Introduction to Object Oriented Programming

Lecture 3

Structures

Unions

Classes

Structures

- Structure is a collection of variables referenced under a common name.
- Sometimes, some logically related elements need to be treated under one single unit.
- For instance, the elements that store student information (e.g., name, class, marks and grade) need to be processed together under one roof.

Defining A Structure

```
struct tag-name
{
        Member1 ;
        Member2 ;
        Member3 ;
} ;
```

- **struct** is a required keyword to define a structure.
- **tag-name** is the name that identifies the structure.
- **member1, member2 and member3** are the members of the structure.

Example

```
struct stdrecord
{
    int class ;
    char name[40] ;
    float marks ;
} std ;
```

- **std** is structure variable of type **stdrecord.**

Referencing Structure Elements

- Once a structure has been defined, its members can be accessed with the use of dot (**.**) operator.

Syntax:

structure-name.element-name ;

Example:

std.name = “Ahmed” ;

std. class = 2 ;

- The structure variable name followed by a period (.) and the element name is the name of the specific structure variable.

Structures In C++

- Structures in C++ differ from those in C in that members can be functions.
- A special member function is the “constructor”, whose name is the same as the structure. It is used to initialize the object:

```
struct buffer {
    buffer(){size=MAXBUF+1; A=B=0;}
    char buf[MAXBUF+1];
    int size, A, B;
    }
```

Structures In C++

```
#include <iostream>
using namespace std;
#define MAXBUF 5
struct buffer {
      buffer() {size=MAXBUF+1; A=B=0;}
      char buf[MAXBUF+1];
      int size, A, B;
      };
void main()
{
  buffer x;
  cout<<x.A <<"\t"<<x.B <<"\t"<<x.size <<endl;
}
```

Structures In C++

- The definition (body) of a member function can be included in the structure's declaration, or may appear later. If so, use the name resolution operator (::)

```
void  buffer::enter(int p) {
    size = p;
    A = p;
    B = p;
}
```

Structures In C++

```
#include <iostream>
using namespace std;
#define MAXBUF 5
struct buffer {
         buffer() {size=MAXBUF+1; A=B=0;}
         char buf[MAXBUF+1];
         int size, A, B;
         void inc(int i)  { size+=i;}
         void enter(int p);
         };
void  buffer::enter(int p) {
     size=p;     A=p;    B=p;
}
void main()
{
   buffer x;
   cout<<x.A <<"\t"<< x.B <<"\t"<< x.size <<endl;
   x.enter(100);
   cout<<x.A <<"\t"<< x.B <<"\t"<< x.size <<endl;
   x.inc(100);
   cout<<x.A <<"\t"<< x.B <<"\t"<< x.size <<endl;
}
```

Unions

- A union is a memory location that is shared by several variables that are of different types. The union definition is similar to that of a structure:

```
union union-name
{
        Member1 ;
        Member2 ;
        Member3 ;
} ;
```

Example

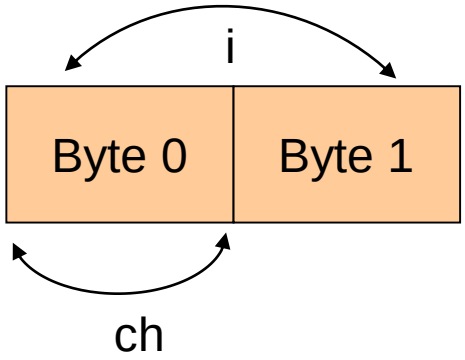
```
union asciicode
{
    int i ;
    char ch ;
} Ascii ;
```

- **Ascii** is union variable of type **asciicode.**

- When a union is declared, the compiler automatically creates a variable large enough to hold the largest variable type in the union.

Unions

- In example **asciicode**, compiler will reserve 2-bytes since (i) occupies 2 bytes.
- The figure below shows how (i) and (ch) share the same address.



- In C++ union are used frequently when type conversion are necessary.

Ex: Ascii.ch = ‘A’ ; cout << Ascii.i ; output = 65

Abstract Data Types

- A **data type** consist of a collection of values together with a set of basic operations defined on these values.

- A data type is called an **abstract data type (ADT)** if the programmers who use the type do not have access to the details of how the values and operations are implemented

Classes & Defining a Class

- A **class** is a data type whose variables are objects.
- An **object** is a variable that has member functions as well as the ability to hold data values.
- The definition of a class should be a data type definition that describes what kinds of values the variables can hold and the operations on these values.
- These operations are carried out by and referred to as **member functions**.

Defining a Class in C++

- Classes in C++ evolve from the C notion of structures (referred to as records in Pascal).
- The keyword **struct** intoduces the structure definition.
- Structures are data type aggregations built using elements of other types. For example:

```
struct Time {
   int hour;        //  0-23
   int minute;      //  0-59
   int second;      //  0-59
};
```

Accessing Members of Structures

- Members of a structure(or class) are accessed using the dot operator (**.**) and the arrow operator (**– >**).

```
cout << mytime.hour;
cout << time_ptr->hour;
```

Problems with Structures

- It is possible to have uninitialized data.
- If a **struct** implementation is changed, all programs that use it must be changed.
- There is no “interface” to insure that the programmer uses the data type correctly and to insure that the data remains “stable”.

Class Definitions

- Classes enable the programmer to model objects that have *attributes* (represented as *data members*) and *behaviors* or *operations* (represented as *member functions*).

- Types containing data members and member functions are defined in C++ using the keyword **class**.

Class Time

```
class Time {
  public:
      Time( );
      void setTime( int, int, int );
      void printStandard( );
  private:
      int hour ;          // 0-23
      int minute ;        // 0-59
      int second ;        // 0-59
} ;
```

Class Member Access

- The **public:** and **private:** labels are called *member access specifiers*.
- Any member identifier declared after member access specifier **public** is accessible wherever the program has access to the object.
- Any member identifier declared after member access specifier **private** is accessible only to member functions of the class.
- Member access specifiers are always followed by a colon(**:**).
- Member access specifiers can appear multiple times and in any order in the class definition.
- The default access specification is **private**.

Time Member Functions

- Time contains prototypes for three *member functions* after the the **public** access specifier: **Time**, **setTime**, and **printStandard**.
- These are referred to as the *behaviors*, *services* or *interface* of the class.
- These functions are used by *clients* of the class to manipulate the data of the class.

Class Constructor

- A member function with the same name as the class is called a *constructor* function for the class.
- A constructor is a special member function that initializes the data members of a class object.
- The class constructor is called automatically when an object of that class is created.
- Constructors can contain default arguments
- By providing default arguments to the constructor, even if no arguments are provided in a constructor call, the object is still initialized to a consistent state.

Default Constructor

- A programmer-supplied constructor that defaults all its arguments(or explicitly requires no arguments) is called a *default constructor*.

- Hence, a constructor that can be invoked with no arguments is a default constructor.

- There can only be one default constructor per class.

Constructor with default args

```
class Time {
  public:
      Time( int = 0, int = 0, int =0 );
      void setTime( int, int, int );
      void printStandard( );
  private:
      int hour ;          // 0-23
      int minute ;        // 0-59
      int second ;        // 0-59
} ;
Time::Time( int hr, int min, int sec )
{
  setTime( hr, min, sec ) ;
}
```

setTime( ) & printStandard( ) Definition

```
void Time::setTime( int h, int m, int s )
{
    hour = (h >= 0 && h < 12) ? h : h % 12 ;
  minute = (m >= 0 && m < 60) ? m : m % 60 ;
  second = (s >= 0 && s < 60) ? s : s % 60 ;
}
void Time::printStandard()
{
  cout<<hour<<":"<< minute<<
  ":"<<second<<endl;
}
```

Time Data Members

- Three integer members appear after the **private** access specifier: **hour**, **minute**, and **second**.

- Thus, these members are only accessible by member functions of the class(and “friends” of the class).

The End