Week 4

Muhao Chen

Email: muhaochen@ucla.edu

Outline

- Preprocessor
- Arrays
  - Basic arrays
  - Char arrays
  - String arrays
- Use arrays inside a function

Preprocessor

- Preprocessor: executed before the actual compilation of code, therefore the preprocessor digests all these directives before any code is generated by the statements
- Start with hash sign #
- **No semicolon at the end**
- #include is also one kind of preprocessors
- Marco definition is another kind of preprocessors:
  - #define pattern target_value

```
#define PI 3.14
#define MAXSIZE 1000
```

Preprocessor

```
#include <iostream>
using namespace std;
#define PI 3.14159
#define NEWLINE '\n'

int main ()
{
        double r=5.0;
        circle = 2 * PI * r;
        cout << circle; cout << NEWLINE;
        return 0;
}
```

3.14159
It is one way to define a constant value

*http://www.cplusplus.com/doc/tutorial/constants/

Preprocessor

- You can change the value of a defined pattern by using *#undef*
- In following example, we define three arrays, a1[10], a2[100], a3[1000]

```
#define MAX 10
int a1[MAX];
#undef MAX
#define MAX 100
int a2[MAX];
#undef MAX
#define MAX 1000
int a3[MAX];
```

Preprocessor

- Conditional inclusions: allow to include or discard part of the code of a program if a certain condition is met
  - #ifdef, #ifndef, #if, #else and #elif
- #ifdef allows a section of a program to be compiled only if specific macro has been defined (no matter what value it has).
- #ifndef prevents redefinition of a macro

```
#ifdef DISPLAY
// run some code here
#endif
```

Preprocessor

```
#ifndef DISPLAY1
#define DISPLAY1 2147483647
#endif
#define DISPLAY2 -2147483648

int main()
{
        #ifdef DISPLAY1
        cout << "We defined display1!" << endl;
        #endif

        #ifdef DISPLAY2
        cout << "We defined display2!" << endl;
        #endif
}
```

We defined display1!
We defined display2!

Preprocessor

- A widely used technique to comment out large pieces of code in large projects
- #if 0

  func();

  …
- #endif

- Companies use this instead of using /* … */

Arrays

Arrays

- A consecutive set of variables of the same type

Arrays

- Two classes of basic arrays

The ordinary ones	The only special one
int a[5]; float b[5]; double c[5]; …	char c[5]

- String arrays

Declare an array

- How to declare an array
  - Type name [# of elements];
  - int a[5];
- # of elements:
  - A positive number: 5, 100, …
  - A predefined integer macro:
    - #define MAX_LENGTH 100
    - int a[MAX_LENGTH];
  - A constant int:
    - const int num=100;
    - int a[num];

Declare an array

- int variables as # of elements->

```
int length = 5;
int a[length];
```

X This is not allowed in many compilers.

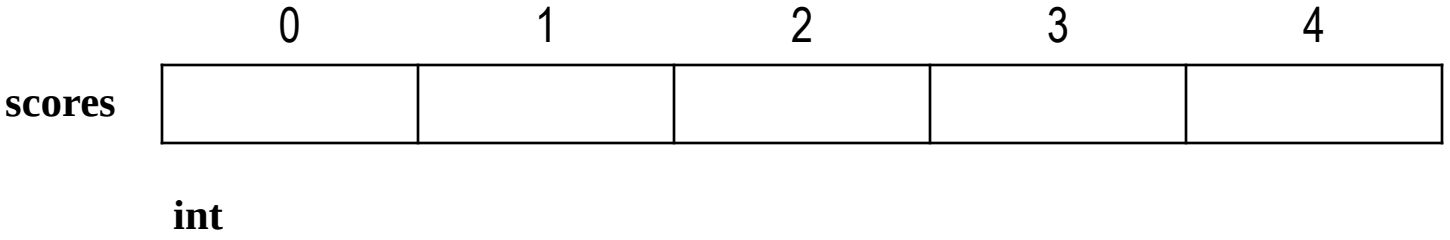
- int a[]; X (int a[] = {1,2,3} is allowed)
- int a[0]; X
- int a[2.1]; X

Declare an array

- Why do we need to specify a constant #elements for an array?
- Actually it's just an ill-designed feature of the C language
- (Though, it appears that some newer versions of compilers are removing this constraint.)

Declare an array

- int score[5];
  - These elements are numbered (indexed) from 0 to 4



  - The first index in the array is always 0

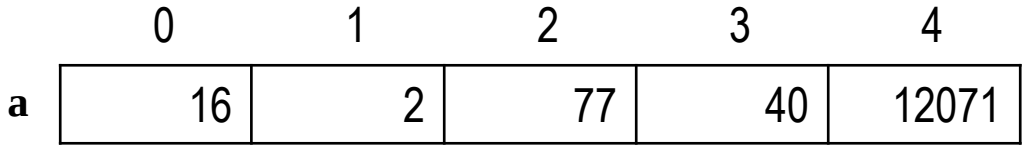
Initialize an array

- Declare an array without initializing it
  - int a[4]; //hasn't been initialized, the values are undefined

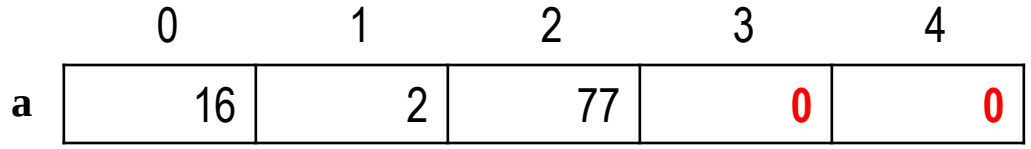
```
int a[4];
cout << a[0] << " " << a[1] << " " << a[2] << " " << a[3] << endl;
```

Acceptable initialization

- The standard way:
  - int a[5] = {16, 2, 77, 40, 12071} ;
  - int a[] = {16, 2, 77, 40, 12071};



- #Values less than #Elements
  - int a[5] = {16, 2, 77} ;
  - The rest will become **all 0**.



Unacceptable initialization

- Invalid #Elements:
- #Values are more than #Elements
  - int scores[5] = {16, 2, 77, 40, 12071, 0} X
- Inconsistent and unconvertible types
  - int a[3] = {18, 72, “abc”} X
- How about *int a[3] = {18, 72, ‘a’}*?
  - -> {18, 72, 97} ✓

Acceptable / unacceptable

Acceptable	Unacceptable
int a[5]; const int n=5; int a[n] float a[] = {1.1, 2.0, 3.5, 4, 5} int a[5] = {1, 2, 3} int a[5] = {1, 2, ‘a’}	int a[]; int n=5; int a[n]; int a[3] = {1, 2, 3, 4}; int a[3] = {1, 2, “a”};

Initialize an array

- Questions
- How to initialize int a[100] to all 0?

```
int a[100] = {0};
```

- How to initialize int a[100] to all 1?

```
for (int i = 0; i <= 99; ++i)
      a[i] = 1;
```

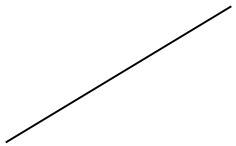
Access elements of an array

- Random access (or, direct access)
  - name[index] //index must be non-negative int value
- Arithmatics can be combined with random access
- int a[5] = {1,2,3,4,5}
  - a[2] = 5;
  - ++a[3];
  - in x=1; int b = a[x+2];
  - a[a[2]] = a[2] + 5; (i.e. a[3] = 3 + 5;)
  - a[5] = 5; X (out of bound)

print an array

```
int a[5] = {1,2,3,4,5}
cout << a[1] << " " << a[3] << endl;
cout << a << endl;
```

Output:
2 4
0089FF08

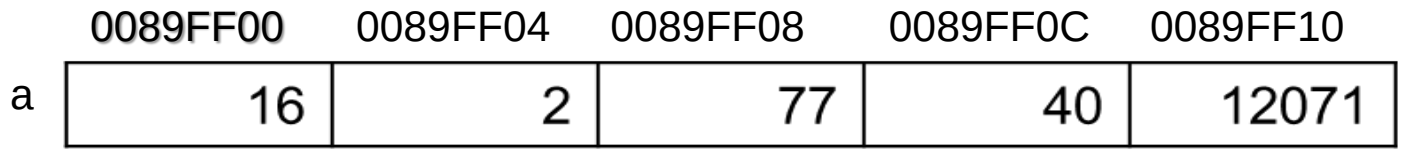


Starting address of a[] in the memory

- To access an element, we'll get its value.
- If we just access the name of a basic array we will only see the address of it.
- Cout is different for char array (see following pages).

Copy an array (Deep copy)

- The name of an array: The address of the first element in the array.



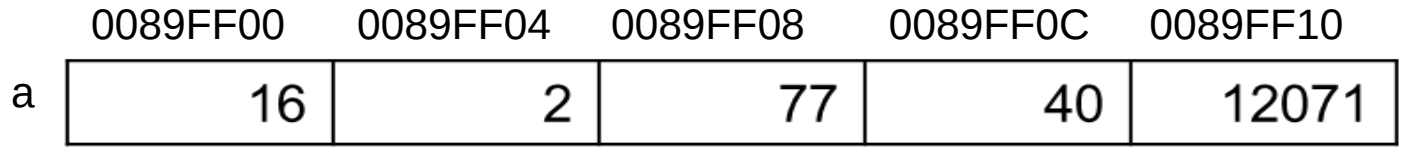
- How to copy the content of a[] to b[]?
  - Copy it element by element! (Deep Copy)

```
int a[] = {16, 2, 77, 40, 12071};
int b[5];
for (int i=0; i<5; ++i) b[i] = a[i];
```

Copy an array

Not allowed in some compilers.

- What if we do b=a?



- This just make *b* and *a* the same single array. *b* is not a hard copy of *a.*
  - (This is shallow copy; i.e. b = 0x0089FF00)

Differences between deep and shallow copy

```
int a[] = {16, 2, 77, 40, 12071};
int b[5];
b = a;
a[0] = 1;
cout << b[0] << endl;
```

Output: 1

a is {1, 2, 77, 40, 12071}
b is {1, 2, 77, 40, 12071}

```
int a[] = {16, 2, 77, 40, 12071};
int b[5];
for (int i=0; i<5; ++i) b[i] = a[i];
a[0] = 1;
cout << b[0] << endl;
```

Output: 16

a is {1, 2, 77, 40, 12071}
b is {16, 2, 77, 40, 12071}

Char Array (C-String)

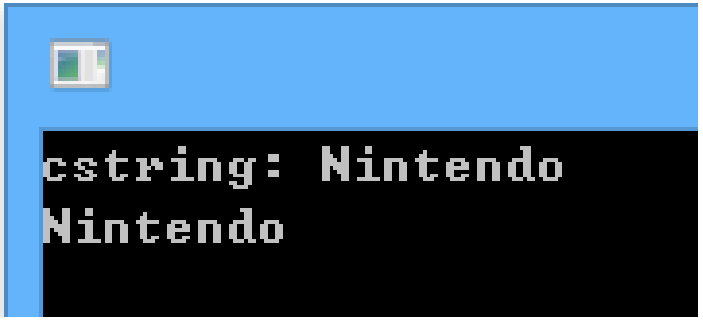
Character array (char c[])

- string in C language (when C++ has not been invented, we call it C-string)
- The special type of array.
  - E.g., we can initialize it with a string value ("…").
  - We can cin / cout the entire char[] with its name.
  - It uses a '\0' (0) to denotes its end
- C++ string class is a class extended from char[]

Initialize a char c[]

- initialize a char c[]
  - char c[10] = {‘a’, ‘b’, ‘c’}
  - char c[10] = “abc”
  - cin >> c (not supported for other types of arrays)
- cout << c;

```
char c[100];
cout << "cstring: ";
cin >> c;
cout << c << endl;
```



Initialize a char c[]

- Question: {‘a’, ‘b’, ‘c’} == “abc”?
  - X “abc” is actually {‘a’, ‘b’, ‘c’, ‘\0’}, where ‘\0’ (or 0, or NULL) is the end of a string
  - (sizeof(“abc”) / sizeof(char)) is 4 (not 3)
- Question: char c[3];
  - char c[3]={‘a’, ‘b’, ‘c’}; ? ✓
  - char c[3]=“abc”; ? X
- With a cstring c[100], we can initialize it with a string value with the maximum length of 99.

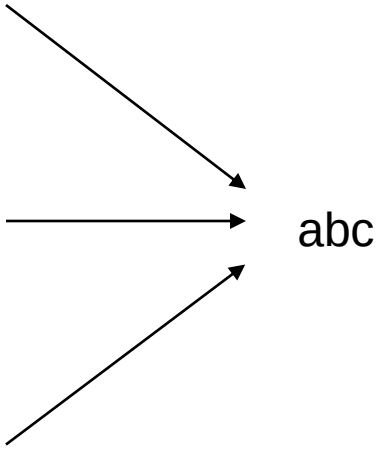
cout a char c[]

- Output characters until reaching a '\0'

```
char c[100] = "abc";
cout << c;
```

```
char c[100] = {'a', 'b', 'c', 0};
cout << c;
```

```
char c[100] = {'a', 'b', 'c'};
cout << c;
```



cout a char c[]

- Question:

  char c[3] = {‘a’, ‘b’, ‘c’};

  cout << c << endl;

- What will we get now?
- Undefined behavior.

```
char c[3] = {'a', 'b', 'c'};
*(c + 3) = '1'; //ignore this for now
cout << c << endl;
```

A Glance at Two Dimensional Arrays

- int xy[3][4] = { {1,2,3,4} , {5,6,7,8}, {4,3,2,1} };

1	**2**	**3**	**4**
5	6	7	8
4	3	2	1

- xy[1]

5	**6**	**7**	**8**

- xy[1][3] is 8
- xy[2][2] is 2

String array

- Array of strings.
  - string fruits[4] = {“lemon”, “coconut”, “apple”, “orange”}
- Each element is a string
  - cout << fruits[1] <<endl;

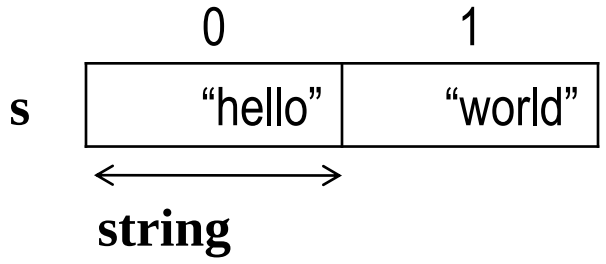
coconut

- Similar to a two dimensional character array.
  - cout << fruits[1][2] << endl;

c

String Arrays

```
string s[2];
string hello = “hello”;
s[0] = hello;
s[1] = “world”;
cout << hello[1];
cout << s[0][1];
```




```
this prints:
ee
```

Use Arrays inside a Function

- How to use arrays in functions

```
void print_array(int a[], int len)
{
	for (int i=0;i<len;i++)
		cout << "[" << i << "] = " << a[i] << endl;
}
int main()
{
	int a[7] = [2, 0, 1, 2, 2, 2, 7];
	print_array(a, 7);
}
```

Cannot add #elements to an array parameter

A function will not know the length of a basic array unless you provide it.

To pass an array to a function, just pass the name of the array.

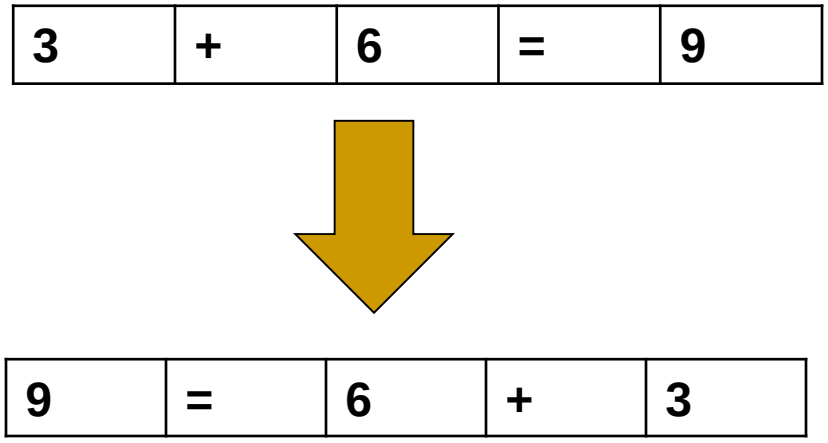
Array as a Parameter

- An array as a parameter is always an actual parameter. (mutable)

```
void invert_array (char a[], int len) {
      for (int i=0; i < len / 2; ++i) {
            char tmp = a[i];
            a[i] = a[len – 1 – i];
            a[len – 1 – i] = tmp;
      }
}

int main(){
      char a[6] = {‘3’, ‘+’, ‘6’, ‘=‘, ‘9’};
      invert_array(a, 5);
      cout << a;
}
```

9=6+3



One (might be) useful function for project 3

- string substr(unsigned int pos, unsigned int len);
  - str.substr(pos, len): return the substring starting at pos with a length of len
  - string str = “D3/F#3/A3/D4//D3F#3A3D4/”;
  - string chord = str.substr(3, 4); //will get “F#3/”

Preprocessor

- Is macro definition the same as a constant variable?

```
#include <iostream>
using namespace std;
#define MAX(a,b) ((a)>(b)?(a):(b))

int main ()
{
        cout << MAX(2147483648, INT_MAX);
        return 0;
}
```

That is the MAX macro in the C standard library.