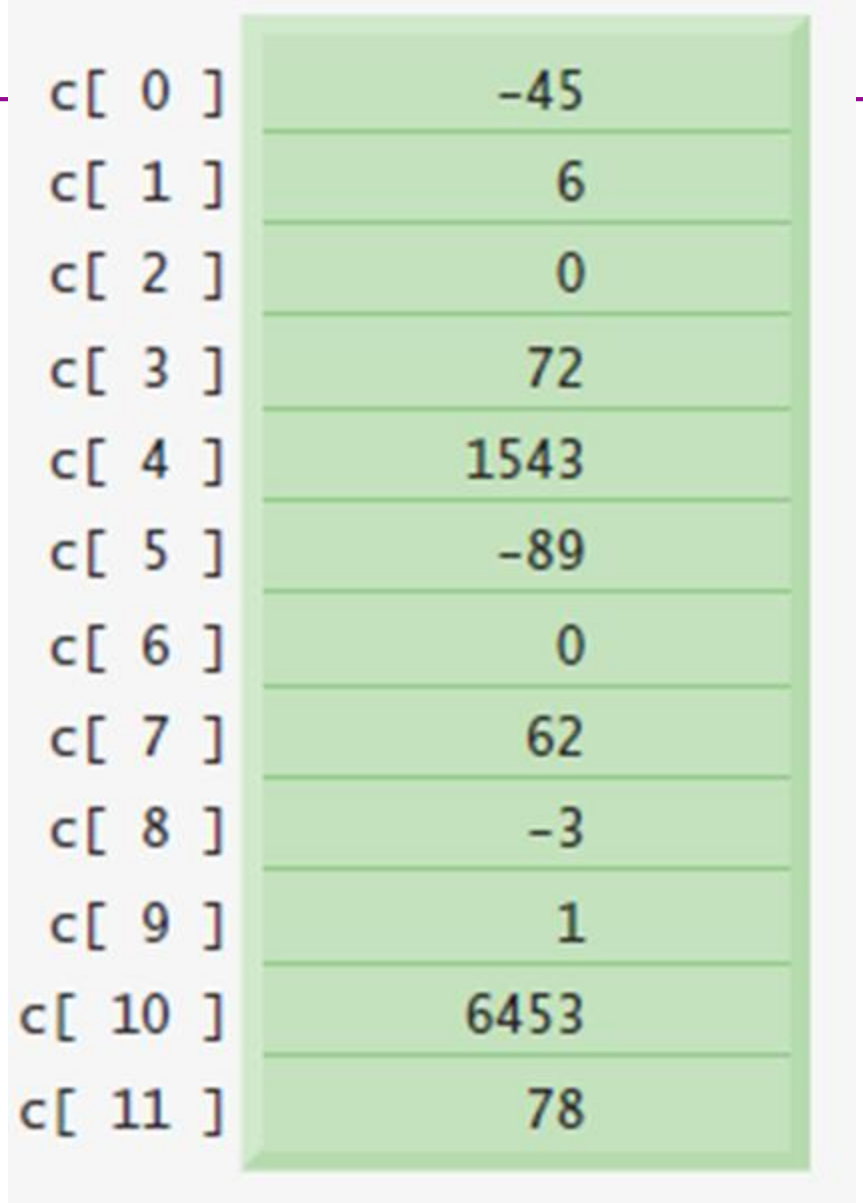
اندیس (index یا subscript)

- شماره ترتیبی هر عنصر در آرایه، **اندیس** (index) نامیده می‌شود
- نکته مهم: شماره خانه‌های آرایه (اندیس) از **صفر** شروع می‌شود (نه از یک)

  خانه‌های یک آرایه به طول n : خانه صفرم تا خانه n-1 اُم
- برای دستیابی به یک خانه از آرایه، باید نام و شماره خانه را داشته باشیم
- مثال: `char a[10];`

آرایه

```
int c[12];

c[0 ] = -45;
c[1 ] =6;
...
c[11] = 78;
```



- نام آرایه
- طول آرایه
- اندیس
- عنصر (element)

مثال

```
int ages[3];
ages[0]=19;
ages[1]=20;
ages[2]=19;

char grades[4];
grades[0]=grades[1]='A';
grades[2]='B';
grades[3]='C';

bool conditions[2];
conditions[0] = true;
conditions[1] = false;
```

مثال

```
int e=1;
double grades[38];
grades[0] = 18.25;
grades[1] = 9.0;
int f = grades[0];
if(grades[0] == grades[1])
      e=2;
else if(grades[e] == e)
      e=3;
else if(grades[2*e+1] == 2*e+1)
      e=4;
e = grades[2*e];
grades[2*e] = e;
```

مثال: قطعه‌برنامه‌ای بنویسید که
نمرات ۳۸ دانشجو را از ورودی بگیرد،
سپس ۱- همه نمرات را چاپ کند
۲- معدل کل کلاس را محاسبه و چاپ کند

```
double grades[38];
for(int i=0;i<38;i++)
      cin>>grades[i];
for(int i=0;i<38;i++)
      cout<<grades[i]<<endl;
double sum = 0;
for(int i=0;i<38;i++)
      sum+=grades[i];
double average = sum/38;
cout<<average<<endl;
```

```
const int STUDENTS = 3;
double grades[STUDENTS];
for (int i = 0; i < STUDENTS; i++)
      cin>>grades[i];
double sum = 0;
for (int i = 0; i < STUDENTS; i++){
   cout<<grades[i]<<endl;
   sum+=grades[i];
}
double average = sum / STUDENTS;
cout<<average<<endl;
}
```

دستور #define

- دستور define یک دستور پیش‌پردازشی (preprocessing) است
- یک نام مستعار (نامگذاری مجدد) برای یک مقدار یا عبارت تعیین می‌کند
- دستورات پیش‌پردازشی قبل از کامپایل برنامه اجرا می‌شوند
  - مانند include و define
- مثال:

```
#define PI 3.141592
#define integer int
#define longint long int
```

مثال

```
#define WIDTH 80
#define LENGTH (WIDTH + 10)
#define PI 3.141592
#define multiply(f1,f2) (f1 * f2)
#define max(a,b) ((a)>(b)?(a):(b))
#define swap(a,b) {a+=b;b=a-b;a-=b;}
int main() {
   double var = WIDTH; // double var = 80;
   var = LENGTH*2; // var=(80+10)*2;
   var = 2*PI; // var = 2 * 3.141592;
   var = multiply (2, var); // var = (2*var);
   double a = max(WIDTH, LENGTH);
   //double a =((WIDTH)>(LENGTH)?(WIDTH):(LENGTH));
   swap(a,var);//{a+=var;var-=a;a-=var;}
}
```

طول آرایه

- نکته: طول آرایه (تعداد خانه‌های آرایه) هنگام تعریف باید ثابت و مشخص باشد
  - مثلاً یک عدد یا یک ثابت، نه یک متغیر

- یعنی طول آرایه در **زمان کامپایل** باید مشخص باشد (compile-time value)
  - نه این که در **زمان اجرا** (runtime value) مشخص شود

- آرایه با طول متغیر ← تعیین طول آرایه در زمان اجرا ← آرایه پویا ← بعداً خواهیم دید

طول آرایه: مثال

```
#define LENGTH 10
const int SIZE = 5;
int variable;
cin>>variable;
const int VAR = variable;
```

```
int a[LENGTH];
int b[SIZE];
int c[variable];
int d[VAR];
```

- متغیرها و ثابت‌های روبرو را در نظر بگیرید:
- کدام‌یک از آرایه‌های زیر به درستی تعریف شده‌اند؟
  - آرایه‌های a و b به درستی تعریف شده‌اند
    - زیرا طول آن‌ها در **زمان کامپایل** مشخص است
  - آرایه‌های c و d به درستی تعریف نشده‌اند، زیرا طول آن‌ها در زمان اجرا مشخص می‌شود

- نکته: برخی از کامپایلرها از تعریف c و d هم خطا نمی‌گیرند
  - مثلاً gcc و g++ خطا نمی‌گیرند ولی Visual C++ خطا می‌گیرد
  - این امکان، در نسخه‌های جدیدتر C (مثل C99) مجاز شده است
  - زبان استاندارد C++ (ISO C++) اجازه تعریف آرایه به این شکل را نمی‌دهد

Variable-length automatic arrays

روش‌های مقداردهی اولیه اعضای آرایه

```
int a[4]={1,1,5,7};
```

1	1	5	7

```
float b[7]={1,1,5.5,7.2};
```

1	1	5.5	7.2	0	0	0

```
double c[7]={1};
```

1	0	0	0	0	0	0

عناصری که مقدارشان مشخص نشده، با صفر مقداردهی می‌شوند

```
int m[]={1,2,5};//int m[3]={1,2,5};
```

1	2	5

تعیین ضمنی طول آرایه (implicit array size)

```
int n[2]={1,1,5,7};//syntax error
```

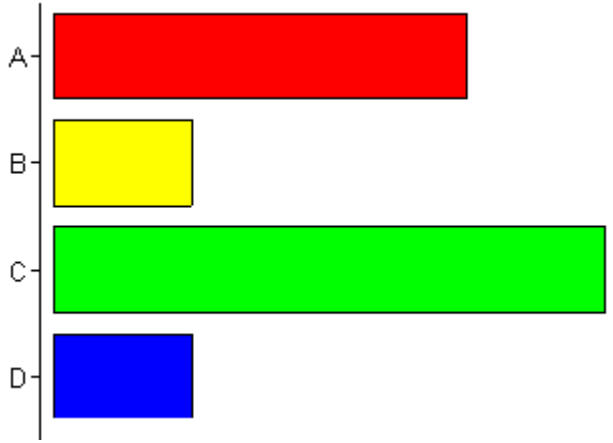
پیمایش آرایه

- در بسیاری از موارد، عملی روی تک‌تک اعضای آرایه انجام می‌شود
- به این کار، **پیمایش** آرایه گفته می‌شود
- مثال:

```
double array[LENGTH] = {1, 2.5, 3.14};
for (int i = 0; i < LENGTH; i++) {
   f(array[i]);
   cout << array[i];
    ...
}
```

- پیمایش: استفاده از خانه شماره صفر تا خانه شماره LENGTH-1 از آرایه

مثال



- یک آرایه، در چهار خانه، تعداد نمرات A و B و C و D را نگهداری می‌کند
  - خانه اول: تعداد نمرات A (عالی)، خانه دوم: تعداد نمرات B (خوب) و ...
- برنامه‌ای بنویسید که مقادیر این آرایه را به صورت یک نمودار میله‌ای رسم کند
  - با کمک تعدادی ستاره (*)

پاسخ

```
const int SIZE = 4;
int n[SIZE] = { 5, 10, 7, 3 };
cout << "Grade distribution:" << endl;
for (int i = 0; i < SIZE; ++i) {
   switch(i){
   case 0 : cout << " A: "; break;
   case 1 : cout << " B: "; break;
   case 2 : cout << " C: "; break;
   case 3 : cout << " D: "; break;
   }
   for (int stars = 0; stars < n[i]; ++stars)
      cout << '*';
   cout << endl; // start a new line of output
}
```

```
Grade distribution:
 A: *****
 B: **********
 C: *******
 D: ***
```

تمرین

- قطعه‌برنامه‌ای بنویسید که

دو آرایه شامل سن و معدل دانشجویان از کاربر دریافت کند

سپس بیشترین معدل و کمترین سن را پیدا کند و چاپ کند

- آرایه‌های موازی:
  - خانه‌های متناظر این آرایه‌ها، ابعاد مختلف اطلاعاتی یک موجودیت را نشان می‌دهند
  - مثلاً خانه پنجم آرایه سن یک فرد است، خانه پنجم آرایه دیگر معدل همان فرد است

نکته

- امکان انتساب یک آرایه به آرایه دیگر وجود ندارد

```
int a[4]={1,1,5,7};
int b[4];
b=a;
```

Compile error: Invalid array assignment

- راه حل؟

```
int a[4]={1,1,5,7};
int b[4];
for (int i = 0; i < 4; ++i)
    b[i]=a[i];
```

- هر عنصر (element) از آرایه،

مثل یک متغیر معمولی قابل استفاده است.

- مثال:

```
int a[4]={1,1,5,7};
int b[]={1,2,3,4,5,6};
int n=2;
a[1]++;
a[0]=--a[1];
a[n] = b[2*n-1];
a[b[0]] = b[a[1]];
```

دسترسی خارج از محدوده به آرایه

- اندیس مورداستفاده، ممکن است از محدوده اندازه آرایه خارج باشد
- زبان C++ محدوده آرایه را کنترل نمی‌کند
- اگر دسترسی به اندیسی خارج از محدوده رخ دهد، (لزوماً) خطا نمی‌دهد
  - مقدار حافظه موردنظر مشخص نخواهد بود
- اگر دسترسی از محدوده حافظه برنامه خارج شود،
  و به حافظه برنامه‌های دیگر برسد،
  سیستم‌عامل خطا می‌دهد و برنامه را قطع می‌کند

```
int a[3]={1,2,3};
a[-2] = 3;
a[3] = 1;
a[5] = 2;
a[2000] = 2;
```

احتمال خطا توسط سیستم عامل

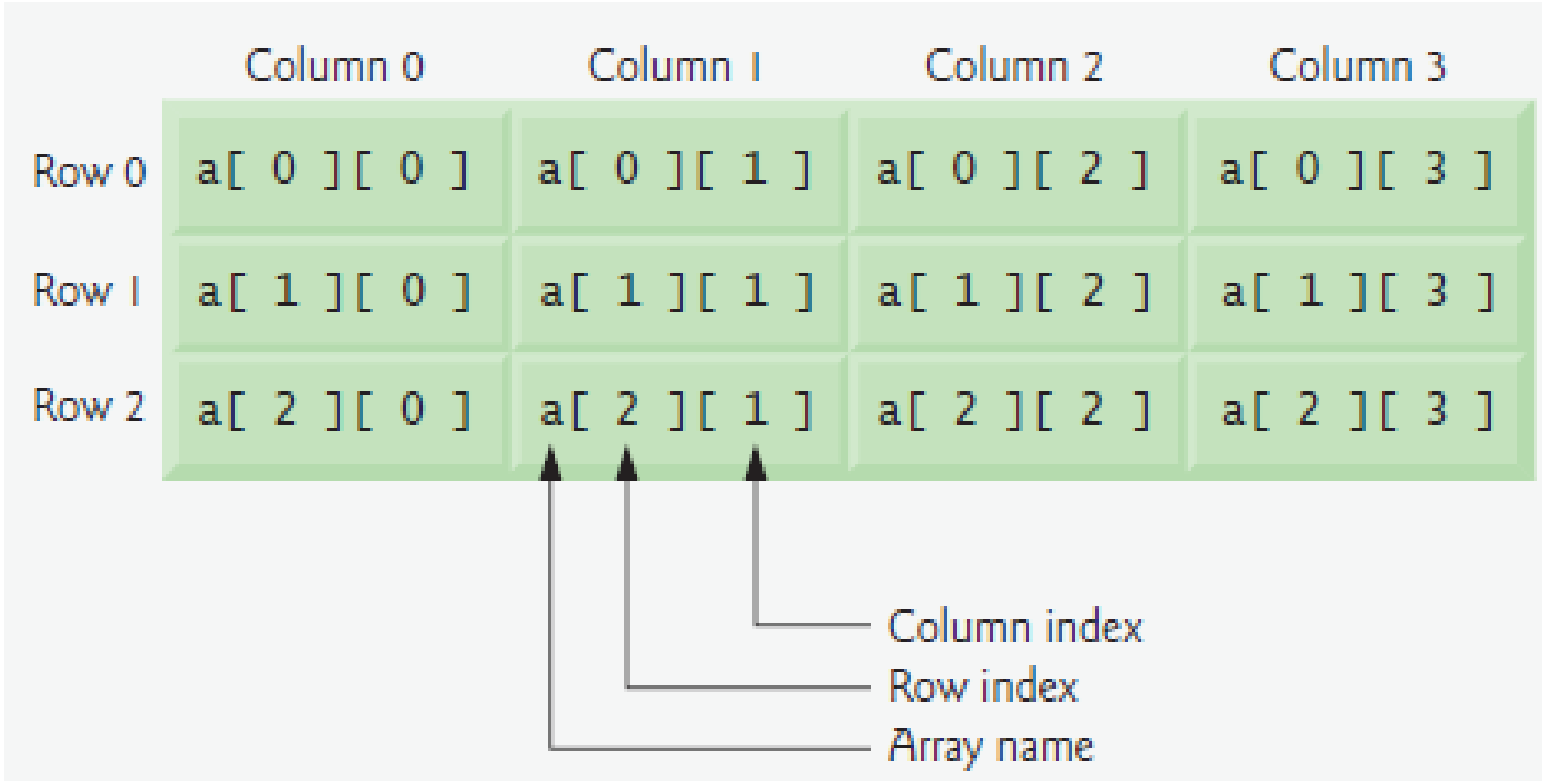
آرایه چندبُعدی

Multi-dimensional Array

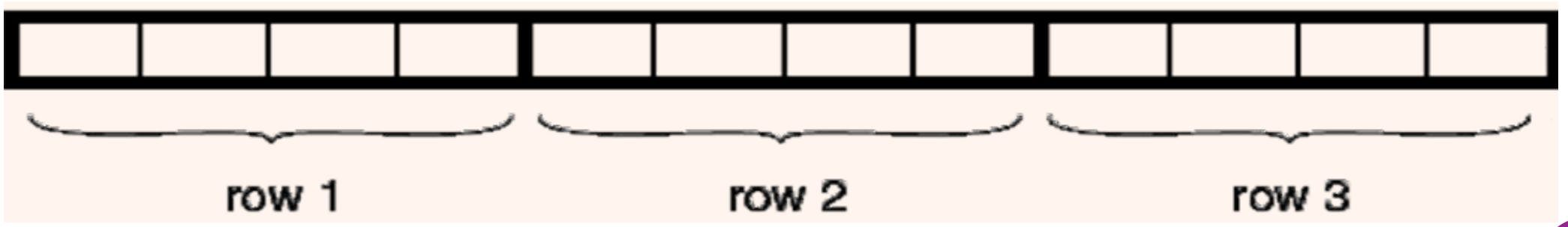
- یک آرایه چندبُعدی از چند آرایه معمولی تشکیل می‌شود
- مثلاً یک آرایه دوبعدی مانند آرایه‌ای از آرایه‌هاست
- مثال: تعریف آرایه دوبعدی از نوع صحیح: `int a[5][4];`
  - یعنی یک آرایه دو بعدی با ۵ سطر و ۴ ستون
- نکته: شهود دوبعدی، در ذهن ماست
  - در کامپیوتر همه خانه‌های آرایه‌ی چندبعدی، پشت سرهم هستند (حافظه دوبعدی نیست)
- آرایه سه‌بعدی از نوع کاراکتر: `char c[5][3][7];`
  - شهود: مکعب
  - مثل یک آرایه ۵ تایی که هر عضو آن، یک آرایه دوبعدی است که ۳ سطر و ۷ ستون دارد

مثال

```
int matrix[3][4];

matrix[2][3] = 2;

cout << matrix[2][1];
```



وضعیت حافظه



مثال

- سؤال: آرایه int a[5][4][2] چقدر حافظه مصرف می‌کند؟
- 5*4*2*sizeof(int)
  - اگر فرض کنیم int (در محیط موردنظر) ۴ بایت اشغال می‌کند، مجموعاً ۱۶۰ بایت
- برنامه‌ای بنویسید که یک آرایه دوبعدی ۱۰در۱۰ شامل جدول ضرب ایجاد کند

```
int a[10][10];
for (int i = 0; i < 10; i++)
   for (int j = 0; j < 10; j++)
      a[i][j] = i * j;
```

مقداردهی اولیه به آرایه چندبعدی

```
int a[3][2] = {{1,2},{3,4},{5,6}};
int a[3][2] = {1,2,3,4,5,6};
int a[][2] = {1,2,3,4,5,6};
```

1	2
3	4
5	6

```
int b[3][2] = {1,2,3};
```

1	2
3	0
0	0

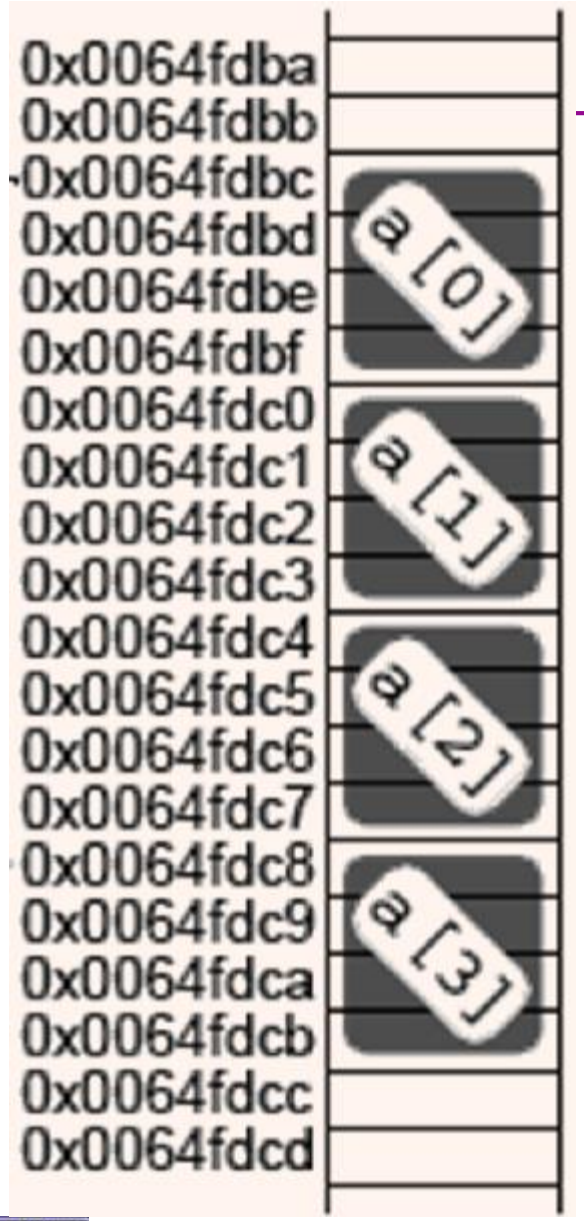
```
int c[3][2][2] = {{{1,1},{2,2}},{{3,3},{4,4}},{{5,5},{6,6}}};
```

```
int d[3][2] = {{1},{3,4},{5}};
```

1	0
3	4
5	0

```
int e[1][2] = {1,2,3,4}; // Syntax Error
int x[][] = {1,2,3,4,5,6}; // Syntax Error
int y[][][3] = {1,2,3,4,5,6}; // Syntax Error
```

نکته



- نام آرایه مانند آدرس شروع آرایه در حافظه عمل می‌کند
  - این آدرس ثابت و غیرقابل تغییر است
- مثلاً a[3] یعنی سه خانه در حافظه جلوتر از شروع آرایه
- مفهومی مشابه اشاره‌گر (pointer) که بعدها خواهیم دید
- مثلاً:

```
int a[4];
a[3] = 2;
cout<<a[3]<<endl;
cout<<a<<endl;
```

```
2
0x0064FDBC
```

ارسال یک عنصر (element) از یک آرایه به تابع

- یادآوری: هر عنصر از یک آرایه مثل یک متغیر معمولی است
- ارسال یک عنصر از آرایه به یک تابع نکته خاصی ندارد
- مثال:

```
double sum(double a, double b){
      return a+b;
}
int main() {
      double a[4]={1.5,2.2,3,4.0};
      cout<< sum (a[1], a[3]);
}
```

آرایه به عنوان پارامتر

- گاهی لازم است یک آرایه به یک تابع ارسال شود
  - پردازشی روی آرایه انجام شود
- مثال:
  - تابعی بنویسید که یک آرایه به عنوان پارامتر بگیرد و میانگین اعضای آرایه را برگرداند
  - تابعی که یک آرایه و یک مقدار X به عنوان پارامتر بگیرد و اندیس خانه‌ای با مقدار X را برگرداند
  - تابعی که یک آرایه به عنوان پارامتر بگیرد و اعضای آرایه را به صورت صعودی مرتب کند

ارسال آرایه به عنوان پارامتر

- برای تعریف پارامتری از نوع آرایه‌ی یک‌بعدی، کافیست نوع اعضای آرایه مشخص شود
  - برای استفاده از آرایه‌ای یک‌بعدی به عنوان پارامتر، لازم نیست طول آرایه را مشخص کنید
- مثال:

```
void f(int array[]){...}
int main() {
   int a[4]={1,2,3,4};
   f(a);
}
```

- نکته مهم: چیزی که به تابع پاس می‌شود، فقط آدرس شروع آرایه است
  - در واقع محتوای آرایه، کپی نمی‌شود
  - اگر تغییری در عناصر پارامتر ایجاد شود، عناصر آرایه اصلی هم تغییر می‌کند

مثال

- تابعی بنویسید که یک آرایه به عنوان پارامتر بگیرد و همه اعضای آرایه را چاپ کند

```
void print(int a[], int size){
   for (int i = 0; i < size; ++i)
      cout<<a[i]<<endl;
}
```

```
int main() {
   int a[]={1,2,3,4};
   print(a, 4);
}
```

- این تابع باید طول آرایه را هم به عنوان پارامتر بگیرد

مثال

- تابعی بنویسید که یک آرایه به عنوان پارامتر بگیرد و کمینه اعضای آرایه را برگرداند

```
int min(int a[], int size){
   int m = a[0];
   for (int i = 1; i < size; ++i)
      if(a[i]<m)
            m = a[i];
   return m;
}
```

```
int main() {
int a[]={1,2,3,4};
cout<<min(a, 4);
```

مثال

- خروجی برنامه زیر چیست؟

```
void print(int a[], int size){
   for (int i = 0; i < size; ++i)
      cout<<a[i]<<endl;
}
void increment(int a[], int size){
   for (int i = 0; i < size; ++i)
      ++a[i];
}
int main() {
   int a[]={1,2,3,4};
   increment(a, 4);
   print(a, 4);
}
```

```
2
3
4
5
```

مسأله جستجو در آرایه (Search)

- تابعی که یک آرایه و یک مقدار X به عنوان پارامتر بگیرد و اندیس خانه‌ای با مقدار X را برگرداند

```
int search(int a[], int size, int value){
   for (int i = 0; i < size; ++i)
      if(a[i]==value)
            return i;
   return -1;
}
int main() {
   int a[]={5,6,9,2};
   cout<<search(a,4, 9);
}
```

2

طول آرایه، وقتی آرایه یک پارامتر است

- در تعریف یک تابع، اگر پارامتری از نوع آرایه یک‌بعدی باشد، لازم نیست طول آرایه ذکر شود
  - چرا؟
  - چون برای دسترسی به یک خانه از آرایه، دانستن نوع (حجم) هر خانه برای کامپایلر کافیست
  - مثال:

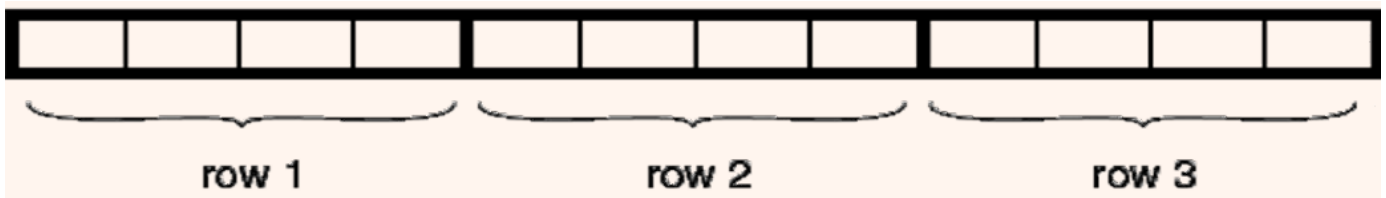
```
void print(int a[], int size){
   for (int i = 0; i < size; ++i)
      cout<<a[i]<<endl;
}
```

  - کامپایلر می‌داند که برای رسیدن به خانه a[i] باید به اندازه i * sizeof(int) از شروع آرایه جلوتر برود
- اما برای آرایه‌های چندبعدی، فقط اندازه بعد اول را می‌توانیم ذکر نکنیم
  - اندازه (طول) همه ابعاد بعدی را باید ذکر کنیم. چرا؟

ارسال آرایه چندبعدی به تابع

```
void print1(int a[3][4], int size){}
void print2(int a[][4], int size){}
void print3(int a[][], int size1, size2){}
```

- کدام تعریف صحیح است؟
- تعریف print1 صحیح است
  - البته ذکر عدد 3 به عنوان تعداد سطرها (طول بُعد اول) لازم نیست
  - (اگر ذکر شود، کامپایلر چک می‌کند که آرگومان تابع به همین شکل تعریف شده باشد)
- تعریف print2 صحیح است
- تعریف print3 صحیح نیست
  - تعداد ستون‌ها باید مشخص شود
  - خطای کامپایل دارد



- کامپایلر چطور آدرس خانه a[2][3] را محاسبه کند؟

تمرین

- فرض کنید همه اعضای یک آرایه، به جز عضو آخر، به ترتیب صعودی مرتب شده‌اند
- مثلاً:
- 1,1,2,2,2,4,7,9,**5**
- تابعی بنویسید که چنین آرایه‌ای را به عنوان پارامتر بگیرد، و کل آرایه را مرتب کند
- با کمک این ایده،

  تابعی بنویسید که یک آرایه نامرتب را به عنوان پارامتر بگیرد و کل آرایه را مرتب کند

مرتب‌سازی آرایه (Sorting)

مسأله مرتب‌سازی (Sort)

- تابعی که یک آرایه به عنوان پارامتر بگیرد و اعضای آرایه را مرتب کند
  - مثلاً به صورت صعودی
- الگوریتم‌های مختلف و متنوعی برای مرتب‌سازی ابداع شده‌اند
  - هر الگوریتم مزایا و معایبی دارد
- در پیاده‌سازی یک الگوریتم هم تفاوت‌هایی دیده می‌شود
  - مثلاً یک نفر ممکن است الگوریتم مرتب‌سازی حبابی را با for و دیگری با while پیاده کند
  - تفاوت‌های جزئی دیگر و برخی بهینه‌سازی‌ها هم در پیاده‌سازی‌ها ممکن است
  - اما ایده اصلی هر الگوریتم مرتب‌سازی، مشخص و متمایز است

تابع swap

- تابعی که دو متغیر را به عنوان پارامتر می‌گیرد و مقدار آن دو را با هم عوض می‌کند
- پیاده‌سازی؟

```
// function definition to swap the values.
void swap(int &x, int &y)
{
   int temp = x; /* save the value at address x */
   x = y;     /* put y into x */
   y = temp; /* put x into y */
}
```

- و یا:

```
#define swap(a,b) {a+=b;b=a-b;a-=b;}
```

- و یا ...
- در بسیاری از الگوریتم‌های مرتب‌سازی، از swap استفاده می‌شود

مرتب‌سازی حبابی (bubble sort)

6 5 3 1 8 7 2 4

- یک روش ساده برای مرتب‌سازی
- عناصر متوالی با هم مقایسه می‌شوند

و در صورت لزوم جابجا (swap) می‌شوند

- تا ترتیب صحیحی بیابند
- مانند بالا رفتن حباب در آب
- شبه کد حالت ساده الگوریتم:

```
for i = 0 to n-1
    for j = 0 to n-1-i
        if a[j+1] < a[j]
            swap a[j] , a[j+1]
```

- بعد از هر مرحله ، i عنصر (از انتها) دقیقاً در جای صحیح خود هستند

مرتب‌سازی حبابی

```
void bubble_sort(int a[], int size) {
   for (int i = 0; i < size; i++)
      for (int j = 0; j <size-i-1; j++)
         if (a[j + 1] < a[j])
            swap(a[j], a[j + 1]);
}
```

```
int main() {
   const int SIZE = 5;
   int a[SIZE];
   for(int i=0;i<SIZE;i++)
      cin >> a[i];
   bubble_sort(a, SIZE);
   for(int i=0;i<SIZE;i++)
      cout<<a[i]<<endl;
}
```

```
for i = 0 : n-1
   swapped = false
   for j = n-1 : i+1
      if a[j] < a[j-1]
         swap a[j] , a[j-1]
         swapped = true
   // ➔ a[0..i] in final position
   break if not swapped
end
```

- شکل‌های دیگر الگوریتم مرتب‌سازی حبابی:

مرتب‌سازی دَرجی (insertion sort)

6 5 3 1 8 7 2 4

- تمام عناصر لیست را یکی یکی برمی‌دارد

و آن را در موقعیت مناسب در بخش مرتب شده قرار می‌دهد (درج می‌کند)

- نتیجه بعد از مرحله i ← i عنصر اول آرایه مرتب هستند
  - نه این که این i عنصر دقیقاً در جای صحیح خود باشند
  - در مرتب‌سازی حبابی:

بعد از مرحله i ، i عنصر دقیقاً در جای صحیح خود بودند

- شبه کد:

```
for i = 1 to n - 1
   x = A[i]
   j = i
   while j > 0 and A[j-1] > x
      A[j] = A[j-1]
      j --
   end while
   A[j] = x
end for
```

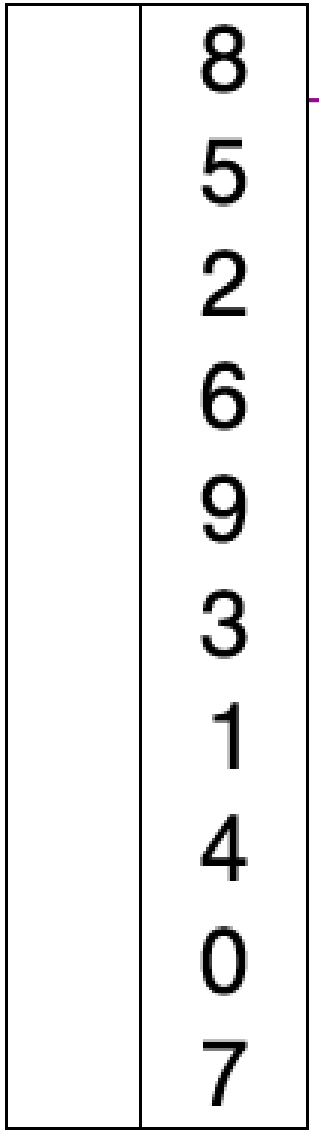
پیاده‌سازی insertion sort

```
void insertion_sort(int a[], int n) {
 for (int i = 1 ;i<n;i++){
    int x = a[i];
    int j = i;
    while (j > 0 && a[j-1] > x){
        a[j] = a[j-1];
        j--;
    }
    a[j] = x;
 }
}
```

```
int main() {
    srand(time(0));
    const int SIZE = 7;
    int a[SIZE];
    for(int i=0;i<SIZE;i++)
        a[i]=rand()%10;
    insertion_sort(a, SIZE);
    for(int i=0;i<SIZE;i++)
        cout<<a[i]<<"  ";
}
```

```
0  4  4  6  6  7  8
```

مرتب‌سازی انتخابی (selection sort)



- ابتدا کوچکترین عنصر مجموعه را یافته (select) با اولین عدد جابجا می‌کنیم
- سپس دومین عنصر کوچکتر را یافته با دومین عدد جابجا می‌کنیم
- و این روند را برای همه آرایه تکرار می‌کنیم
- بعد از i مرحله، i عنصر اول آرایه در جای صحیح خود قرار دارند

الگوریتم Selection Sort

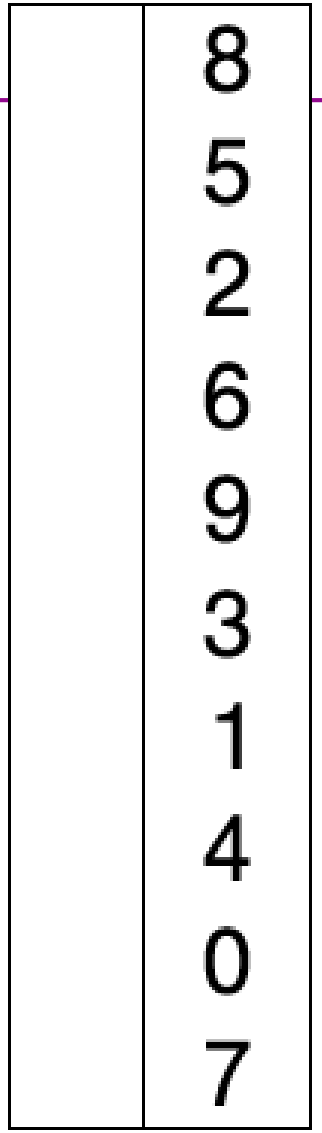
```
void sort(int a[], int n) {
   int min_index;
   for (int i = 0; i < n - 1; i++) {
      min_index = i;
      for (int j = i + 1; j < n; j++)
         if (a[j] < a[min_index])
            min_index = j;

      if (min_index != i) {
            swap(a[i], a[min_index]);
      }
   }
}
```

چند نکته

- الگوریتم‌هایی که دیدیم، آرایه را به صورت صعودی مرتب می‌کردند
  - ایده و پیاده‌سازی برای مرتب‌سازی نزولی هم مشابه است
- الگوریتم‌هایی که دیدیم، آرایه‌ای از اعداد صحیح را مرتب می‌کردند
  - مرتب‌سازی آرایه‌ای از انواع دیگری هم مشابه است
  - کافی است مقایسه بین دو عنصر (element) ممکن باشد
    - مثلاً فهرست اسامی (رشته‌ها) و افراد (بر اساس معدل) هم قابل مرتب‌سازی است

تمرین

- الگوریتم‌های مرتب‌سازی مختلف را پیاده‌سازی کنید
- (هریک را در یک تابع مجزا)



Selection Sort

6 5 3 1 8 7 2 4

Bubble Sort

6 5 3 1 8 7 2 4

Insertion Sort

مرتب‌سازی ادغامی (merge sort)

- اگر طول آرایه صفر یا یک باشد: آرایه مرتب است (شرط پایان)
- آرایه را به دو زیرآرایه تقریباً مساوی تقسیم کنید
- هر زیرآرایه را به صورت بازگشتی مرتب کنید
- دو زیرآرایه مرتب را با هم ادغام (merge) کنید

- ایده‌های اصلی این روش:
  - یک آرایه کوچک، نسبت به یک آرایه بزرگ از گام‌های کمتری برای مرتب‌سازی استفاده می‌کند
  - برای ادغام دو آرایه مرتب‌شده نسبت به دو آرایه نامرتب گام‌های کمتری نیاز است (باید هر آرایه را فقط یکبار پیمایش کنید)

ایده کلی merge

```
void merge(int m, int n, int A[], int B[], int C[]) {
      int i=0, j=0, k=0;
      while (i < m && j < n) {
            if (A[i] <= B[j]) {
                  C[k] = A[i];
                  i++;
            } else {
                  C[k] = B[j];
                  j++;
            }
            k++;
      }
      if (i < m)
            for (int p = i; p < m; p++) {
                  C[k] = A[p];
                  k++;
            }
      else
            for (int p = j; p < n; p++) {
                  C[k] = B[p];
                  k++;
            }
}
```

- توجه: این تابع merge است
- تابع merge_sort نیست
  - در merge_sort از

چنین تابعی می‌توان استفاده کرد

جستجوی دودویی (binary search)

- مسأله جستجو (Search) :

تابعی که یک آرایه و یک مقدار X به عنوان پارامتر بگیرد و اندیس خانه‌ای با مقدار X را برگرداند

- فرض کنید آرایه موردنظر، مرتب (sorted) باشد
- راه‌کارهای سریع‌تری برای جستجو در آرایه مرتب می‌توان یافت

که از جستجوی خطی (پیمایش معمولی) بهترند

جستجوی دودویی

- برای جستجو، آرایه را از وسط به دو بخش چپ و راست تقسیم می‌کنیم
- مقدار مورد جستجو با آخرین عنصر بخش چپ مقایسه می‌کنیم
  - یعنی با عنصری که تقریباً در میانه آرایه جای گرفته
- اگر این عنصر کوچک‌تر از مقدار جستجو باشد، مورد جستجو در بخش چپی وجود ندارد
  پس باید در بخش راستی به دنبال آن بگردیم
- وگرنه، اگر این عنصر بزرگ‌تر از مقدار جستجو بود، مورد جستجو در بخش چپی وجود دارد
  پس باید در همان بخش چپ به دنبال آن بگردیم
- دوباره برای بخش چپ یا راست، آرایه را به دو بخش مساوی تقسیم می‌کنیم و گام‌های بالا تکرار می‌شود
- شرط پایان: در نهایت محدوده‌ی جستجو به یک عنصر محدود می‌شود
  - یا آن عنصر با مورد جستجو برابر است و عنصر مذکور یافت شده
    و یا این که آن عنصر مورد جستجو برابر نیست و لذا مورد جستجو در آرایه وجود ندارد
- این روش پیچیده تر از روش جستجوی خطی است اما بسیار سریع تر به جواب می رسیم

پیاده‌سازی جستجوی دودویی

```
int Bsearch(int,int[],int);
int main(){  int a[] = { 2, 43, 88, 188, 464, 567, 655};
cout << "index(464,a,7) = " << Bsearch(464,a,7) << endl;
cout << "index(500,a,7) = " << Bsearch(500,a,7) << endl;
}

int Bsearch(int x, int a[], int n){      // binary search:
  int l=0, h=n-1, i;
  while (l <= h){
    i = (l + h)/2;              // the average of l and h
    if (a[i] == x)
      return i;
    if (a[i] < x)
      l = i+1;   // continue search in a[i+1..h]
    else
      h = i-1;                 // continue search in a[0..i-1]
  }
  return -1;                        // x was not found in a[0..n-1]
}
```

جستجوی دودویی بازگشتی

```
int rBinarySearch(int sortedArray[], int first, int last, int key) {
    if (first <= last) {
        int mid = (first + last) / 2;
        if (key == sortedArray[mid])
                return mid;
        else if (key < sortedArray[mid])
                return rBinarySearch(sortedArray, first, mid - 1, key);
        else
                return rBinarySearch(sortedArray, mid + 1, last, key);
    }
    return -1;
}
```

```
int main() {
    int a[] = { 1, 2, 3, 5, 6, 8, 9, 12 };
    int n;
    cin>>n;
    cout << rBinarySearch(a, 0, 7, n);
}
```

جمع‌بندی

جمع‌بندی

- مفهوم آرایه
- نحوه تعریف آرایه
- نحوه استفاده از آرایه‌ها
- روش‌های مرتب‌سازی آرایه
- روش‌های جستجو در آرایه‌ها

مطالعه

- فصل ۶ از کتاب: C How to Program (Deitel&Deitel) 7th edition
  - و یا فصل متناظر در کتاب‌های مشابه

جستجوی بیشتر

- چه روش‌های دیگری برای مرتب‌سازی آرایه‌ها وجود دارد؟
- الگوریتم‌های مرتب‌سازی از نظر کارایی (سرعت) چگونه مقایسه می‌شوند؟
  - کلاً چگونه می‌توانیم یک الگوریتم را تحلیل کنیم؟
  - مثلاً $O(n^2)$ یا $O(n \lg n)$ یعنی چه؟
- هزینه الگوریتم جستجوی باینری چقدر بهتر از جستجوی معمولی است؟

- چه دستورات پیش‌پردازشی دیگری به جز include و define وجود دارند؟

تمرین

- تابعی بنویسید که یک ماتریس مربعی (آرایه دوبعدی) به عنوان پارامتر بگیرد و دترمینان آن را برگرداند

- تابعی بنویسید که یک ماتریس به عنوان پارامتر بگیرد و قطر اصلی آن را مشخص کند (به صورت یک آرایه یک‌بعدی که در یکی از پارامترها پر می‌شود)

پایان