
EE450

Discussion #2

A Brief Introduction to Software
Development in C++

Introduction to C/C++

Strategies for learning C++

- Focus on concepts and programming techniques (Don't get lost in language features)
- Learn C++ to become a better programmer
 - More effective at designing and implementing
- C++ supports many different programming styles
- Learn C++ gradually
- Don't have to know every detail of C++ to write a good C++ program
- Implement full working examples to see how they work
 - See also here: <http://www.cplusplus.com/doc/tutorial/>

In a Nutshell: “Hello World” in C++

- Our first program in C++: “Hello World”

```
/* my first program in C++ with comments */  
#include <iostream>  
using namespace std;  
int main ()  
{  
    cout << "Hello World! ";    // prints Hello World!  
    return 0;  
}
```

- After you are connected to Nunki, create a text file named “hello.cpp”, and copy-paste the source code shown above
- In the command line, run: `g++ hello.cpp`
 - This generates an executable `a.out`. It can be executed with `./a.out`
- Alternatively, run: `g++ -o somename hello.cpp`
 - Run it with `./somename`

Comments

- How to make comments:
- In C:
 - `a = a + b; /* comment in C */`
- In C++:
 - `a = a + b; // line comment`
 - `a = a + b; /*block comment
 block comment
 block comment*/`

Variable declaration

- In C: all variable definitions must occur at the **beginning** of a block.

- Example:

- ```
int i;
for (i=0; i<5; i++) { ... }
```

- In C++: variable definitions may occur at the **point of use**.

- Example:

- ```
for(int i=0; i<5; i++) { ... }
```

Identifiers

- C++ reserved keywords that can not be used as an identifier:
- asm, auto, bool, break, case, catch, char, class, const, const_cast, continue, default, delete, do, double, dynamic_cast, else, enum, explicit, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, operator, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_cast, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t.
- and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq.
- far, huge, near (for some compiler).

Boolean

- Built-in type ***bool***:
 - In C: true is represented by nonzero integer values, and false by zero.
 - In C++: type ***bool*** is added to represent boolean values. A ***bool*** object can be assigned the literal value true and false
-

Boolean

- Example:
 - `int* f (int);`
 - `bool flag1, flag2;`
 - `flag1 = false;`
 - `flag2 = f(5);`

 - A zero value or a null pointer value can be converted to false implicitly; all other values are converted to true
-

Constant

- In C: Constants are handled by the preprocessor through macro substitution.
 - `#define MAX 10`

Declared Constants

- In C++: The keyword ***const*** allows explicit definition of constants objects.
- *// max* and *a* **cannot be modified after initialization**
- `const int max = 10;`
- `const int a = f(5);`
- `void f(int i, const int j)`
`// j is treated as a constant in f()`
`{`
 `i++; // ok`
 `j = 2; // error`
`}`

Cast: Type Conversion

- **Cast: Type conversion**
- In C: `(double) a;`
- **In C++: functional notation.**
- Example:
 - `average = double(sum) / double(count);`
- **Other C++ type conversion operator for Object Oriented programming:**
 - static cast, const cast, reinterpret cast, dynamic cast

Structure

- In C++: The keyword **struct** denotes a type (aggregation of elements of arbitrary type).
- Two structures are different even when they have the same members.

```
struct student{  
    char name[20];  
    int id;  
};  
student s1;  
struct new_student{  
    char name[20];  
    int id;  
};  
new_student s2;
```

s1 = s2; //Wrong! Two different types.

Function Prototype

- In C++, function prototyping is required to type-check function calls.
 - Format for prototype:
 - ***type name (argument_type1, argument_type2, ...);***
 - It does not include a *statement* for the function (no function body).
 - It ends with a semicolon sign (;).
 - In the argument enumeration it is enough to put the type of each argument.
-

Function Prototype

- **Version 1:** (function definition occurs before main, works even without prototype)
- `void f1(int);` // function prototype (declaration)
- `int f2(int x) // declaration & definition`
`{ return x + 10;}`
- `void f1(int x)// definition`
`{cout << x;}`
- `int main()`
`{f1(f2(10));}`

Function Prototype

- **Version 2:** (function definition occurs after main, not working)
- `int main()`
 `{f1(f2(10));}`
- `void f1(int);` // function prototype (declaration)
- `int f2(int x) // declaration & definition`
 `{ return x + 10;}`
- `void f1(int x) // definition`
 `{cout << x;}`

Function Prototype

- Better approach:
- `void f1(int);`
- `int f2(int);`
- `int main(){`
- `f1(f2(10));`
- `}`
- `int f2(int x) { return x + 10;}`
- `void f1(int x){cout << x;}`

Function Prototype

```
#include <iostream>
using namespace std;
void odd (int a); //void odd (int) is enough,, void odd (int x) also
OK.but
                                //not recommended.

void even (int a);
int main () {
    int i;
    do {
        cout << "Type a number: (0 to exit)";
        cin >> i; odd (i); }
    while (i!=0);
    return 0;
}
void odd (int a)
{ if ((a%2)!=0) cout << "Number is odd.\n"; else even (a); }
void even (int a)
{ if ((a%2)==0) cout << "Number is even.\n"; else odd (a); }
```

Function Prototype

- Two or more functions maybe given the same name provided the ***type signature*** (number of the arguments AND the type of the arguments) for each function is **unique**.
- Example:
 - ❑ `int multi (int, int);`
 - ❑ `double multi (double, double);`
 - ❑ `int multi (int, int, int);`
- How about:
 - ❑ `int add(int, int);`
 - ❑ `double add(int, int);`

Overloading functions is only applicable to C++

Namespace

- C++ provides a mechanism for **grouping** a set of global classes, objects and/or functions **under a name**. They serve to split the global scope in sub-scopes known as ***namespaces***.
- The C++ standard library is defined in namespace **std**.
- Advantage: **avoid name conflict and improve program modularity**.
- The form to use *namespaces* is:
 - **namespace** *identifier*
{
 namespace-body
}

Namespace

- To **access** name declared in a namespace, we have three approaches:
- **1. Prefix with scope operator ::**

```
#include <iostream.h>
namespace first
{   int var = 5; }
namespace second
{ double var = 3.1416; }
int main ()
{
    cout << first::var << endl;
    cout << second::var << endl;
    return 0;
}
```

Namespace

■ 2. Approach of *using directive*

```
#include <iostream.h>
```

```
namespace first
```

```
{ int var = 5; }
```

```
namespace second
```

```
{ double var = 3.1416; }
```

```
int main ()
```

```
{ using namespace second;
```

```
cout << var << endl;
```

```
cout << (var*2) << endl;
```

```
return 0; }
```

Namespace

- **using namespace** has validity only in the block in which it is declared.

```
int main ()  
{  
    { using namespace first; cout << var << endl; }  
    { using namespace second; cout << var << endl; }  
    return 0;  
}
```

Namespace

■ 3. Approach of *using declaration*

```
int main ()  
{  
    using first::var;  
    //using second::var;  
    cout << var << endl;  
    cout << second::var << endl;  
    return 0;  
}
```


String

- In C, a string is a null-terminated array of characters. It can be represented in two ways;
- 1) An array of type char
- 2) By a pointer of type char
- `char s1[] = "spring";`
- `char s1[7] = "spring";`
- `char s1[] = {'s','p','r','i','n','g'};`
- `char *s2 = "fall";`

String

- In C++, there is no built-in string data type. The C++ standard library provides library type string.
 - To use type string provided by the standard library, the header string must be included.
 - `#include <cstring>`
-

String

- Example:
- `#include <iostream>`
- `#include <cstring>`
- `using namespace std;`
- `string s1;`
- `string s2 = "hello!";`
- `string s3 = s2;`
- `string s4(5, 'x');`
- `string s5 = s2 + s3 // string concatenation`

String

- The string type provides a variety of useful string operations.

- For example:

```
String name= "windsor library";  
void m()  
{  
    string s=name.substr (0,6); //s="windsor"  
    name.replace (0,6, "windsor public"); //name="windsor public  
library";  
}
```

String

- The `c_str` function convert a string type to C-style string.
- `#include <iostream>`
- `#include <cstring>`
- `using namespace std;`
- `int main ()`
- `{`
- `string name="windsor";`
- `printf("name:%s\n", name.c_str());`
- `cout<<"name: "+name<<endl;`
- `return 0;`
- `}`

Simple I/O

- Input/output in C++ is supported by the use of I/O stream libraries.
- The stream library defines input/output for every built-in type.
- The standard output is defined as `cout` and standard input `cin`.

- “<<” put to
- “>>” get from

Simple I/O Example

- Example 1.
- //Input data from keyboard:
 cin >> x;
 cin >> x >> y;
- // Output data to screen
 cout << x;
 cout << "hello world!";
 cout << "The result is:" << GetResult();
 cout << "x is: " << x << "y is:" << y << endl;

Simple I/O Example

■ Example 2.

```
#include <iostream>
using namespace std;
int main() {
    int id;
    float av;
    char name[20];
    cout << "Enter the id, average and the name:";
    cin >> id >> av >> name;
    cout << "ID: " << id << endl << "Name: " << name << endl <<
        "Average: " << av << endl;
    return 0;}
```


Simple I/O Example

```
int main()
{
    string str;
    cout<<"Please enter your name:";
    cin>>str;
    cout<<"Hello, " <<str<<"!\n";
}
```

Input: "Harry Potter", what will be the output?

Simple I/O Example

- Using `getline()` function to read a **whole line**.

```
int main()
{
    string str;
    cout<<"Please enter your name:";
    getline(cin, str);
    cout<<"Hello, " << str << "!\\n";
}
```

Formatting Output

- C++ stream manipulators can be used to format the output.
 - The `<iomanip>` header file has to be included.
 - `#include <iomanip>`
-

Formatting Output

```
#include<iostream>
#include<iomanip> //Include header file iomanip.
using namespace std;
int main(){ double a[5];
    for ( int i=0; i<5; i++)
        a[i] = 3.1415926 * i * i * i;
    cout << "Output using default settings" << endl;
    for ( int i=0; i<5; i++)
        cout << i << " " << a[i] << endl;
    cout << "Output using formatted setting" << endl;
    cout << fixed << setprecision(2); //Use fixed and setprecision as //combination to
    set the precision of float number output.
    for ( int i=0; i<5; i++)
        cout << setw(2) << i << " " << setw(8) << a[i] << endl; //Use setw to set //the
    width of output.
    return 0; }
```

Formatting Output

- Output of the previous example:
- Output using default settings
- 0 0
- 1 3.14159
- 2 25.1327
- 3 84.823
- 4 201.062
- Output using formatted setting
- 0 0.00
- 1 3.14
- 2 25.13
- 3 84.82
- 4 201.06

Formatting Output

```
#include <iostream>
#include <iomanip>
#include <cmath> // sqrt prototype
using namespace std;
int main()
{
    double root2 = sqrt( 2.0 ); // calculate square root of 2
    int places;
    cout << "Square root of 2 with precisions 0-9." << endl
    << "Precision set by ios_base member-function "
    << "precision:" << endl;
    cout << fixed; // use fixed precision
```

Formatting Output

```
for ( places = 0; places <= 9; places++ ) {  
    cout.precision( places );  
    cout << root2 << endl;  
}  
cout << "\nPrecision set by stream-manipulator "  
<< "setprecision:" << endl;  
    // set precision for each digit, then display square root  
for ( places = 0; places <= 9; places++ )  
    cout << setprecision( places ) << root2 << endl;  
return 0;  
    } // end main
```

Formatting Output

- Square root of 2 with precisions 0-9.
- Precision set by `ios_base` member-function `precision`:
- 1
- 1.4
- 1.41
- 1.414
- 1.4142
- 1.41421
- 1.414214
- 1.4142136
- 1.41421356
- 1.414213562
-
- Precision set by stream-manipulator `setprecision`:
- 1
- 1.4
- 1.41
- 1.414
- 1.4142
- 1.41421
- 1.414214
- 1.4142136
- 1.41421356
- 1.414213562

File I/O

- The iostream library contains the file stream component which provides facilities for file I/O.
 - Object of type ifstream is defined to read from a file, and object of type ofstream to write to a file.
 - `#include <fstream>`
 - `ifstream infile;`
 - `ofstream outfile;`
 - `infile.open("input_file.name");`
 - `outfile.open("output_file.name") ;`
-

File I/O

- Operator >> and << are used in the same way as they are used in cin and cout for input/output. `int x;`
 - `infile >> x;`
 - `outfile << x;`
-

File I/O

- Example 1:
 - Write a program that reads an income from the file `income.in`, calculate the tax and output the income and tax to the file `tax.out`.
-

File I/O

```
#include <fstream>
using namespace std;
const int CUTOFF = 5000;
const float RATE1 = 0.3;
const float RATE2 = 0.6;
int main(){
    int income, tax;
    ifstream infile;
    ofstream outfile;
    infile.open( "income.in" );
    outfile.open( "tax.out" );
```

File I/O

```
while ( infile >> income )
{
    if ( income < CUTOFF )
        tax = RATE1 * income;
    else
        tax = RATE2 * income;
    outfile << "Income = " << income << " dollars\n"
    << "Tax = " << tax << " dollars\n\n";
}
infile.close();
outfile.close();
return 0;
}
```

File I/O

- Example 2:
 - Write a program that reads lines until end-of-file from a name file, sort the names, and then write the sorted name list to another file.
-

File I/O

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;
const int MaxSize = 1000;
void sort( string a[], int count);
```

File I/O

```
int main(){
    // get input from file
    string a[MaxSize];
    int count;
    ifstream infile;
    cout<< "Enter input file name: ";
    cin >> filename;
    infile.open( filename.c_str() );
    if( !infile ){
        cerr << "Can't open file " << filename << endl;
        exit(0);
    }
    int count;
    for( count = 0 ;
        count < MaxSize && getline( infile, a[count] );
        count ++ );
```


File I/O

Example 2 - cont'd

```
// sort and output
sort( a, count );
ofstream outfile;
cout << "Enter output file name: ";
cin >> filename;
outfile.open( filename.c_str() );
for ( int i=0; i<count; i++ )
    outfile << a[i] << endl;
infile.close();
outfile.close();
}
```

File I/O

Example 2 - cont'd

//Insertion sort

```
void sort( string a[], int count ){
    string temp;
    int i, j;
    for( i=0; i< count -1; i++ ){
        temp = a[i+1];
        for ( j = i; j>=0; j-- )
            if (temp < a[j] )
                a[j+1] = a[j];
            else
                break;
        a[j+1] = temp;
    }
}
```

File I/O

- **String Streams**
- By using a string stream, we can easily convert a number to string, or a string to number using << and >> operator.
- Example 1: From number to string:
- `float x = 3.1415926;`
- `ostringstream ostr;`
- `ostr << x;`
- `string output = ostr.str();`

File I/O

- **Example 2: From string to number:**
 - `string number = "1234.567";`
 - `istringstream instr(number);`
 - `double x;`
 - `instr >> x;`
-

File I/O

- We can use string streams to "break down" a line of input.

- Example:

```
#include <sstream> // must be included
```

```
#include <string>
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main(){
```

```
    istringstream buf( "A test 12 12.345" );
```

```
    string s1, s2;
```

```
    int x;  float y;
```

```
    buf >> s1 >> s2 >> x >> y; // white-space delimited input
```

```
    cout << s1 << endl<<s2 <<endl<< x<<endl << y<<endl; }
```

Header File

- Header File for C++
 - Header file names no longer maintain the **.h** extension.
 - Header files that come from the C language is preceded by a “**c**” character to distinguish from the new C++ exclusive header files that have the same name. For example **stdio.h** becomes **cstdio** .
 - All classes and functions defined in standard libraries are under the **std** *namespace* instead of being global.
-

Header File

- List of the standard C++ header files:
- <algorithm> <bitset> <deque> <exception> <fstream>
<functional> <iomanip> <ios> <iosfwd> <iostream>
<istream> <iterator> <limits> <list> <locale> <map>
<memory> <new> <numeric> <ostream> <queue> <set>
<sstream> <stack> <stdexcept> <streambuf> <string>
<typeinfo> <utility> <valarray> <vector>

Header File

- **// ANSI C++ example**

```
#include <cstdio>
using namespace std;
int main ()
{ printf ("Hello World!");
return 0; }
```

- **// pre ANSI C++ example**

```
// also valid under ANSI C++, but deprecated
#include <stdio.h>
int main ()
{ printf ("Hello World!");
return 0; }
```


Object-Oriented Programming

- First-class objects - atomic types in C
 - int, float, char
 - have:
 - values
 - sets of operations that can be applied to them
 - how represented irrelevant to how they are manipulated
- Other objects - structures in C
 - cannot be printed
 - do not have operations associated with them (at least, not directly)

Object-Oriented Idea

- Make all objects, whether C-defined or user-defined, first-class objects
- For C++ structures (called classes) allow:
 - functions to be associated with the class
 - only allow certain functions to access the internals of the class
 - allow the user to re-define existing functions (for example, input and output) to work on class

Classes of Objects in C++

■ Classes

- ❑ similar to structures in C (in fact, you can still use the struct definition)
- ❑ have fields corresponding to fields of a structure in C (similar to variables)
- ❑ have fields corresponding to functions in C (functions that can be applied to that structure)
- ❑ some fields are accessible by everyone, some not (data hiding)
- ❑ some fields shared by the entire class

Instances of Classes in C++

- A class in C++ is like a type in C
- Variables created of a particular class are instances of that class
- Variables have values for fields of the class
- Class example: Student
 - has name, id, gpa, etc. fields that store values
 - has functions, changeGPA, addCredits, that can be applied to instances of that class
- Instance examples: John Doe, Jane Doe
 - each with their own values for the fields of the class

Classes in C++

- Classes enable a C++ program to model *objects* that have:
 - attributes (represented by *data members*).
 - behaviors or operations (represented by *member functions*).
- Types containing *data members* and *member function* prototypes are normally defined in a C++ program by using the keyword *class*.

Classes in C++

- A class definition begins with the keyword *class*.
- The body of the class is contained within a set of braces, `{ };` (notice the semi-colon).
- Within the body, the keywords *private:* and *public:* specify the access level of the members of the class. Classes default to *private*.
- Usually, the data members of a class are declared in the *private:* section of the class and the member functions are in *public:* section.
- Private members of the class are normally not accessible outside the class, i.e., the information is hidden from "clients" outside the class.

Classes in C++

- A member function prototype which has the very same name as the name of the class may be specified and is called the constructor function.
 - The definition of each member function is "tied" back to the class by using the binary scope resolution operator (::).
 - The operators used to access class members are identical to the operators used to access structure members, e.g., the dot operator (.).
-

Classes in C++

```
#include <iostream>
#include <cstring>          // This is the same as string.h in C
using namespace std;
class Numbers    // Class definition
{
    public:        // Can be accessed by a "client".
        Numbers ( ) ;          // Class "constructor"
        void display ( ) ;
        void update ( ) ;
    private:      // Cannot be accessed by "client"
        char name[30] ;
        int a ;
        float b ;
} ;
```


Classes Example (continued)

```
Numbers::Numbers ( )      // Constructor member function
{
    strcpy (name, "Unknown") ;
    a = 0;
    b = 0.0;
}

void Numbers::display ( )    // Member function
{
    cout << "\nThe name is " << name << "\n" ;
    cout << "The numbers are " << a << " and " << b
        << endl ;
}
```

Classes Example (continued)

```
void Numbers::update ( ) // Member function  
{  
    cout << "Enter name" << endl ;  
    cin.getline (name, 30) ;  
    cout << "Enter a and b" << endl ;  
    cin >> a >> b;  
}
```

Classes Example (continued)

```
int main ( )           // Main program
{
    Numbers no1, no2 ;  // Create two objects of
                        // the class "Numbers"

    no1.update ( ) ;    // Update the values of
                        // the data members

    no1.display ( ) ;   // Display the current
    no2.display ( ) ;   // values of the objects
}
```

Example Program Output

> example.out

The name is Rick Freuler

The numbers are 9876 and 5.4321

The name is Unknown

The numbers are 0 and 0

More Detailed Classes Example

```
#include <iostream>
#include <cstring>
using namespace std;
class Numbers           // Class definition
{
    public:
        Numbers (char [ ] = "Unknown", int = 0, float = 0.0) ;
        void display ( ) ;
        void update ( ) ;
    private:
        char name[30];
        int a;
        float b;
};
```

More Detailed Classes Example (continued)

```
Numbers::Numbers (char nm[ ], int j, float k )
```

```
{  
    strcpy (name, nm) ;  
    a = j ;  
    b = k ;  
}
```

```
void Numbers::update ( )
```

```
{  
    cout << "Enter a and b" << endl ;  
    cin >> a >> b ;  
}
```

More Detailed Classes Example (continued)

```
void Numbers::display( )
{
    cout << "\nThe name is " << name << '\n';
    cout << "The numbers are " << a << " and " << b
        << endl ;
}
int main ( )
{
    Numbers no1, no2 ("John Demel", 12345, 678.9);
    no1.display ( ) ;
    no2.display ( ) ;
}
```

More Detailed Example Program Output

> example.out

The name is Unknown

The numbers are 0 and 0

The name is John Demel

The numbers are 12345 and 678.9

More Details on C++

- For a more **detailed** introduction, see here
 - <http://www.cplusplus.com/>
 - <http://www.cplusplus.com/doc/tutorial/>