Arrays II

DM2111
C++ Programming

Introduction

Introduction	Break
Problem solving	Array and Strings
Basic elements of C++	Array and Strings
Basic elements of C++	Pointers
Statements	Pointers
Repetition	I/O operations
Functions	Structs
Functions	Others

Agenda

- C-Style Strings
- String Class (C++)
- STL Vector
- Multi-dimensional Arrays (FYI)

C-style string literals are enclosed in double quotes

```
char name[5] = "John";

'k'; // character literal
"k"; // C-style string literal
```

A character array is an array of type char

```
char school[50];
char name[4] = {'B', 'o', 'n', 'd'};
char skinny[] = "Sean";
```

school	u	P	j	Y	p	a
name	B	o	n	d	q	W
skinny	S	e	a	n	\0	i

```
cout << school << endl; // ???Z?
cout << name << endl;   // Bond???Z?
cout << skinny << endl; // Sean
```

“C-Style Strings must be null-terminated”

```
char school[50] = "";
char name[5] = {'B', 'o', 'n', 'd', '\0'};
char skinny[] = "Sean";
```

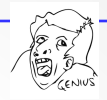
school	\0	P	j	Y	p	a
name	J	o	h	n	\0	W
skinny	S	e	a	n	\0	i

```
cout << school << endl; // nothing
cout << name << endl;   // Bond
cout << skinny << endl; // Sean
```

C-style string initialization

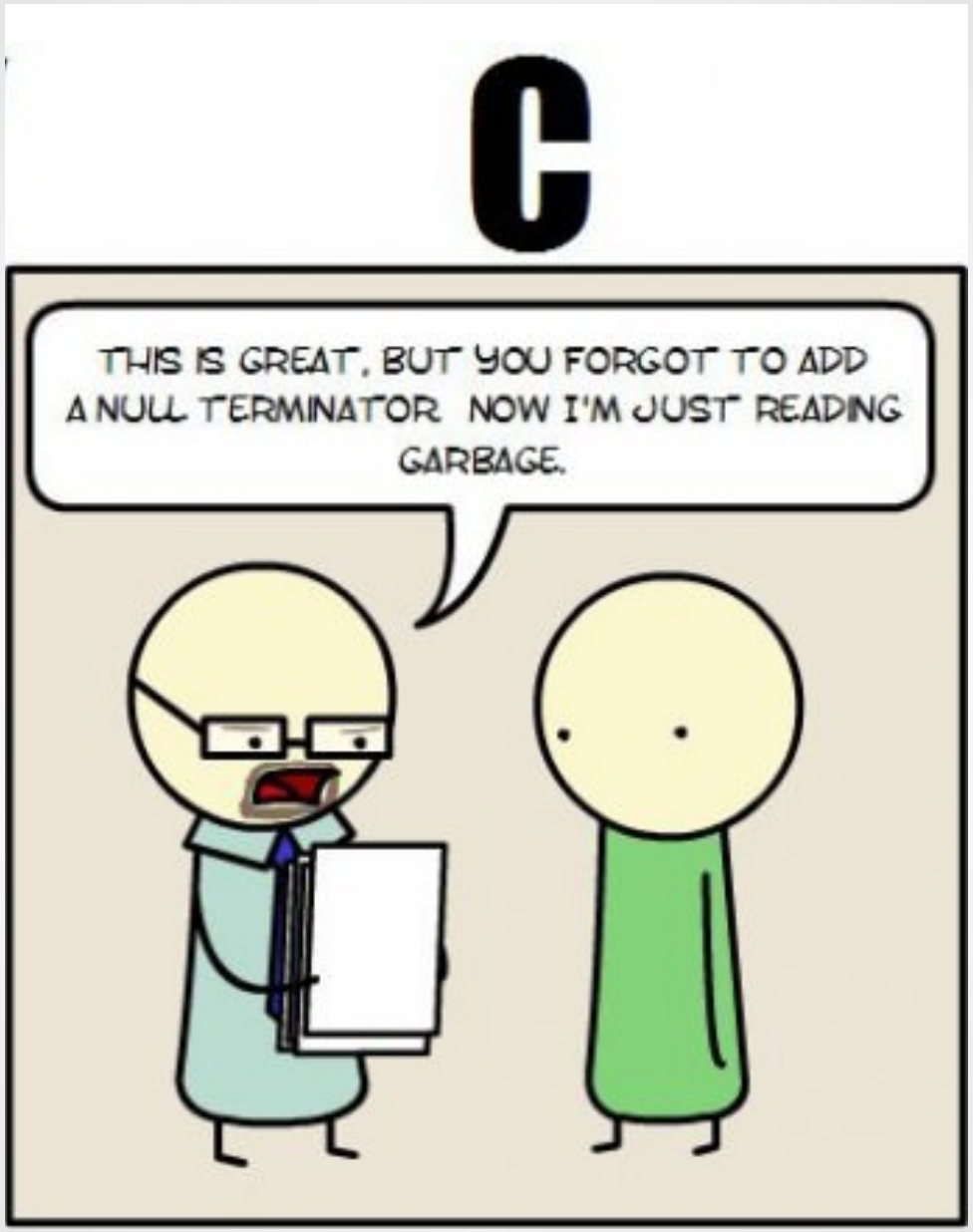
Different ways of initialization

```
char name1[] = {'A', 'n', 'd', 'y'};
char name2[] = {'A', 'n', 'd', 'y', '\0'};
char name3[] = "Andy";

cout << "name1 = " << name1 << endl;
cout << "name2 = " << name2 << endl;
cout << "name2 = " << name3 << endl;
```



output

```
name1 = Andy╣ ☺♣♥
name2 = Andy
name3 = Andy
```



http://d3dsacqprgcsqh.cloudfront.net/photo/aWZXKyx_700b_v1.jpg

C-Style string functions

String length

```
char name[10] = "John";
cout << strlen (name);    // 4
```

String copy

```
char name[10] = "John";
char name2[5];

strcpy (name2, name);
cout << name2;            // John
```

String compare

```
char name[10] = "Joe";
char name2[5] = "Moe";
char name3[5];

strcpy (name3, name);
cout << strcmp (name, name3);    // 0
cout << strcmp (name, name2);    // -ve
```

C-style strings with iostream

Reading a word at a time

```
char name[50];

cout << "Please enter your name: ";
cin >> name;

cout << "Hi " << name << endl;
```

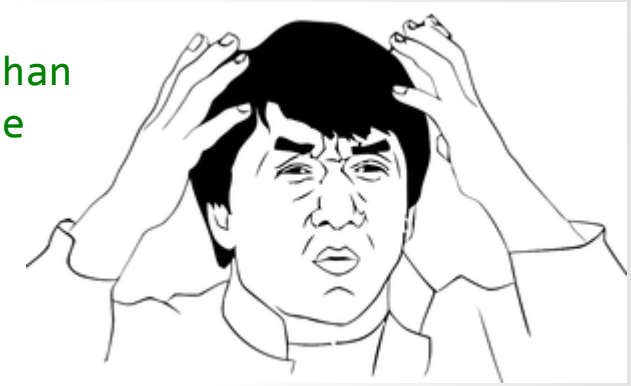
To read strings with blanks

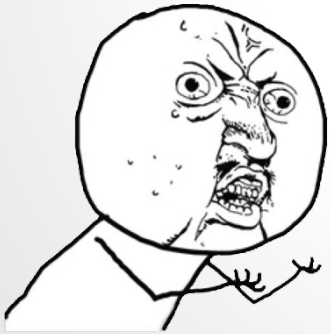
```
char name[20];

cin.get (name, 20);                 // John Tango
cout << "Hi " << name << endl;      // Hi John Tango
```

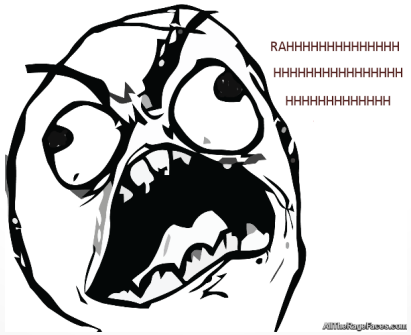
C-style strings with iostream

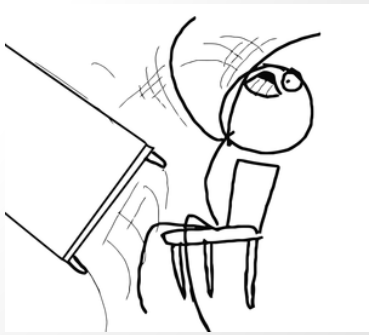
Lots of things can go wrong

```
char name[8];

cin.get (name, 8);                // Jackie Chan
cout << "Hi " << name << endl;    // Hi Jackie

cin.getline (name, 8);
cout << "Hi " << name << endl;    // Hi Chan
```





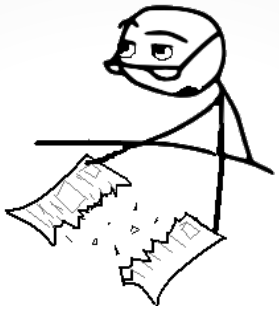




Read the documentation!

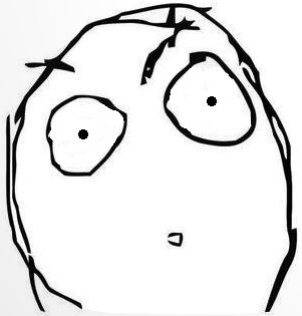
Passing C-style string as functions

Your C-style strings are interpreted as a `char` *pointer*



Same with all array types in C / C++

Pointers?



```
int sum (int num[10]);
int sum (int num[]);
int sum (int* num);
// They are all the same
```

Array as read-only parameters

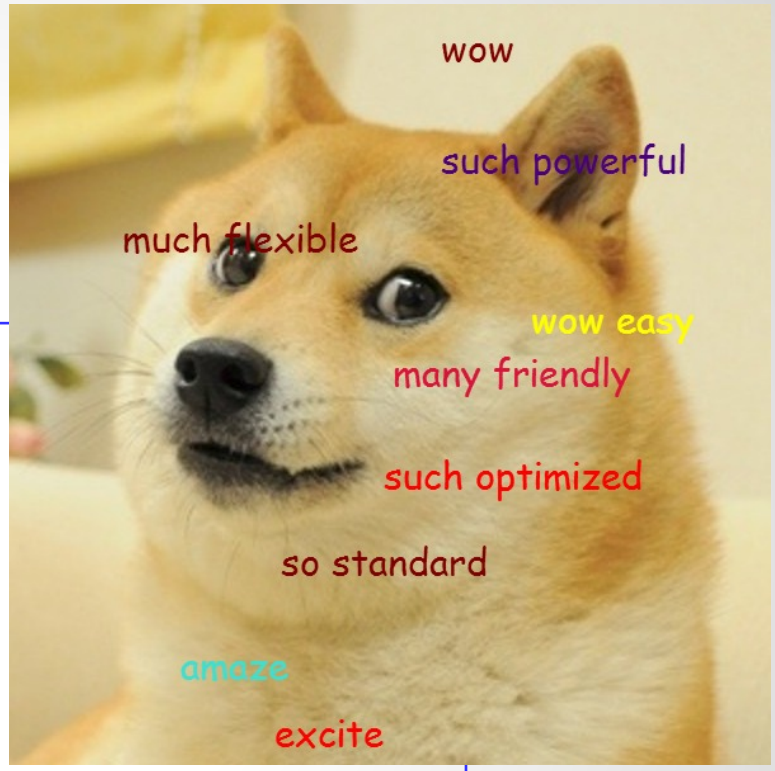
To prevent function from changing parameter values, declare them as const

```
// somewhere in the code
print("You da man!");

void print (char str[])
{
    str[0] = toupper(str[0]); // you crash here
    cout << str; // works
}
```

```
// somewhere in the code
print("You da man!");

void print (const char str[])
{
    str[0] = toupper(str[0]); // not allowed, compilation error
    cout << str; // works
}
```

String class makes life easier

```
#include <string>
using std::string;
```




```
#include <string>
#include <iostream>

using std::string;

void main (void)
{
    string str1, str2, str3;

    str1 = "Hello";              // Hello
    str2 = str1 + " There!";     // Hello There!
    str3 = str2;                 // Hello There!
    str3[6] = 't';               // Hello there!
}
```

String Class

Some string class functions

```
string str, str2;

str.length ();       // return length of string
str.size ();         // return length of string
str.compare (str2); // compare current string with str2
str.substr (3, 5);  // generate substring
...
```

Converting between strings and C-style strings

```
string str = "data";
char cstr[50];

strcpy (cstr, str.c_str()); // copy contents of str to cstr

string str2 (cstr);
```

Why do we still use C-style strings?

Compatibility issues with older libraries

Some say C-style strings are faster

When you just need an array of `char`

Arrays giving you problems?

Don’t know how much space to allocate?

Don’t know which is the last element?

Arrays giving you a lot of bugs?

Sleepless nights because of arrays?

C++ have the perfect solution!

STL Vector is a flexible “array”

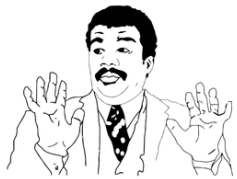
```
#include <vector>
using std::vector;
```

Not the 2*i* + 3*j* vector

It is a **collection** of objects

A **container** type

It is a **class template**



Don’t worry, just use it.

STL Vector dynamically allocates memory

Compared to arrays, you don't need to know the exact size you need beforehand.

```
vector<int>  ivec;
vector<char> cvec;
vector<string> svec;
vector<vector<int>> iivec;
// some older compilers require you to leave a space between >>

ivec.push_back(1); // add 1 to ivec
ivec.push_back(3); // add 3 to ivec
ivec.push_back(5); // add 5 to ivec

vector<int> ivec2(ivec); // ivec2 now contains {1, 3, 5}
```

STL Vector access

```
vector<int>  ivec; // empty vector of ints

ivec.push_back(1); // add 1 to ivec
ivec.push_back(3); // add 3 to ivec
ivec.push_back(5); // add 5 to ivec

cout << ivec; // error, you still need to cout element by element

for (size_t i = 0; i < ivec.size(); ++i)
{
    cout << ivec[i]; // access is the same as arrays
    ivec[i] *= 2;    // you can assign values the same way too
}
```

What is the last accessible element?

You can't access elements which you haven't defined.

STL Vector access

STL Vector access with iterators

```
vector<int>  ivec; // empty vector of ints

ivec.push_back(1); // add 1 to ivec
ivec.push_back(3); // add 3 to ivec
ivec.push_back(5); // add 5 to ivec

cout << ivec; // error, you need to cout element by element

for (vector<int>::iterator iter = ivec.begin(); iter != ivec.end(); ++iter)
{
    cout << *iter; // need to dereference the iterator
}
```

Safer way of accessing elements

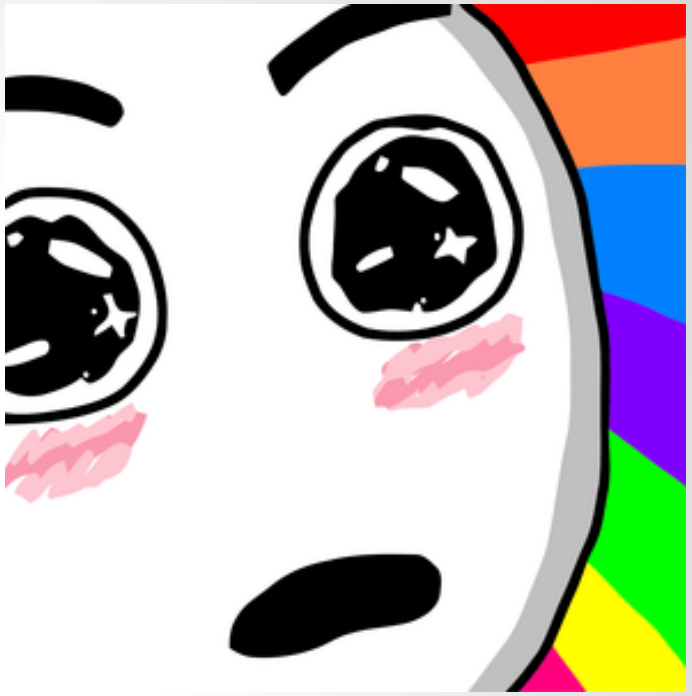
Could get long winded

New C++11 standard makes it easier, but we are not using it for now.

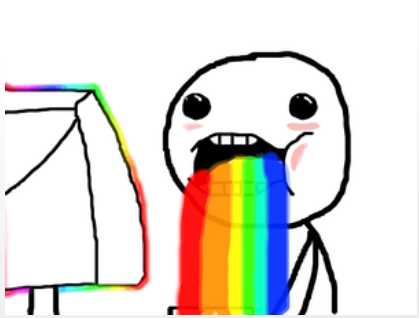
STL Vector other awesome stuff

With STL vector, you can

- delete elements in the middle!
- insert elements anywhere!
- append one vector to another!
- assign one vector to another!
- check if it is empty!
- clear contents!
- compare vectors!



Other STL containers use the same convention, so you don't need to relearn stuff!

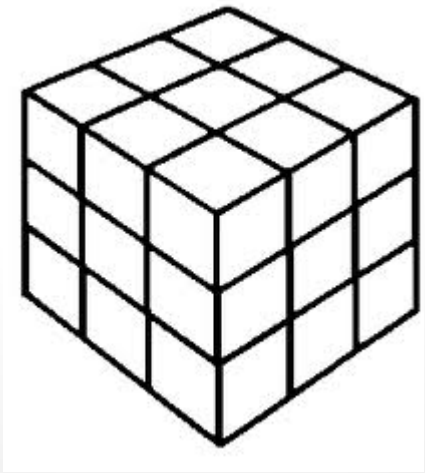


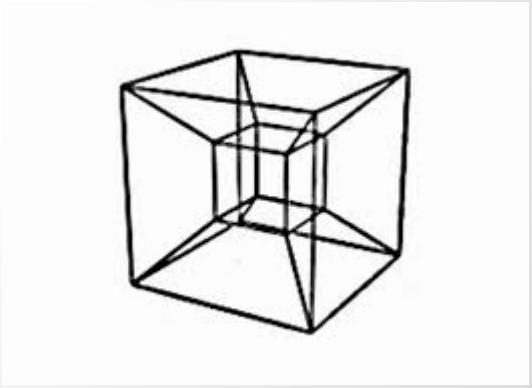
Multi-dimensional Arrays

- 2 dimensional arrays – a fixed number of components arranged in rows and columns (matrix)
- 3 dimensional arrays – a fixed number of components arranged in rows, columns and depth (box)
- Higher dimensional arrays – cannot be visualised

89
78
90
86
95

A	C	Q	T
D	H	W	J
U	Q	G	S





Multi-dimensional Arrays

- Declaration

```
int  score[5];                  // 1 dimensional
char table[3][4];               // 2 dimensional
int  box[3][3][2];              // 3 dimensional
int  hyper[3][3][4][2];         // 4 dimensional
```

- Usage

```
table[0][0] = 'A';
table[0][1] = 'C';
table[0][2] = 'Q';
table[score[2]/45][1] = table[0][2];
...
box[0][0][0] = 1;
...
```

Multi-dimensional Arrays

- Initialisation

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

1	4
2	5
3	6

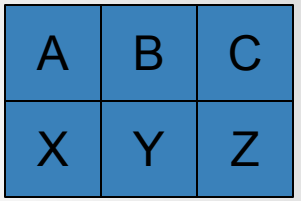
```
char values[][3] = { {'A', 'B', 'C'},
                     {'X', 'Y', 'Z'}
                   };
```

A	B	C
X	Y	Z

Multi-dimensional Arrays

- Processing

```
char values[][3] = { {'A', 'B', 'C'},
                     {'X', 'Y', 'Z'}
                   };

for (int row = 0; row < 2; row ++) {
    for (int col = 0; col < 3; col ++)
        cout << values[row][col];

    cout << endl;
}

for (int col = 0; col < 3; col ++) {
    for (int row = 0; row < 2; row ++ )
        cout << values[row][col];

    cout << endl;
}
```




```
ABC
XYZ
```

```
AX
BY
CZ
```

Multi-dimensional Arrays

- > 2 dimensions

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

for (int row = 0; row < 3; row ++)
    for (int col = 0; col < 2; col ++)
        for (int dep = 0; dep < 3; dep ++)
            cout << values[row][col][dep] << " ";
```