Lab 4: Class and objects & Structure in C++

A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and functions within a class are called members of the class.

A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword **class** as follows:

```
class Box
{    public:
      double length;    // Length of a box
double breadth;  // Breadth of a box
double height;    // Height of a box };
```

The keyword **public** determines the access attributes of the members of the class that follow it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as **private** or **protected**.

Define C++ Objects:

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box:

```
Box Box1;               // Declare Box1 of type Box
Box Box2;               // Declare Box2 of type Box
```

Both of the objects Box1 and Box2 will have their own copy of data members.

Data Abstraction

Data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user. C++ supports the properties of encapsulation and data hiding through the creation of user-defined types, called **classes**. We already have known that a class can contain **private, protected** and **public** members. By default, all items defined in a class are private. For example:

```
class Box
{     public:
       double getVolume(void)
       {
        return length * breadth * height;
       }
     private:
       double length;       // Length of a box
       double breadth;      // Breadth of a box
       double height;       // Height of a box
};
```

The variables length, breadth, and height are **private**. This means that they can be accessed only by other members of the Box class, and not by any other part of your program. The public member **getVolume** is the interface to the outside world and a user needs to know them to use the class. This is one way encapsulation is achieved.

To make parts of a class **public** (i.e., accessible to other parts of your program), you must declare them after the **public** keyword. All variables or functions defined after the **public** specifier are accessible by all other functions in your program. There are no restrictions on how often an access label may appear. Each access label specifies the access level of the succeeding member definitions. The specified access level remains in effect until the next access label is encountered or the closing right brace of the class body is seen.

The Class Constructor

A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class. A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

The Class Destructor

A **destructor** is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class. A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Following example explains the concept of constructor and destructor:

```
#include <iostream> using
namespace std;

class Line
{    public:
      void setLength( double len );
double getLength( void );
      Line();    // This is the constructor declaration
Line(double len);
      ~Line();   // This is the destructor: declaration

     private:
      double length;
};

// Member functions definitions including constructor
Line::Line(void)
{
    cout << "Object is being created" << endl;
}
Line::Line( double len)
{
    cout << "Object is being created, length = " << len
<< endl;
    length = len; }
Line::~Line(void)
{
    cout << "Object is being deleted" << endl;
}
void Line::setLength( double len )
{
    length = len;
}
double Line::getLength( void )
{
    return length;
}
// Main function for the program void
main( )
{
   Line line;

   // set line length    line.setLength(6.0);
   cout << "Length of line : " << line.getLength()
<<endl;
}
```

Part A: Class and Object exercise

Exercise 1:
We want to maintain the bank account. To that end, consider the following skeletal definition of a class.

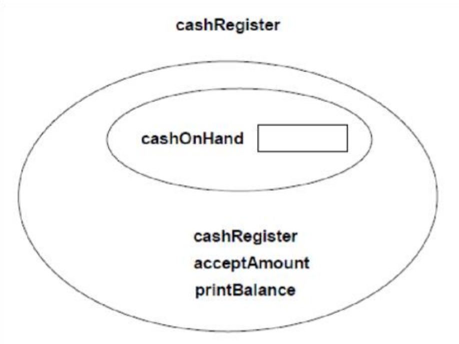
```
class BankAccount
{    private:
      double balance;
double interest_rate;
    public:
      void set(int dollars, int cents); //The account balance is set to
$dollars.cents;
      void update(); //One year of simple interest is added to account
balance
      double get_balance(); //Return the current account balance
      double get_rate(); //Return the current interest rate
      void set_rate(int rate); // rate is set to interest rate
};
```

Write the code for each member function.

Exercise 2
The objective is to define a class **cashRegister** for a candy machine. The register has an initial cash of 500 units. It accepts an amount of cash from the customer. The following operations are to be performed:

1. Set an initial balance of 500 cashOnHand.
2. Accept an amount from the customer, and update the amount in the register.
3. Print the current balance.

A conceptual diagram appears below.



Exercise 3
In a population, the birth rate and death rate are calculated as follows:

Birth rate = Number of births / population
Death rate = Number of deaths / population

For example, in a population of 100,000 that has 8,000 births and 6,000 deaths per year, the birth rate and death rate are:

Birth rate = 8,000 / 100,000 = 0.08

Death rate = 6,000 / 100,000 = 0.06

Design a **population** class that stores a population, number of births, and number of deaths for a period of time. Member functions should return the birth rate and death rate. Implement the class in a program.
Input validation: Do not accept population figures less than 1 or birth/ death numbers less than 0.

Exercise 4
A new fruit juice machine has been purchased for the cafeteria, and a program is needed to make the machine function properly. The machine dispenses apple juice, orange juice, mango lassi, and fruit punch in recyclable containers. Write a program for the fruit juice machine so that it can be put into operation.

The program should do the following:

1. Show the customer the different products sold by the juice machine.
2. Let the customer make the selection.
3. Show the customer the cost of the item selected.
4. Accept money from the customer.
5. Release the item.

A juice machine has two main components: a built-in cash register and several dispensers to hold and release the products.

cashRegister class has the following member variable and functions:

- cashOnHand – an int variable that holds the cash in the register.
- getCurrentBalance – Function to show the current amount in the cash register.
- acceptAmount - Function to receive the amount deposited by the customer and update the amount in the register.
- cashRegister() - Default constructor to set the cash in the register to 500 cents.
- cashRegister(int) - Constructor with parameters to set the cash in the register to a specific amount.

dispenserType class has the following member variable and functions:

- numberOfItems – variable to store the number of items in the dispenser.
- cost - variable to store the cost of an item.
- getNoOfItems- Function to show the number of items in the machine.
- getCost - Function to show the cost of the item.
- makeSale - Function to reduce the number of items by 1.
- dispenserType() - Default constructor to set the cost and number of items in the dispenser to 50.
- dispenserType(int, int) - Constructor with parameters to set the cost and number of items in the dispenser.

The dispenser should show the number of items in the dispenser and the cost of the item. If the dispenser is empty, the program should inform the customer that this product is sold out.
The dispenser releases the selected item if it is not empty.

Part B: Structure exercise

A structure is a ***user-defined data type*** in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type. The ‘struct’ keyword is used to create a structure.

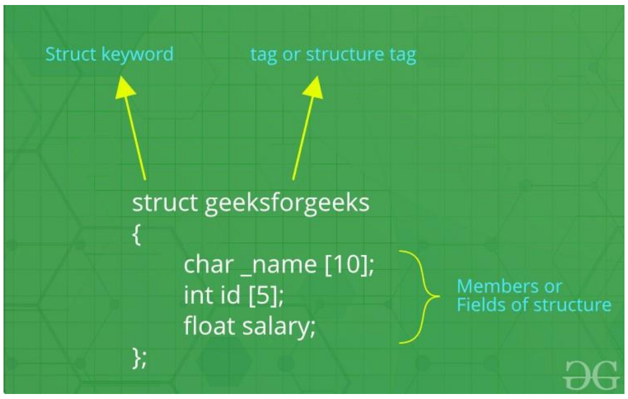


Figure 1: structure in C++

What is a structure pointer?

Like primitive types, we can have pointer to a structure. If we have a pointer to structure, members are accessed using arrow (->) operator instead of the dot (.) operator.

Part 1: Learn how to use the pointer to access the structure members

1.
```
#include <iostream>
using namespace std;

struct Point {
        int x, y;
};

int main()
{
        //p1 is a pointer to a new Point structure
        Point * p1 = new Point;

        //To access the new structure members using pointer
        p1->x = 3; //insert value
        p1->y = 4;

        cout << p1->x << " " << p1->y; // to display
        return 0;
}
```

[Estimate Finish Time: 5 minutes]

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:

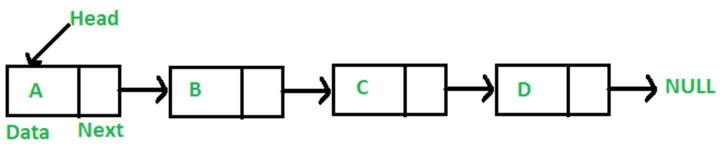


Figure 2: Linked list sample

Part 2: Understand the concept of linked-list

1. Learn how to connect two independent structures to become a short linked-list.

```
#include <iostream>
using namespace std;

struct Point {
        int x, y;
        Point * nextaddress;
};

int main()
{
        //first structure
        Point * p1 = new Point;
        p1->x = 3;
        p1->y = 4;
        p1->nextaddress = NULL;

        //second structure
        Point * p2 = new Point;
        p2->x = 7;
        p2->y = 16;
        p2->nextaddress = NULL;

        //to link the first structure with second structure
        p1->nextaddress = p2;

        //display the p1 information
        cout << "P1 info : \n ----------------\n";
        cout << "P1 Address : " << p1 << endl;
        cout << "P1 x value : " << p1->x << endl;
        cout << "P1 y value : " << p1->y << endl;
        cout << "P1 nextaddress value : " << p1->nextaddress << endl << endl;

        ////display the p2 information
        cout << "P2 info : \n ----------------\n";
        cout << "P2 Address : " << p2 << endl;
        cout << "P2 x value : " << p2->x << endl;
        cout << "P2 y value : " << p2->y << endl;
        cout << "P2 nextaddress value : " << p2->nextaddress << endl << endl;
}
```

[Estimate Finish Time: 15 minutes]

2. Learn how to display the linked structures using a structure pointer.

```
#include <iostream>
using namespace std;

struct Point {
        int x, y;
        Point * nextaddress;
};

int main()
{
        //first structure
        Point * p1 = new Point;
        p1->x = 3;
        p1->y = 4;
        p1->nextaddress = NULL;

        //second structure
        Point * p2 = new Point;
        p2->x = 7;
        p2->y = 16;
        p2->nextaddress = NULL;

        //to link the first structure with second structure
        p1->nextaddress = p2;

        //display the p1 information
        cout << "P1 info : \n ----------------\n";
        cout << "P1 Address : " << p1 << endl;
        cout << "P1 x value : " << p1->x << endl;
        cout << "P1 y value : " << p1->y << endl;
        cout << "P1 nextaddress value : " << p1->nextaddress << endl << endl;

        ////display the p2 information
        cout << "P2 info : \n ----------------\n";
        cout << "P2 Address : " << p2 << endl;
        cout << "P2 x value : " << p2->x << endl;
        cout << "P2 y value : " << p2->y << endl;
        cout << "P2 nextaddress value : " << p2->nextaddress << endl << endl;

        //to display: create a new pointer 'head' to follow the list from p1 to p2
        Point * head = NULL;
        head = p1;

        while (head != NULL)
        {
                cout << head->x << " , ";
                cout << head->y << " , ";
                cout << head->nextaddress << endl;
                head = head->nextaddress;
        }

}
```

[Estimate Finish Time: 15 minutes]