COMS W3101 Programming Language: C++ (Fall 2015)

Ramana Isukapalli
ramana@cs.columbia.edu

Lecture-2

- Overview of C … continued
  - C character arrays
  - Functions
  - Structures
  - Pointers
- C++ string class
- C++
  - Design, difference with C
  - Concepts of Object oriented Programming
  - Concept of class and Object
  - Constructor and destructor
  - Data and Member functions
  - Data encapsulation
    - public, private and protected members

C - character arrays

C uses character arrays for strings

- C uses character arrays for strings.

C	O	L	U	M	B	I	A

- Useful string functions
  - strlen – find the length of a string.
  - strcmp – compares two strings
    - Returns 0 if they match.
  - strstr – check if a string is sub-string of another string.
  - strcat – concatenate two strings.
  - Many others

C++ string class

C++ strings

- C uses char arrays to represent strings
- char arrays are messy
  - Need to predefine the size of array
  - Size can't be increased easily for longer strings.
  - Copying strings need to use strcpy.
- C++ strings – don't have these issues.
  - E.g. string str1 = "abc";

    string str2 = str1;

    string str3 = str1 + "pqr";

  Much more convenient than C character arrays

C functions

- A group of statements
  - To perform a task
  - Possibly return a value

- Syntax

```
<return_type> fn_name (arguments)
    {
        // function body
    }
```

C functions … example

- Example

```
int square (int x)  /* fn to compute square*/
{
   return (x * x);
}
void main()         /* Starting point of ANY C program*/
{
   int i = 5;
   int i_sq = square (5);
   cout << "square of 5: " << i_sq << endl;
}
```

C pointers

- A pointer "points" to a memory location.
  - E.g., int x; /* x is an integer */

    int *y; /* y points to an integer */

    x = 5;

    y = 6; / NOT y = 6 ! */
- Pointers can point to any data type;
  - int *x; short *y; char *str;
  - double *z; void *p, etc.

C pointers, contd.

- Why do we need pointers?
  - Mainly to manage the memory, as opposed to the compiler managing memory.
  - User needs to assign and delete memory.
  - Allocate memory using malloc.
  - Delete memory using free.
- Examples
  - int *x = (int *) malloc (sizeof (int));
    *x = 3;
    free (x);

C structs

- C struct
  - used to contain > 1 basic data types
  - Can contain other structs
- E.g.,

```
typedef struct
{
   int a, b, c;
   float x,y,z;
} myStruct;

myStruct m;
m.a = 1;
```

C++

C++ — Philosophically different from C

- High level features of C++
  - Uses concepts of "object oriented programming" (OOP)
  - Everything that works in C works in C++
    - C syntax, operators, structures, control statements, etc. work in C++
    - Reverse is NOT true
- Object Oriented Programming
  - Concept of class/object, methods, inheritance, encapsulation, abstraction, polymorphism
  - Key concepts in this
    - Separation of data and methods

A simple "account" example

```
class account
{
  private:
     int user_SSN;                  // data
     int accountNumber;             // data
  public:
     void withdrawMoney (int amount);   // method
     void depositMoney (int amount);    // method:
     void computeInterest( );           // method
};
account x;  // x is an object of class "account"
```

Constructor and Destructor

Constructor and destructor … contd.

Constructor

- A function with the same name as the class
- Called when an object is created
- A class can have more than one constructor

Destructor

- Called when an object is cleaned up (goes out of scope)
- One class can have only one destructor

Examples

```
account x; // constructor code is called
account *y = new account; // constructor code is called
delete (y); // destructor code is called
```

Constructor and destructor

Constructor code

```
account::account( )
{ user_ssn = -1; accountNumber = -1; }

account::account( ) : user_ssn (-1),
                      accountNumber(-1) { }
account::account (int ssn, int acctNum)
{
    user_ssn = ssn;
    accountNumber = acctNum;
}
```

Destructor code

```
account::~account( )
{  // Any memory/resource cleanup, etc. }
```

Initializing member values

```
class Account
  {
     private:
       int balance;
     public:
       Account ( ) : balance (0)
         { }
       Account (int amount) :
   balance (amount) { }
  };
```

```
class checkingAccount : public
    Account
{
   checkingAccount (int amount)
     :  Account (amount) { }
}
```

Class methods

Syntax:

```
<ret_type> class::functionName(args)
{
   // code
}
```

Method code can be present in class definition

- Outside the class definition
- In a separate file

Example

```
void account::withdrawMoney (int amount)
{
   // code
}
```