

C++: Some Syntax

Assignments

- Read documents on webpage
- If you are using Windows or a Mac, please test your projects on linux before you submit.

Where we are

- C++ has lots of similarities to Java (more as we go)
- C++ faster than Java
- How to write a simple C++ “Hello World” program
- How to compile using command line
- How to compile using an IDE
- Basic IDE usage (Debug/release builds, debugging, variables, breakpoints...)
- How to run a program
- **PRACTICE PLEASE**

Create project

Were we are going

Variables

Decisions and Loops(if,switch,for,while)

Functions

Arrays

Array Alternatives

Input (cin)

Variables

- Variables are defined
- But, no initialization guarantee

```
int cnt;  
if(cnt < 5)  
{ cnt++;}
```

- So **ALWAYS** initialize your variables

```
int cnt = 0;  
if(cnt < 5)  
{ cnt++;}
```

Variables - Common Built in Types

- int 16-32 bits **OS dependent!**
- double 64 bit real number
- bool true or false
- char 8-bit character, 'a' or '\0' to '\xFF'
or -128 to 127, note char is
surrounded by single quotes
- LOTS MORE (see readings on course
webpage)
 - http://www.tutorialspoint.com/cplusplus/cpp_data_types.htm

Open Page

Decisions - If

Syntax Same as Java

```
if (pad==0)
    conditionmet(pad);
else
    conditionNotMet(pad);
```

Decisions - Switch

Syntax Same as Java no easy equivalent in python

```
switch (x) {  
    case 1:  
        break;  
    case 2:  
        break;  
    default:  
}
```

```
switch (x) {  
    case 1:  
    case 2:  
    case 3:  
        cout << "x is 1, 2 or 3";  
        break;  
    default:  
        cout << "x is not 1, 2 nor 3";  
}
```


Const (Java equiv is final, no equiv in python)

const is a compiler enforced promise not to modify;

```
const int MYINT = 3;    //dandy
MYINT = 2;             //cannot modify
MYINT++;               //"  
const int MYINT2;      //must initialize when created  
MYINT2 = 5;            //cannot modify
```

Use const as often as possible

BTW Don't use magic numbers

- Magic number- don't know what it means

```
return 0;
```

**What does 0
mean?**

- Use self-documented const value

```
const int    SUCCESS = 0;  
return SUCCESS;
```

**indicates things
went well**

Loops - For

Syntax Same as Java

```
for (int n=0; n<NUMBER_TIMES; n++) {  
    if(conditionmet(n))  
        break;  
}
```

Loops - While

Syntax Same as Java

```
while (myCount>0) {  
    if (myDangerousArray[myCount]==SOUGHT_AFTER_VALUE)  
        break;  
    --myCount; //loop control  
}
```

Functions – Mostly Same as in Java

- The Rule is: **The compiler insists you declare everything before it is used.**
- Must see the function declaration before you call the function
- How?
 - Put function declaration in header and include header at top of file
 - Put function declaration before place where called
 - Either just the declaration
 - Or entire function
- **THIS IS FRUSTRATING FOR JAVA AND PYTHON PROGRAMMERS, ITS ALL THERE BUT DOES NOT WORK!**

[Go to Functions and demo](#)

Arrays – Similar to Java with a catch

- Groups a bunch of elements together
 - `T a[N]` //array of N elements of type T
- T can be any type or object
- Access `a[0]...a[N-1]`
- Problem is that they are not dynamically resizable

```
int  iInts[20];  
char cBuff[10];
```

Character Arrays

- Tricky to deal with, easy to get wrong
- Run time checks now (on some compilers)
- char use single quotes ‘
- char array, use double quotes “

```
char aChar = 'a';  
char cSrc[30] = "I like lemon custard";
```
- Terminate strings with ‘\0’ (note single quotes)
- Manipulate with strncpy. strcpy. strlen. sizeof.strcmp...

See <http://www.cplusplus.com/reference/cstring>

Include <string.h>

Character Arrays – Correct

```
//source string and intended destination
char cSrc[30] = "I like lemon custard";
char cDst[30];

int iLen1,iLen2=0;
iLen1 = strlen(cSrc); //size of string
iLen2 = sizeof(cSrc); //size of buffer

strcpy(cDst,cSrc); //copy the src to the dest,
strncpy(cDst,cSrc, sizeof(cDst)); //copy all 30 chars

//=0 same
//<0 cDst <cSrc
//>0 cDst >cSrc
int iRes = strcmp(cDst, cSrc);
```



Correct

Go to 3Basics and demo

Character Arrays – Crash Program

- Easy to get wrong, crashes now. In the past it did not.

```
//source string and intended destination
char cSrc[] = "I like lemon custard, and this string is lengthy";
char cDst[10];
```

```
//uhoh  cDst is not terminated, no worries
//cDst[10] = '\0'; //this will throw exception since
                //strings are 0 indexed
cDst[sizeof(cDst)-1] = '\0';
```

```
//here comes the bufferoverflow, copy more than 10 chars in
//because cSrc is much larger than cDst
strcpy(cDst,cSrc);                //boom! crashes
```

```
strncpy(cDst,cSrc,sizeof(cDst)); //copy only amount
                                //that fits
```



Buffer
Overflow

BTW a crash is the
best case scenario.
It could keep going

Go to bufferoverflow and demo

Strings – Special Characters

- strings enclosed in double quotes"""
- chars enclosed in single quotes"
- Characters with special meaning
 - '\n' newline equiv to std::endl
 - '\"' treat " as part of string not end of it
 - '\"' same as above
 - '\\ ' include a \ in the string
 - '\0' null

Arrays – An easier safer alternative

- Standard Library
- Use `std::string` if you need a string
- Use `std::vector` if you need array like functionality (more on this later)
 - Both - Dynamically Resizable
 - Both – Speed ranges from almost as fast to much faster as array based code
 - Easy to get right

```
char longbuff[] = "what if this is more than 10 chars?";  
std::string shortbuff = longbuff;
```

Input using std::string

```
// ask for a person's name, and greet the person
#include <iostream>
#include <string>

int main()
{
    // ask for the person's name
    std::cout << "Please enter your first name: ";

    // read the name
    std::string name;      // define `name'
    std::cin >> name;      // read into `name'

    // write a greeting
    std::cout << "Hello, " << name << "!" << std::endl;
    return 0;
}
```

Typedef – used a lot

- typedef - just a redefinition

```
typedef type newname;
```

```
typedef int feet;    //feet is another name for int  
feet length;
```

- Why? Simplifies syntax. Hides scary innards.

```
string outputfilename;
```

is really

```
typedef basic_string<char> string;
```

Enum – used a lot

- Defines a range of allowable values

```
enum enum-name { list of names } var-list;
```

- Why? Defensive programming, can only be one of defined values. NOTHING ELSE

```
//var of type color can only be one of 3 values  
enum color{ RED=1, GREEN, BLUE };  
  
color myEnumColor;  
myEnumColor = BLUE;
```

- myEnumColor can be RED, GREEN, or BLUE not 1,2, or 3!
- enum values also tend to be descriptive

Operators

- For this class mostly same as Java and Python

Assignment (=)

Arithmetic operators (+, -, *, /, %)

Compound assignment (+=, ...)

Increase and decrease (++ , --)

Relational and equality operators (==, !=, >, <, >=, <=)

Logical operators (!, &&, ||)

Operators

- **Conditional operator (?) – same as Java**

```
c = (a > b) ? a : b;
```

- **Bitwise Operators (&, |, ^, ~, <<, >>)**
 - useful for combining flags

Bitwise Operators (something new)

```
enum MyOptions {
    OpAutoRedraw    = 0x01,    // 0x01 == 1 == "00000001"
    OpAntiAlias     = 0x02,    // 0x02 == 2 == "00000010"
    OpPixelShader   = 0x04,    // 0x04 == 4 == "00000100"
    OpVertexShader  = 0x08,    // 0x08 == 8 == "00001000"
    OpFullscreen    = 0x10,    // 0x10 == 16 == "00010000"
    OpDaylight      = 0x20,    // 0x20 == 32 == "00100000"
    OpGlow          = 0x40,    // 0x40 == 64 == "01000000"
                        // 0x80 == 128 == "10000000" do not need to use all bits
};
```

```
int main() {
    //note this is a hex representation
    //always positive all bits are info, no sign bit
    unsigned char options = 0x00;

    //lets say you want to set (0x01) and fullscreen(0x10)
    options=options|OpAutoRedraw| OpFullscreen;

    //now options = 00010010

    //should succeed
    if (options & OpAutoRedraw)
        cout<<"OpAutoRedraw is set";

    //should fail
    if (options & OpAntiAlias)
        cout<<"OpAntiAlias is set";
}
```

See <http://www.cplusplus.com/forum/general/1590/>

Structs (like a class with public only members)

- User defined data type
- Set of data elements grouped under one name

```
struct product {  
    int weight;  
    float price;  
} ;  
  
product apple;  
product banana, melon;
```

- To access data members use .

```
apple.weight  
apple.price
```

- Convenient way to store chunks of related data

So Far

- Decisions and Loops
- Some Built in types
- Special chars
- Variables
- How to handle input (`cin>>...`)
and output (`cout<<...`)
- Do not use arrays
 - Use `std::string` for strings
 - Use vector as array substitute (**later**)
- Const, typedef, enum
- structs

What we can build

- Data processing app
- Standard I/O (no files)
- Bit manipulation
- Custom data types using structs