505 22240 Data Structures: Lecture 1
Introduction and Review of C++ Programming

§ Introduction

◎What is Data Structure?

- **Algorithm**: an outline, the essence of a computational procedure or formula expressed as a finite sequences of steps.
- **Program**: an implementation of an algorithm in some programming language (C, C++, Java, …) for a specific task.
- **Data Structure**: a scheme for organizing and storing data required to solve a computational problem in a computer.

◎Some considerations for programming

1. How long does the program take to run?
2. How much memory does it need?
3. How long did it take to write the program?
4. What standard data formats does the program use?
5. How reliable do you want the program to be?
6. How do you pay for the mistakes this program makes?
7. How hard would it be to use some of that code in another program?

◎Questions for writing a program

1. How easy is the source code to read?
2. How easy is the program to debug?
3. How flexible is the program?
4. How easy is it to verify the program?
5. Did I learn something useful writing this program?
6. Was the program fun to write?
7. Is the program powerful and clean, something to be proud of?

§ Basic C++ Programming Element

◎Steps for C++ programming

1. Create a C++ source file whose name ends with the .cpp suffix.
2. Compiler, which creates a machine-code interpretation of this program.
3. Linker, which is typically invoked automatically by the compiler, includes any required library code functions needed and produces the final machine-executable file.
4. Request the system to execute the file.

◎A simple C++ program

```
#include <cstdlib>
#include <iostream>
/* This program inputs two numbers and outputs their sum.*/
int main(){
    int x, y;
    std::cout << “please enter two numbers:”;
    std::cin >> x >> y;       // input x and y
    int sum = x + y;          // compute their sum
    std::cout << “Their sum is ” << sum << std::endl;
    return EXIT_SUCCESS;  // terminate successfully
}
```

- Comments are indicated with two slashes (//).
  Longer block comments are enclosed between /* and */.
- Header files (“cstdlib” and “iostream”) are used to provide special declarations and definitions, which are of use to the program.
- The initial entry point for C++ programs is the function “main”.
- The function body is given within curly braces “{“ and “}”.
- The program terminates when the “return” statement is executed. Zero indicate success, nonzero failure.
- Standard error → std:: cerr

◎Fundamental Types

bool	Boolean value, either true or false
char	character
short	short integer
int	integer
long	long integer
float	single-precision floating-point number
double	double-precision floating-point number
enum	enumeration
void	indicate absence of any type

- bool, char, short, int, long, and enum are called integral types, which are capable of handling whole numbers.

★Characters

- A char variable holds a single character (8-bit).
- A literal is a constant value appearing in a program.
- Special character literals:

‘\n’	newline	‘\t’	tab
‘\b’	backspace	‘\0’	null
‘\’’	single quote	‘\”’	double quote
‘\\’	baclslash		

- The function int(ch) returns the integer value associated with a character variable ch.

★Integers

- An int variable holds an integer.
- Three sizes:

short (short int)	16bit
int	32bit
Long (long int)	≥ 32bit

- Suffix “l” or “L” can be added to indicate a long integer, e.g. 12345L.
- 256
  0400 : octal (base 8)
  0x100 : hexadecimal (base 16)

  } all represent the integer value 256 (in

- Declarations of integral variables:

```
short n;                        // n’s value is undefined
int octalNumber = 0400;         // 400 (base 8) = 256 (base 10)
char newline_character = ’\n’;
long BIGnumber = 314159265L;
```

- Given a type T, sizeof(T) return the size of type T, expressed as some number of

multiples of the size of char. e.g. typical char is 8 bits long, and int is 32 bits long, and hence sizeof(int) is 4.

★Enumerations

- An enumeration is a user-defined type that can hold any of a set of discrete values.
- Enumerations behave much like an integer type.
- e.g.

```
enum Day {SUN, MON, TUE, WED, THU, FRI, SAT};
enum Mood {HAPPY=3, SAD=1, ANXIOUS=4, SLEEPY=2};
Day today = THU;              // today may be any of MON, …, SAT
Mood myMood = SLEEPY;         // myMood may be HAPPY, …, SLEEPY
```

★Floating Point

- float holds single-precision floating-point number, “f” or “F”.
- double holds double-precision floating-point number (default).
- e.g.

```
double a = 3.14E5;      // (3.14×10^5)
float b = 2.0f;
float c = 1.234e-3F;
```

◎Strings

- Standard Template Library (STL) strings: <string>
- Example:

```
#include <string>
#include <iostream>
using std::string;      // we can omit the “std::” prefix
using namespace std;    //  makes std:: available, e.g. cout
string s = “to be”;
string t = “not ” + s;      // t = “not to be”
string u = s + “ or ” + t;  // u = “to be or not to be”
if (s>t)        // true: “to be” > “not to be”
  cout << u;    // outputs “to be or not to be”
string s = “John”;
int i = s.size();       // i = 4
char c = s[3];          // c = ”n”
s += “smith”;           // s = “John smith”
```

◎Namespaces

- A namespace is a declaration area that allows a group of related names to be defined in one place, helping to ensure that no duplicate global identifier names exist in a program.

```
namespace Name {MemberList}
```

- We can access an object **x** in namespace **group**, using the notation **group::x**, which is called its fully qualified name, e.g., (std::cin, std::cout).
- Example:

```
namespace myglobals {
  int cat;
  string dog = “bow wow”;
}
```

◎”Using” statement

- The **using** declaration tells the complier that you’re going to use some member of

some namespace.

```
using Name::Member;
```

· Example:

```
using std::string;
// makes just std::string accessible
using namespace myglobals;
//  makes all of myglobals accessible
```

§ C++ Program

◎Source Files: "**.cc**", "**.cpp**", and "**.C**"

· Source files may be complied separately by the complier, and these files are combined into one program by linker.

· Each nonconstant global variable and function may be defined only once.

· Use the type specifier "extern" to share a global variable (NOT function) in anther file.

· Example:

File: Source1.cpp

```
int cat = 1;        // definition of cat
int foo(int x) {return x+1;}    // definition of foo
```

File: Source2.cpp

```
extern int cat;     // cat is defined elsewhere
extern int foo(int x);
// foo is defined elsewhere, extern is optional
```

◎Header Files: "**.h**"

· A header file typically contains declarations, including classes, structures, constants, enumerations, and typedefs.

· Header files generally do not contain the definition (body) of a function, except in-line functions.

```
#include <iostream>          // system include file
#include "myIncludes.h"      // user-defined include file
```

◎Linux Complier and Linker:

① `g++ -o program program.cpp methods.cpp`

②
```
g++ -c program.cpp
g++ -c methods.cpp
g++ -o program program.o methods.o
```

◎Some Free C/C++ Compilers And IDEs:

· Code::Blocks: open source and cross platform,
  http://www.codeblocks.org/

· Eclipse: open source and cross platform, need Java,
  http://www.eclipse.org/

· CodeLite: like Code::Blocks C++ ide, cross platform IDE,
  http://codelite.org/

· NetBeans: open source and cross platform,
  https://netbeans.org/