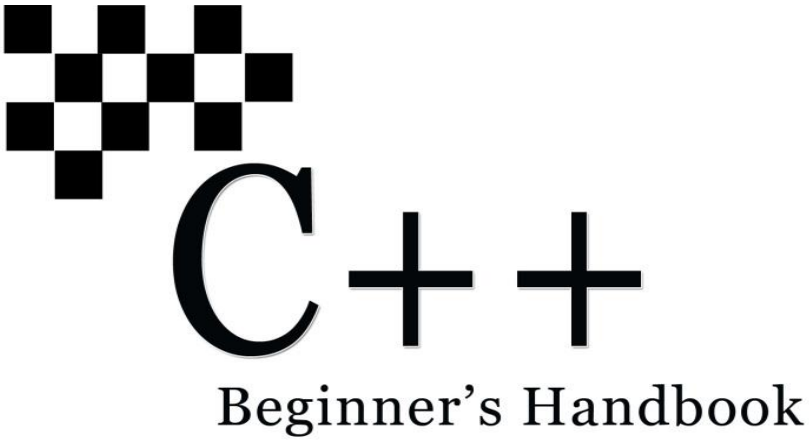
Northern City



cin/cout Samples

Sample1

```
#include <iostream>

using namespace std;

int main()
{
 cout<<"HEY, you, I'm alive! Oh, and Hello World!"<<endl;

}
```

Sample 2

```
#include <iostream>

using namespace std;

int main()

{

cout << "my first C++ program" << endl;

return 0;

}
```

Do the following Exercises

Exercise1.cpp

Output:

```
MENU
[1] Add Numbers
[2] Subtract Numbers
[3] Multiply Numbers
[4] Divide Numbers
[5] Exit program
```

Exercise2.cpp

Output:

```
YOMA BANK
[1] Open Account
[2] Deposit Account
[3] Update Account
[4] Close Account
[5] Exit Program
```

Sample 3

```
#include <iostream>

using namespace std;

int main()
{
 int thisisanumber;

 cout<<"Please enter a number: ";
 cin>> thisisanumber;
 cout<<"You entered: "<< thisisanumber <<"\n";


}
```

Sampl 4

```
#include <iostream>

using namespace std;

int main()
{
 int num1,num2j,sum;

 cout<<"Please enter first number: ";
 cin>> num1;


 cout<<"Please enter second number: ";
 cin>> num2;


 sum=num1+num2;

 cout<<"You entered: "<< sum <<"\n";



}
```

Do the following Exercises:

Exercise 1.cpp

Multiplying Numbers

Enter Input1: 3

Enter Input2: 3

The Answer is : 9

Exercise2.cpp

Divide Numbers

Enter Input1: 5

Enter Input2: 4

The Answer is : 1.25

Conditional Statements

Sample 1

```
#include <iostream>

using namespace std;

int main()                    // Most important part of the program!
{
 int age;                     // Need a variable...

 cout<<"Please input your age: ";    // Asks for age
 cin>> age;                   // The input is put in age

 if ( age < 100 ) {           // If the age is less than 100
   cout<<"You are pretty young!\n"; // Just to show you it works...
 }
 else if ( age == 100 ) {     // I use else just to show an example
   cout<<"You are old\n";     // Just to show you it works...
 }
 else {
  cout<<"You are really old\n";     // Executed if no other statement is
 }

}
```

Sample 2

```
#include <iostream>

using namespace std;

int main(){

int input1,input2,largerNo;

 cout<<"Please input 1: ";          // Asks for input1
 cin>> input1;
```

```
 cout<<"Please input 2: ";            // Asks for input2
 cin>> input2;


if(input1>input2){

    largerNo=input1;

}

else{

   largerNo=input2;

}

Cout<<"The larger input is:"<<largerNo<<"\n";

}
```

Do the following Exercises

Exer1.cpp

Comparing four Inputs

Enter input 1: **2**

Enter Input2: **15**

Enter Input 3: **82**

The Largest Number is : 82

Exer2.cpp

Comparing five Inputs

Enter input 1: **2**

Enter Input2: **15**

Enter Input 3: **82**

Enter Input 4: **12**

The Largest Number is : 621

Exer3.cpp

Comparing five Inputs

Enter input 1: **2**

Enter Input2: **15**

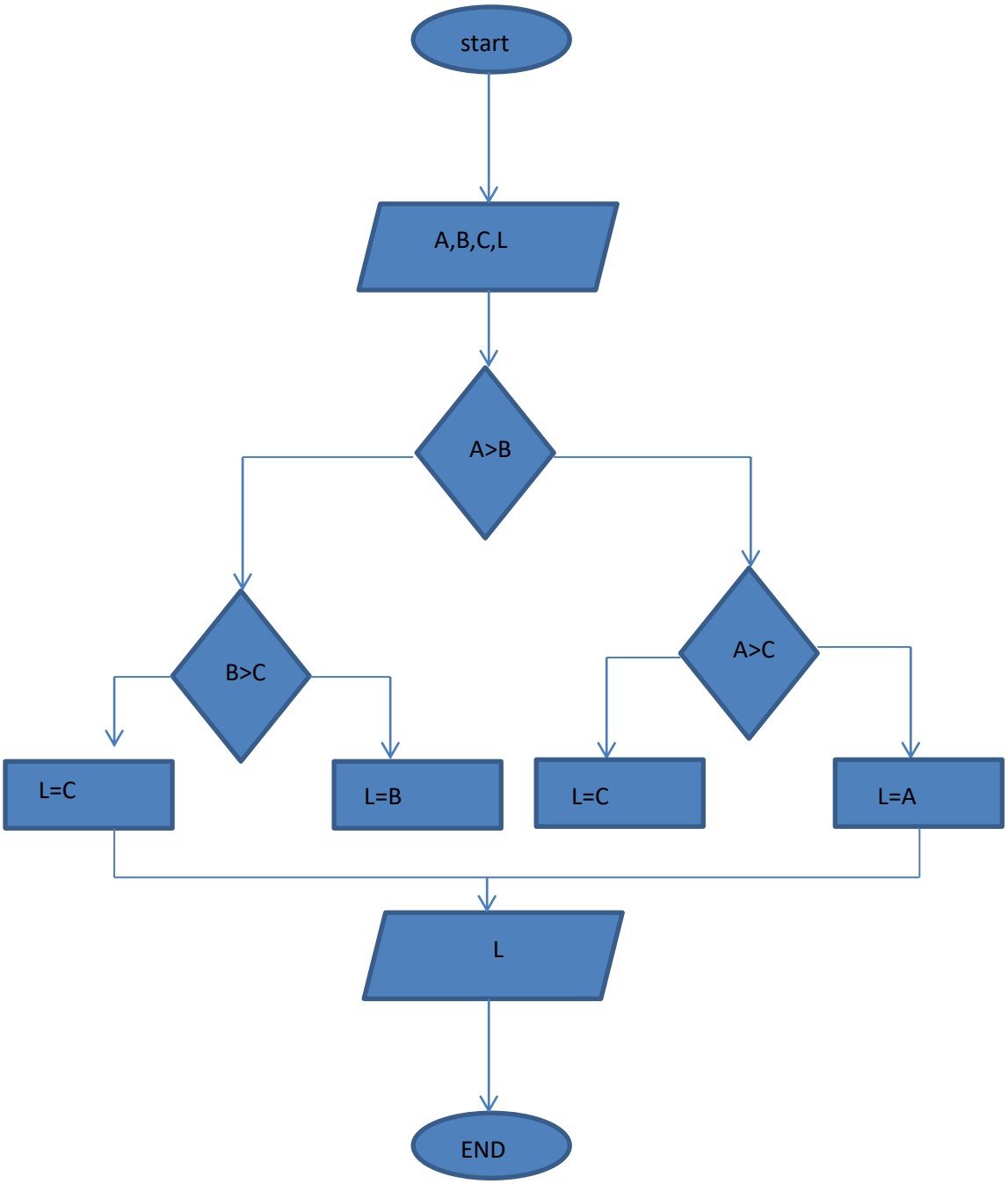
Enter Input 3: **82**

Enter Input 4: **12**

Enter input 5: **621**

The Largest Number is : 621

Pseudo code



Algorithm 1

C++ Coding

```
if(A>B){
        if(A>C){
                L=A;
        }
        else{
                L=C;
        }
}
else{
        if(B>C){
                L=B;
        }
        else{
                L=C;
        }
}
```

Algorithm 2

C++ Coding

```
int A,B,C,D;

int L=0;

if((A>B)&&(A>C)&&(A>D)){

        L=A;

}

else if((B>A)&&(B>C)&&(B>D)){

        L=B;

}

else if((C>A)&&(C>B)&&(C>D)){

        L=C;

}

else{

        L=D;

}
```

Algorithm 3

```
L=0;
if(A>B){
        L=A;
}
else{
        L=B;
}
if(C>L){
        L=C;
}
else{
        L=L;
}
if(D>L){
        L=D;
}
else{
        L=L;
}
```

Switch Statement

```
switch(condition){

case 1: statement ;
        Break;

case 2: statement;
        Break;

case 3: statement;
        Break;

        .
        .
        .
Default: st


}
```

Switch Sample Exercise1

```
#include <iostream>

using namespace std;

int main(){

        int choice=0;

                cout<<" \t   MENU    \n ";
                cout<<"[1] Add Number \n");
                cout<<"[2] Subtract Number \n");

                cout<<"Enter Choice:");
                cin>>choice;

                        switch(choice){
                                case 1: cout<<"Add Number";
                                        cout<<"Enter Num1:";
                                        cin>>num1;
                                        cin.ignore();

                                        cout<<"Enter Num2:";
                                        cin>>num2;
                                        cin.ignore();

                                        ans1=num1+num2;
                                        cout<<"The answer is:"<<ans1<<endl;
                                        break;


                                case 2 : cout<<"Subtract Number";
                                         cin>>num3;

                                         cout<<"Enter Num4:";
                                        cin>>num4;

                                        ans2=num3-num4;
                                        cout<<"The answer is:"<<ans2<<endl;
                                         break;
```

```
                    }

                }

    }
```

Switch Sample Exercise 2

```
#include <iostream>

using namespace std;

Int main(){


Int num1=0,num2=0,num3=0,num4=0,num5=0,num6=0,num7=0,num8=0;

                    cout<<" \t MENU \n  ";
                    cout<<"[1] Add \n  ";
                    cout<< "[2] Substract \n  ";
                    cout<< "[3] Multiply \n  ";
                    cout<<"[4] Divided \n  ";
                    cout<< "[5] Exit program\n " <<endl;


                    cout<< "Enter Choice:";
                    cin>>choice;

                        switch(choice){
                            case 1: cout<<"Enter num1:";
                                    cin>>num1;
                                    cout<<"Enter num2:";
```

```
                cin>>num2;

                ans1=no1+no2;
                cout<<"The Sum is:"<<ans1<<endl;
                    break;

                case 2: cout<<"Enter num1:";
                    cin>>num3;

                    cout<<Enter num2:";
                    cin>>num4;

                ans2=no3-no4;
                cout<<"The Sum is:"<<ans2<<endl;

            break;

            case 3: cout<<"Enter num1:";
                    cin>>num5;

                cout<<Enter num2:");
                cin>>num6;
                ans3=no5*no6;
                cout<<"The Sum is:"<<ans3<<endl;


        break;

        case 4: cout<<"Enter  num1:");
                cint>>num7;

            cout<<"Enter Num2:";
            cin>>num8

            ans4=o7/no8;
            cout<<The Sum is:<<Sum4<<endl;

        break;

    case 5: cout<<"Thank You and Goodbye";
            exit(0);
        break;


    }
}
}
```

}

Do the following exercises

Main Menu

(My Mathematic Program)

[1] Circle

[2] Triangle

[3] Rectangle

Main choice==1
Circle Menu [1] Area of Circle [2] Circumference of Circle Enter Choice:

Main choice==2
Triangle Menu [1] Area of Triangle [2] Volume of Triangle Enter Choice:

Main choice==3
Rectangle Menu [1] Area of Rectangle [2] Volume of Rectangle Enter Choice:

Circle choice == 1
Area of Circle Enter Radius: 3 The Area of the Circle is : 28.26

Circle choice == 2
Circumference of Circle Enter Radius: 3 The Area of the Circle is : 18.84

Formulas
Area=pi*r*r; Cir=2*pi*r;

Triangle choice == 1
Area of Triangle Enter base: 3 Enter height: 3 The Area of the Triangle is : 4.5

Triangle choice == 2
Volume of Triangle Enter base: 3 Enter height: 3 The Volume of the Trianle is : 3

Formulas
Area=(b*h)/2 vol=(b*h)/3

Rectangle choice == 1
Area of Rectangle Enter width: 3 Enter height: 3 The Area of the Rectangle is : 9

Rectangle choice == 2
Volume of Rectangle Enter height: 3 Enter base: 3 Enter width: 3 The Volume of the Tectangle is : 27

Formulas
Area=w*h; Vol=b*h*w

Loops

1) For loop
2) while loop
3) do-while loop

For loop sample1

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

Int main(){

        Int I;

         cout<<"output is:";
          for(i=0;i<10;i++){

            cout<<"loop no "<<i<<"\n";


          }

    }

}
```

```
Output is:
Loop no 0
Loop no 1
Loop no 2
Loop no 3
Loop no 4
Loop no 5
Loop no 6
Loop no 7
Loop no 8
Loop no 9
```

For loop sample 2

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

int main(){

        int row,col;
                cout<<"output is:";

                for(row=0;row<5;row++){
                        for(col=0;col<5;col++){
                                if(row==0||row==4){
                                        cout<<" * ";

                                }
                                if(row>0&&row<4){
                                        if(col==0||col==4){
                                                cout<<" * ";

                                        }
                                        else{
                                                cout<<"  ";

                                        }

                                }

                        }
                        cout<<"\n";

                }


        }
}
```

Output is:

```
* * * * *
*       *
*       *
*       *
* * * * *
```

For Loop Sample 3

```
#include<iostream>

Using nameapace std;

Int main(){

        Int row,col;

                        cout<<"output is:";

                         for(row=0;row<5;row++){
                                for(int col=0;col<5;col++){
                                        if(row==0||row==2||row==4){
                                                cout<<" * ";

                                        }
                                        if((row>0&&row<2)||(row>2&&row<4)){
                                                if(col==0||col==4){
                                                        cout<<" * ";

                                                }
                                                else{
                                                        cout<<"  ";

                                                }

                                        }

                                }
                                cout<<"\n";

                         }

                }

        }
```

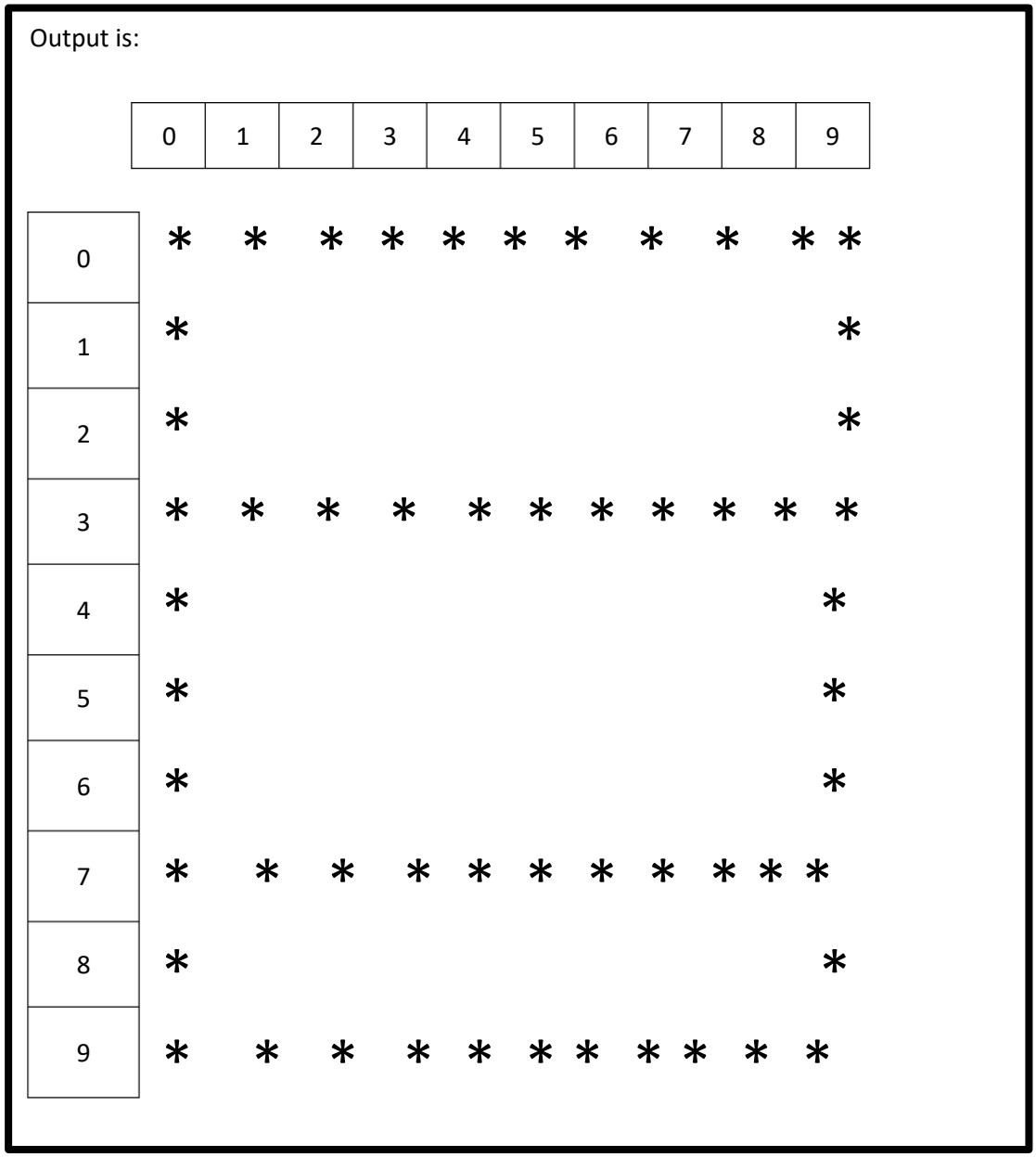
Output is:

```
* * * * *
*       *
* *  * * *
*       *
* * * * *
```

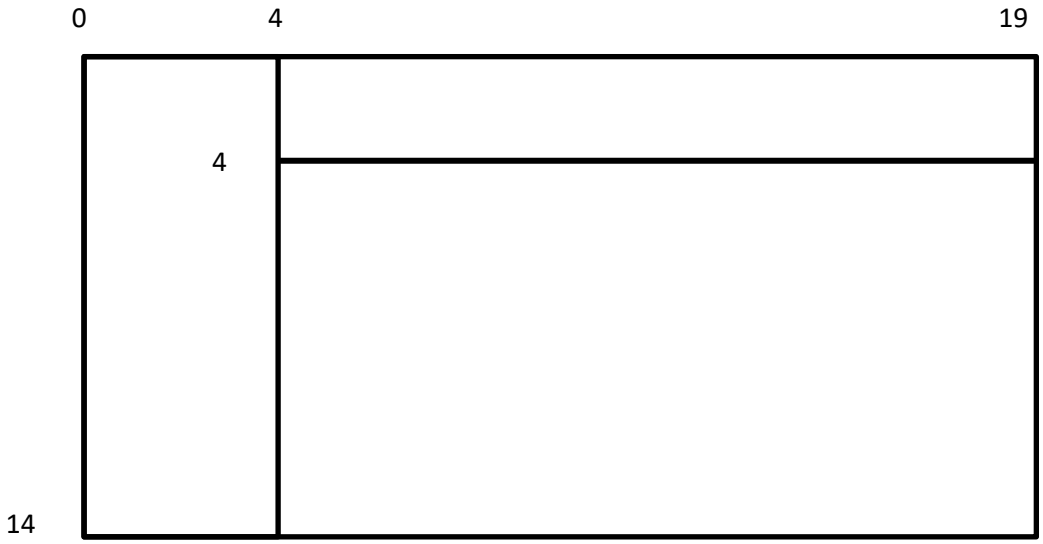
Do the following Exercises

Exercise 1:

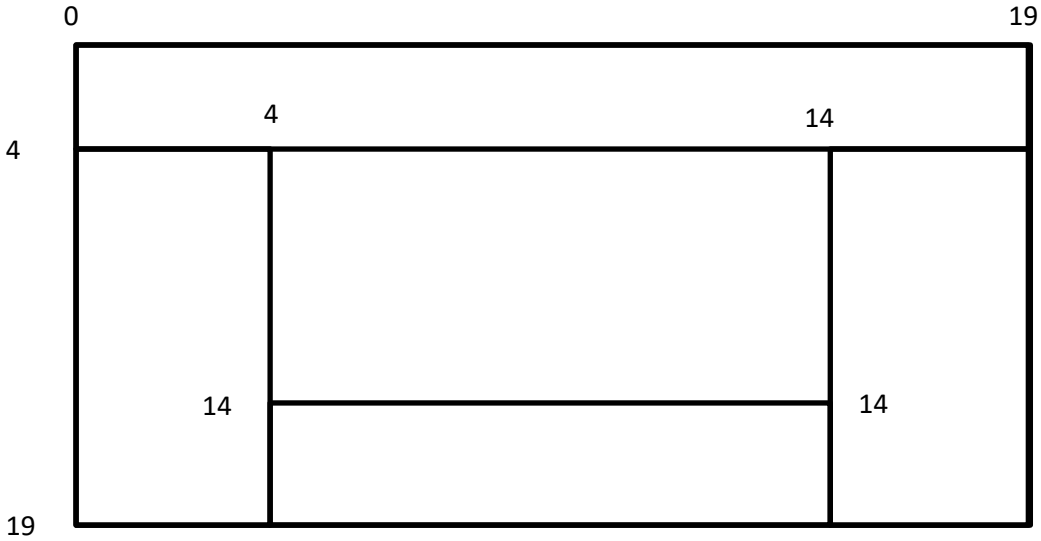
Output is:



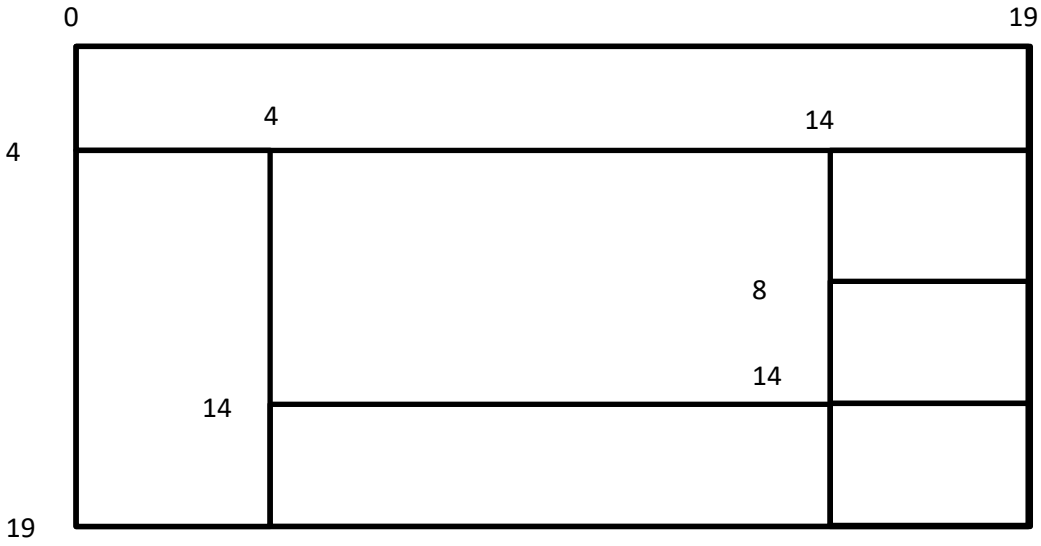
Exercise 2:



Exercise 3:



Exercise 4:



The while Loop

A while loop is a control structure that allows you to repeat a task a certain number of times.

Syntax:

The syntax of a while loop is:

```
while(Boolean_expression)
{
  //Statements
}
```

While Sample 1:

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

Int main(){
    int x = 10;

    while( x < 20 ) {
      cout<<"value of x : " << x <<"\n";
      x++;
    }
  }
```

The Output is:

value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14

```
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

The break Keyword:

The *break* keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

Syntax:

The syntax of a break is a single statement inside any loop:

```
break;
```

Example:

#include <iostream>

Using namespace std;

```
Int main(){

      int [] numbers = {10, 20, 30, 40, 50};
      int x=0;
      while(x<5) {
         if( numbers[x] == 30 ) {
              break;
         }
         Cout<< x<<"\n";
         X++;
      }
   }
}
```

The Output is:

```
10
20
```

The continue Keyword:

The *continue* keyword can be used in any of the loop control structures. It causes the loop to immediately jump to the next iteration of the loop.

- In a for loop, the continue keyword causes flow of control to immediately jump to the update statement.
- In a while loop or do/while loop, flow of control immediately jumps to the Boolean expression.

Syntax:

The syntax of a continue is a single statement inside any loop:

```
continue;
```

Example:

```
#include<iostream>
Using namespace std;
Int main(){

      int [] numbers = {10, 20, 30, 40, 50};
      int x=0;
      while(x<5){

         if( numbers[x] == 30 ) {
              continue;
         }
         Cout<<numbers[x]<<"\n";
         X++;
   }
}
```

The Output is:

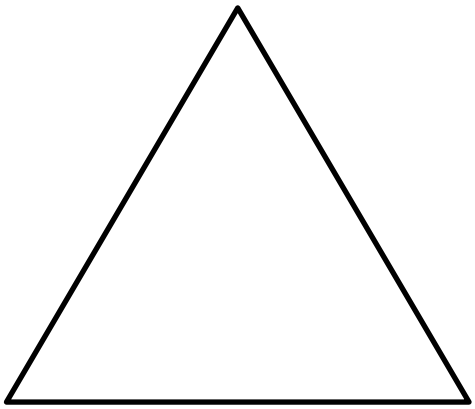
```
10
20
40
50
```

Do the following Exercise Using whilte Loop

Exercise 1

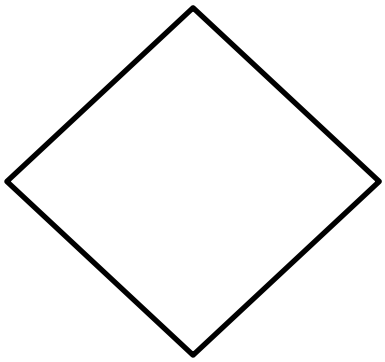
Triangle shape

15x15



Exericse 2

9x9 diamond shape



Exercise 3

20x20 Rectangle Shape

The String Class in C++:

The standard C++ library provides a string class type that supports all the operations mentioned above, additionally much more functionality. We will study this class in C++ Standard Library but for now let us check following example:

At this point, you may not understand this example because so far we have not discussed Classes and Objects. So can have a look and proceed until you have understanding on Object Oriented Concepts.

String concatenation example 1

```
#include <iostream>
#include <string>

using namespace std;

int main ()
{
  string str1 = "Hello";
  string str2 = "World";
  string str3;
  int  len ;

  // copy str1 into str3
  str3 = str1;
  cout << "str3 : " << str3 << endl;
```

```
  // concatenates str1 and str2
  str3 = str1 + str2;
  cout << "str1 + str2 : " << str3 << endl;

  // total lenghth of str3 after concatenation
  len = str3.size();
  cout << "str3.size() :  " << len << endl;

  return 0;
}
```

String Concatenation example 2

```
strcat example
#include <stdio.h>
#include <string.h>

int main ()
{
  char str[80];

  strcpy (str,"these ");
  strcat (str,"strings ");
  strcat (str,"are ");
  strcat (str,"concatenated.");

  puts (str);
  return 0;
}
```

Output is:

these strings are concatenated.

String tokenization example 1

```
/* strtok example */
#include <stdio.h>
#include <string.h>

int main ()
{
  char str[] ="This,a,sample, string.";
  char * pch;
  printf ("Splitting string \"%s\" into tokens:\n",str);
  pch = strtok (str," ,.-");
  while (pch != NULL)
  {
    printf ("%s\n",pch);
    pch = strtok (NULL,",");
  }
  return 0;
}
```

Output:

Splitting string "- This, a sample string." into tokens:
This
a
sample
string

String Tokenization Sample 2

```
#include<iostream>
#include<string>
```

```
using namespace std;

const int arr_size_max=1000;
Int main(void){
char  str[]="hello|how are you| how is life?|";
char *tokchar;

Int i=0;
string output[arr_size_max];

tokchar=strtok(str,"|");

while(tokchar){
output[i]=tokchar;
tokchar=strtok(NULL,"|");
I++;

}

int arr_element;
cout<<"what part of array would you like to see ?";
cin>>arr_element;
cout<<"\n\n"<<output[arr_element];
cin.sync();
cin.get();
return 0;
}
```

String Example 1

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

int main()
{
        String carr;

         cout<<"Enter String:";
         cin>>carr;
```

```
for(int i=0;i<carr.size();i++){
        switch(carr[i]){
                case 'a': carr[i]='A';
                        break;
                case 'b': carr[i]='B';
                        break;
                case 'c': carr[i]='C';
                        break;
                case 'd': carr[i]='D';
                        break;
                case 'e': carr[i]='E';
                        break;
                case 'f': carr[i]='F';
                        break;
                case 'g': carr[i]='G';
                        break;
                case 'h': carr[i]='H';
                        break;
                case 'i': carr[i]='I';
                        break;
                case 'j': carr[i]='J';
                        break;
                case 'k': carr[i]='K';
                        break;
                case 'l': carr[i]='L';
                        break;
                case 'm': carr[i]='M';
                        break;
                case 'n': carr[i]='N';
                        break;
                case 'o': carr[i]='O';
                        break;
                case 'p': carr[i]='P';
                        break;
                case 'q': carr[i]='Q';
                        break;
                case 'r': carr[i]='R';
                        break;
                case 's': carr[i]='S';
```

```
                                        break;
                                case 't': carr[i]='T';
                                        break;
                                case 'u': carr[i]='U';
                                        break;
                                case 'v': carr[i]='V';
                                        break;
                                case 'w': carr[i]='W';
                                        break;
                                case 'x': carr[i]='X';
                                        break;
                                case 'y': carr[i]='Y';
                                        break;
                                case 'z': carr[i]='Z';
                                        break;


                        }

                }
                for(int j=0;j<carr.length;j++){
                        cout<<"carr[j]";


                }

                }

}
```

The Output is:

Enter String: hello northern city

Output is : HELLO NORTHERN CITY

Do the following exercises

StringExer1

String input: hello how is nc

Output is: Hello How Is Nc

StringExer2

String input: hello how is Northern City

Output is: HelloHowIsNorthernCity

StringExer3

String input: hello how is life

Output is:

a: 0 b: 0 c: 0 d: 0 e: 2 f: 1 g: 0 h: 2 i: 2 j: 0 k:0

l:1 m: 0 n: 0 o:2 p: 0 q: 0 r: 0 s: 1 t: 0 u: 0 v:0

w: 1 x: 0 y: 0 z:0

StringExer4

String input: hello friend, how is life, I hope you are fine, God BLess

Output is: hello friend

How is life

I hope you are fine

God Bless

StringExer5

Enter name: maung maung

Enter age: 23

Enter address: Hle Dan, yangon

Enter Phone: 09411361

Enter Email: mgmg@gmail.com

Output is: mang maung | 23| Hle Dan,Yangon| 09411361| mgmg@gmail.com|

One Dimensional Array

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

int main(){
                int I;
                 int num[]={12,13,17,18,64};
                 for( i=0;i<5;i++){
                 cout<<"input"<<i<<"index is:<<+num[i]<<endl;

                 }
                 }

        }
```

Two Dimensional Array

```
#include<iostream>
using namespace std;
int main(){
        int num[][3]={
                        {1,5,8,},
                        {13,6,7}

                        };
                for( int i=0;i<2;i++){
                        for(int  j=0;j<3;j++){
                        cout<<"row[<<i<<"] col[“<<j<<"] is: "<<num[i][j]<<endl;
                        }
                        cout<<"\n";
                }
```

```
        }
    }
```

Three DimensionalArray

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

int main(){

        int num[][3][3]={
                {{1,5,2},
                {3,7,6},
                {14,9,8}

                },
                {{15,28,49},
                {33,17,29},
                {102,48,25}
                },
                {{114,64,71},
                 {32,22,84},
                 {26,12,58}

                }
          };
        for(for i=0;i<3;i++){
                for( for j=0;j<3;j++){
                        for(k=0;k<3;k++){

                                cout<<"inputs["<<i<<"]["<<j<<"]["<<k<<"]is:"<<num[i][j][k];


                        }
                        cout<<"\n";


                        }
                }
            }
        }
```

Array Sample Exercise 1

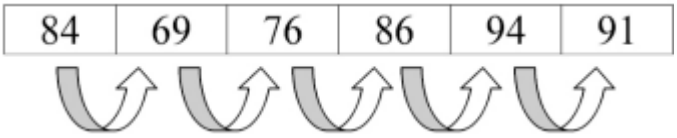
```
#include<iostream>

using namespace std;

int main(){

string  names[5];
int arr_index=0,choice=0;
bool loop=true;


while(loop){

                cout<<"          MENU            "<<endl;
                cout<<"----------------------------"<<endl;
                cout<<"        Choose one          "<<endl;
                cout<<"----------------------------"<<endl;
                cout<<"        1]  Add Name     "<<endl;
                cout<<"        2] View Name     "<<endl;
                cout<<"        3] Exit program   "<<endl;
                cout<<"----------------------------"<<endl;
                cout<<"  Enter Choice:" ;
                cin>>choice;
                switch(choice){

                        case 1: cout<<"Enter Name:";
                                cin>>name;
                                names[arr_index]=name;
                                arr_index++;
                                break;

                        case 2: cout<<"Name list:";
                                for(int j=0;j<arr_index;j++){
                                        cout<<names[j]<<endl;
                                  }
                                break;
                        case 3: cout<<"Goodbye!!"<<endl;
                                loop=false;
                                break;
                                        }

        }
```

```
    }

    }
```

Sorting Algorithm

Bubble Sort

In the **bubble sort,** as elements are sorted they gradually "bubble" (or rise) to their proper location in the array, like bubbles rising in a glass of soda. The bubble sort repeatedly compares **adjacent elements** of an array. The first and second elements are compared and swapped if out of order. Then the second and third elements are compared and swapped if out of order. This sorting process continues until the last two elements of the array are compared and swapped if out of order.



When this first pass through the array is complete, the bubble sort returns to elements one and two and starts the process all over again. So, when does it stop? **The bubble sort knows that it is finished when it examines the entire array and no "swaps" are needed (thus the list is in proper order).** The bubble sort keeps track of the occurring swaps by the use of a flag.

The table below follows an array of numbers before, during, and after a bubble sort for *descending* order. A "pass" is defined as one full trip through the array comparing and if necessary, swapping, **adjacent** elements. Several passes have to be made through the array before it is finally sorted.

Array at beginning:	84	69	76	86	94	91
After Pass #1:	84	76	86	94	91	69
After Pass #2:	84	86	94	91	76	69
After Pass #3:	86	94	91	84	76	69
After Pass #4:	94	91	86	84	76	69
After Pass #5 (done):	94	91	86	84	76	69

The bubble sort is an easy algorithm to program, but it is slower than many other sorts. With a

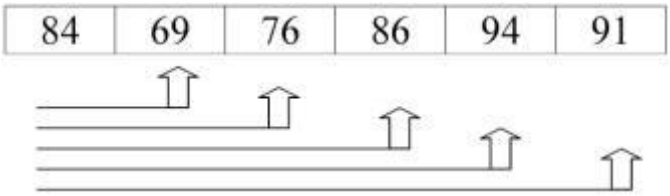
bubble sort, it is always necessary to make one final "pass" through the array to check to see that no swaps are made to ensure that the process is finished. In actuality, the process is finished before this last pass is made.

// Bubble Sort *Method* for Descending Order

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

using namespace std;

int main(){

     int num[]={12,44,7,33,78,31,68,124};

    int j,k;
    int flag = 1;   // set flag to true to begin first pass
    int temp;   //holding variable

    while ( flag )
    {
          flag= 0;    //set flag to false awaiting a possible swap
          for( j=0;  j <8 -1;  j++ )
          {
                if ( num[ j ] <num[j+1] )   // change to > for ascending sort
                {
                      temp = num[ j ];              //swap elements
                      num[ j ] = num[ j+1 ];
                      num[ j+1 ] = temp;
                     flag = 1;            //shows a swap occurred
               }
          }
     }

 cout<<"output is:";

   for(k=0;k<8;k++){

   cout<<num[k]<<"\n";

  }

}
```

Exchange Sort

The **exchange sort** is similar to its cousin, the bubble sort, in that it compares elements of the array and swaps those that are out of order. (Some people refer to the "exchange sort" as a "bubble sort".) The difference between these two sorts is the manner in which they compare the elements. **The exchange sort compares the first element with each following element of the array, making any necessary swaps.**



When the first pass through the array is complete, the exchange sort then takes the second element and compares it with each following element of the array swapping elements that are out of order. This sorting process continues until the entire array is ordered.

Let's examine our same table of elements again using an exchange sort for descending order. Remember, a "pass" is defined as one full trip through the array comparing and if necessary, swapping elements

Array at beginning:	84	69	76	86	94	91
After Pass #1:	94	69	76	84	86	91
After Pass #2:	94	91	69	76	84	86
After Pass #3:	94	91	86	69	76	84
After Pass #4:	94	91	86	84	69	76
After Pass #5 (done):	94	91	86	84	76	69

The exchange sort, in some situations, is slightly more efficient than the bubble sort. It is not necessary for the exchange sort to make that final complete pass needed by the bubble sort to determine that it is finished.

//Exchange Sort *Method* for Descending Order (integers)

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

using namespace std;

int main(){

    int num[]={12,44,7,33,78,31,68,124};
```

```
    int i, j, temp;  //be sure that the temp variable is the same "type" as the array
    for ( i = 0; i <num.length - 1; i ++ ){


        for ( j = i + 1; j <num.length; j ++ ) {
            if( num[ i ] <num[ j ] ){        //sorting into descending order

                    temp = num[ i ];   //swapping
                    num[ i ] = num[ j ];
                    num[ j ] = temp;
             }
        }
    }

cout<<"output is:";

  for(k=0;k<8;k++){

  cout<<num[k]<<"\n";

 }


}
```

Selection Sort

The **selection sort** is a combination of searching and sorting.

During each pass, the unsorted element with the smallest (or largest) value is moved to its proper position in the array.
The number of times the sort passes through the array is one less than the number of items in the array. In the selection sort, the inner loop finds the next smallest (or largest) value and the outer loop places that value into its proper location.

Let's look at our same table of elements using a selection sort for descending order. Remember, a "pass" is defined as one full trip through the array comparing and if necessary, swapping elements.

Array at beginning:	84	69	76	86	94	91
After Pass #1:	84	91	76	86	94	69
After Pass #2:	84	91	94	86	76	69
After Pass #3:	86	91	94	84	76	69
After Pass #4:	94	91	86	84	76	69
After Pass #5 (done):	94	91	86	84	76	69

While being an easy sort to program, the selection sort is one of the least efficient. The algorithm offers no way to end the sort early, even if it begins with an already sorted list.

// Selection Sort Method for Descending Order

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

using namespace std;

int main(){

    int num[]={12,44,7,33,78,31,68,124};

   int i, j, first, temp;
   for ( i = num.length - 1; i > 0; i - - ){

       first = 0;   //initialize to subscript of first element
```

```
        for(j = 1; j <= i; j ++){ //locate smallest element between positions 1 and i.

            if( num[ j ] <num[ first ] )
              first = j;
        }
        temp = num[ first ];   //swap smallest found with element in position i.
        num[ first ] = num[ i ];
        num[ i ] = temp;
     }

    cout<<"output is:";

   for(k=0;k<8;k++){

   cout<<num[k]<<"\n";

  }



}
```

Alphabetic Sorting

Sorting into alphabetic order can be accomplished with both the bubble and exchange processes. The only thing you must remember when sorting alphabetically is the way in which Java deals with comparing String values.

- Java provides two methods for comparing strings: compare()
- If s1 and s2 are String variables, then their values can be compared by s1.compare (s2).
- **Compare() returns an int which is 0 if the two strings are identical, positive if s1>s2, and negative if s1<s2.**
- compare() operates exactly as does compare() except that it handles mixed case strings as single case strings

The ASCII code chart assigns numerical values to the sequences 'a-z', and 'A-Z'. For example, the capital letter A is assigned a numerical value smaller than that for capital B. Thus alphabeticorder is actually ascending order.

Bubble Sort Sample:

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

using namespace std;

int main(){


        String names[] = {"joe", "slim", "ted", "george"};

       int j;
       int flag = 1;  // will determine when the sort is finished
       string temp;

       while ( flag ) {
           flag =0;
           for ( j = 0;  j <4 - 1;  j++ ) {
                 if (names [ j ].compare(names [ j+1 ] )>0) {
```

```
                    // ascending sort
                temp = names[ j ];
                names [ j ] = names [ j+1];     // swapping
                names [ j+1] = temp;
                flag = 1;
            }
        }
    }


    for ( int k = 0;  k < 4;  k++ ){
        cout<<names[k]<<"\n";
}

}
}
```

Exchange Sort Sample:

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

using namespace std;

Int main(){


    string names[] = {"joe", "slim", "ed", "george"};

    int i, j;
    string temp;



    for ( i = 0;  i <3;  i++ ) {
        for ( j = i + 1;  j <4;  j++ ) {
        if (names [ i ].compare(names [ j ])>0 ){

                    // ascending sort
                temp = names[ i ];
                names [ i ] = names [ j ];     // swapping
                names[ j ] = temp;

            }
        }
    }
```

```
for ( int k = 0;  k < 4;  k++ ){
            cout<<names[ k ]<<"\n";
    }


    }
}
```

Sorting Number Sample 1

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

using namespace std;

Int main(){
                                int temp=0;
                                int[] num={12,11,19,8,24};
                                for(int i=0;i<4;i++){
                                        for(int j=i+1;j<5;j++){
                                                if(num[i]>num[j]){

                                                        temp=num[i];
                                                        num[i]=num[j];
                                                        num[j]=temp;
                                                }
                                        }
                                }
                                for(int k=0;k<5;k++){
                                        cout<<num[k]<<"\n");
                                }
                        }
        }
```

Do the following Exercises

MENU

[1] ADD A NUMBER

[2] VIEW NUMBERS

[3] SORT NUMBERS

[4] DELETE NUMBERS

[5] EXIT PROGRAM

ENTER CHOICE:

MENU

[1] ADD NAME

[2] VIEW NAMES

[3] SORT NAMES

[4] DELETE NAME

[5]SEARCH NAME

[5] EXIT PROGRAM

ENTER CHOICE:

STUDENT RECORDS

[1] ADD RECORD (Id,Name,Address)

[2] VIEW RECORDS

[4] DELETE REOCRD

[5]SEARCH RECORD

[5] EXIT PROGRAM

ENTER CHOICE:

Function

A function definition consists of a function header and a function body. Here are all the parts of a function:

- **Modifiers:** The modifier, which is optional, tells the compiler how to call the function. This defines the access type of the method.
- **Return Type:** A function may return a value. The returnValueType is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the returnValueType is the keyword **void**.
- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a method may contain no parameters.
- **Function Body:** The function body contains a collection of statements that define what the function does.

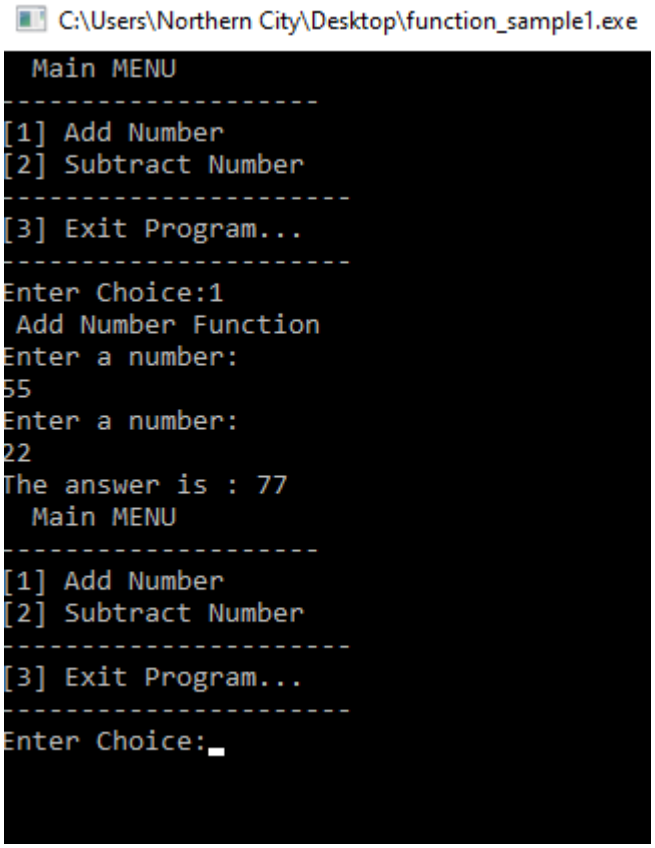
Function Example 1

```
#include<iostream>

using namespace std;


//header or directives for funtions
void main_menu();
int getChoice();
int getNum();
void display();
int cal_add(int x,int y);
int cal_sub(int x,int y);
void add();
void sub();



int main(){

	bool flag=true;
	while(flag){

	main_menu();
	switch(getChoice()){
		case 1:add();break;
		case 2:sub();break;
		case 3:cout<<"goodbye..."<<endl;flag=false;break;
	}
 }
}
```

```
void main_menu(){

        cout<<"  Main MENU    "<<endl;
        cout<<"--------------------"<<endl;
        cout<<"[1] Add Number   "<<endl;
        cout<<"[2] Subtract Number   "<<endl;
        cout<<"----------------------"<<endl;
        cout<<"[3] Exit Program..."<<endl;
        cout<<"----------------------"<<endl;

}

int getChoice(){
        int choice;
        cout<<"Enter Choice:";
        cin>>choice;
        return choice;
}

int getNum(){
        int num;
        cout<<"Enter a number:"<<endl;
        cin>>num;
        return num;
}

int cal_add(int x,int y){
        return x+y;
}

int cal_sub(int x,int y){
        return x-y;
}

void display(int ans){

        cout<<"The answer is : "<<ans<<endl;

}
void add(){
        cout<<" Add Number Function  "<<endl;
```

```
        int x=getNum();
        int y=getNum();
        int ans=cal_add(x,y);
        display(ans);


}


void sub(){
        cout<<" Subtract Number Function  "<<endl;

        int x=getNum();
        int y=getNum();
        int ans=cal_sub(x,y);
        display(ans);


}
```

Output:


C:\Users\Northern City\Desktop\function_sample1.exe
Main MENU
[1] Add Number
[2] Subtract Number
[3] Exit Program...
Enter Choice:1
Add Number Function
Enter a number:
55
Enter a number:
22
The answer is : 77
Main MENU
[1] Add Number
[2] Subtract Number
[3] Exit Program...
Enter Choice:

Do the following Exercises using function

Main Menu

(My Mathematic Program)

[1] Circle

[2] Triangle

[3] Rectangle

Main choice==1	Main choice==2	Main choice==3
Circle Menu [1] Area of Circle [2] Circumference of Circle Enter Choice:	Triangle Menu [1] Area of Triangle [2] Volume of Triangle Enter Choice:	Triangle Menu [1] Area of Rectangle [2] Volume of Rectangle Enter Choice:

MENU

[1] ADD A NUMBER

[2] VIEW NUMBERS

[3] SORT NUMBERS

[4] DELETE NUMBERS

[5] EXIT PROGRAM

ENTER CHOICE:

STUDENT RECORDS

[1] ADD RECORD

[2] VIEW RECORDS

[4] DELETE REOCRD

[5]SEARCH RECORD

[5] EXIT PROGRAM

ENTER CHOICE:

Data structures

A *data structure* is a group of data elements grouped together under one name. These data elements, known as *members*, can have different types and different lengths. Data structures can be declared in C++ using the following syntax:

```
struct type_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} object_names;
```

Where `type_name` is a name for the structure type, `object_name` can be a set of valid identifiers for objects that have the type of this structure. Within braces `{}`, there is a list with the data members, each one is specified with a type and a valid identifier as its name.

Sample 1

```
// example about structures
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

struct movies_t {
  string title;
  int year;
} mine, yours;


void printmovie (movies_t movie);

int main ()
{
  string mystr;

  mine.title = "2001 A Space Odyssey";
```

```
  mine.year = 1968;

  cout << "Enter title: ";
  getline (cin,yours.title);
  cout << "Enter year: ";
  getline (cin,mystr);
  stringstream(mystr) >> yours.year;

  cout << "My favorite movie is:\n ";
  printmovie (mine);
  cout << "And yours is:\n ";
  printmovie (yours);
  return 0;
}

void printmovie (movies_t movie)
{
  cout << movie.title;
  cout << " (" << movie.year << ")\n";
}
```

Output is:

```
Enter title: Alien
Enter year: 1979

My favorite movie is:
2001 A Space Odyssey (1968)
And yours is:
Alien (1979)
```

Sample 2

```
#include <iostream>
#include <cstring>

using namespace std;
void printBook( struct Books book );

struct Books
{
  char  title[50];
  char  author[50];
  char  subject[100];
  int   book_id;
};
```

```
int main( )
{
   struct Books Book1;        // Declare Book1 of type Book
   struct Books Book2;        // Declare Book2 of type Book

   // book 1 specification
   strcpy( Book1.title, "Learn C++ Programming");
   strcpy( Book1.author, "Chand Miyan");
   strcpy( Book1.subject, "C++ Programming");
   Book1.book_id = 6495407;

   // book 2 specification
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Yakit Singha");
   strcpy( Book2.subject, "Telecom");
   Book2.book_id = 6495700;

   // Print Book1 info
   printBook( Book1 );

   // Print Book2 info
   printBook( Book2 );

   return 0;
}
void printBook( struct Books book )
{
   cout << "Book title : " << book.title <<endl;
   cout << "Book author : " << book.author <<endl;
   cout << "Book subject : " << book.subject <<endl;
   cout << "Book id : " << book.book_id <<endl;
}
```

Output is:

```
Book title : Learn C++ Programming
Book author : Chand Miyan
Book subject : C++ Programming
Book id : 6495407
Book title : Telecom Billing
Book author : Yakit Singha
Book subject : Telecom
Book id : 6495700
```

Array of Structures

structures are types, they can also be used as the type of arrays to construct tables or databases of them:

Sample 2

```
// array of structures

#include <iostream>
#include <string>
#include <sstream>
using namespace std;

struct movies_t {
  string title;
  int year;
} films [3];


void printmovie (movies_t movie);

int main ()
{
  string mystr;
  int n;

  for (n=0; n<3; n++)
  {
    cout << "Enter title: ";
    getline (cin,films[n].title);
    cout << "Enter year: ";
    getline (cin,mystr);
    stringstream(mystr) >> films[n].year;
  }

  cout << "\nYou have entered these movies:\n";
  for (n=0; n<3; n++)
    printmovie (films[n]);
  return 0;
}

void printmovie (movies_t movie)
{
```

```
 cout << movie.title;
 cout << " (" << movie.year << ")\n";
}
```

Output is:
Enter title: Blade Runner
Enter year: 1982
Enter title: The Matrix
Enter year: 1999
Enter title: Taxi Driver
Enter year: 1976

You have entered these movies:
Blade Runner (1982)
The Matrix (1999)
Taxi Driver (1976)

My E-Library

1] Add New Book

2] View Books

3] Search Book

4] Delete Book

5] Exit program

Enter Choice:

Specifications:

1) Book id
2) Title
3) Price

Structure Exercise

Main MENU

Movie Disc Rental Service

1] Add New Disk Record

2] Show All Disks

3] Search Disk By ID

4] Delete Record By ID

5] Update Price

6] Exit program

Enter Choice:

Specifications:

1) Disc_ID
2) Title
3) Price
4) Type

CLASSES

Classes are an expanded concept of *data structures*: like data structures, they can contain data members, but they can also contain functions as members.

An *object* is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable.

Classes are defined using either keyword `class` or keyword `struct`, with the following syntax:

```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
```

```
} object_names;
```

CLASS EXAMPLE 1

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

class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area() {return width*height;}
};

void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}

int main () {
  Rectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```

CLASS EXAMPLE 2

```
// example: one class, two objects
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area () {return width*height;}
};

//constructor
void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
```

```
}

int main () {
  Rectangle recta, rectb;
  rect.set_values (3,4);
  rectb.set_values (5,6);
  cout << "rect area: " << recta.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

CLASS EXAMPLE 3

```
// example: class constructor
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
  public:
    Rectangle (int,int);
    int area () {return (width*height);}
};

Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  Rectangle rect (3,4);
  Rectangle rectb (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

CLASS EXAMPLE 4

```
// overloading class constructors
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
  public:
    Rectangle ();
    Rectangle (int,int);
    int area (void) {return (width*height);}
};

Rectangle::Rectangle () {
  width = 5;
  height = 5;
}

Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  Rectangle rect (3,4);
  Rectangle rectb;
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

CLASS EXAMPLE5

```
// member initialization
#include <iostream>
using namespace std;

class Circle {

        double radius;
```

```
        public:
        Circle(double r) : radius(r) { }

        double area() {

        return radius*radius*3.14159265;

        }
};

class Cylinder {
   Circle base;
   double height;
 public:
   Cylinder(double r, double h) : base (r), height(h) {}
   double volume() {

        return base.area() * height;
        }
};

int main () {
  Cylinder foo (10,20);

  cout << "foo's volume: " << foo.volume() << '\n';
  return 0;
}
```

CLASS EXAMPLE 4

```
#include <iostream>

using namespace std;

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

int main( )
{
  Box Box1;       // Declare Box1 of type Box
```

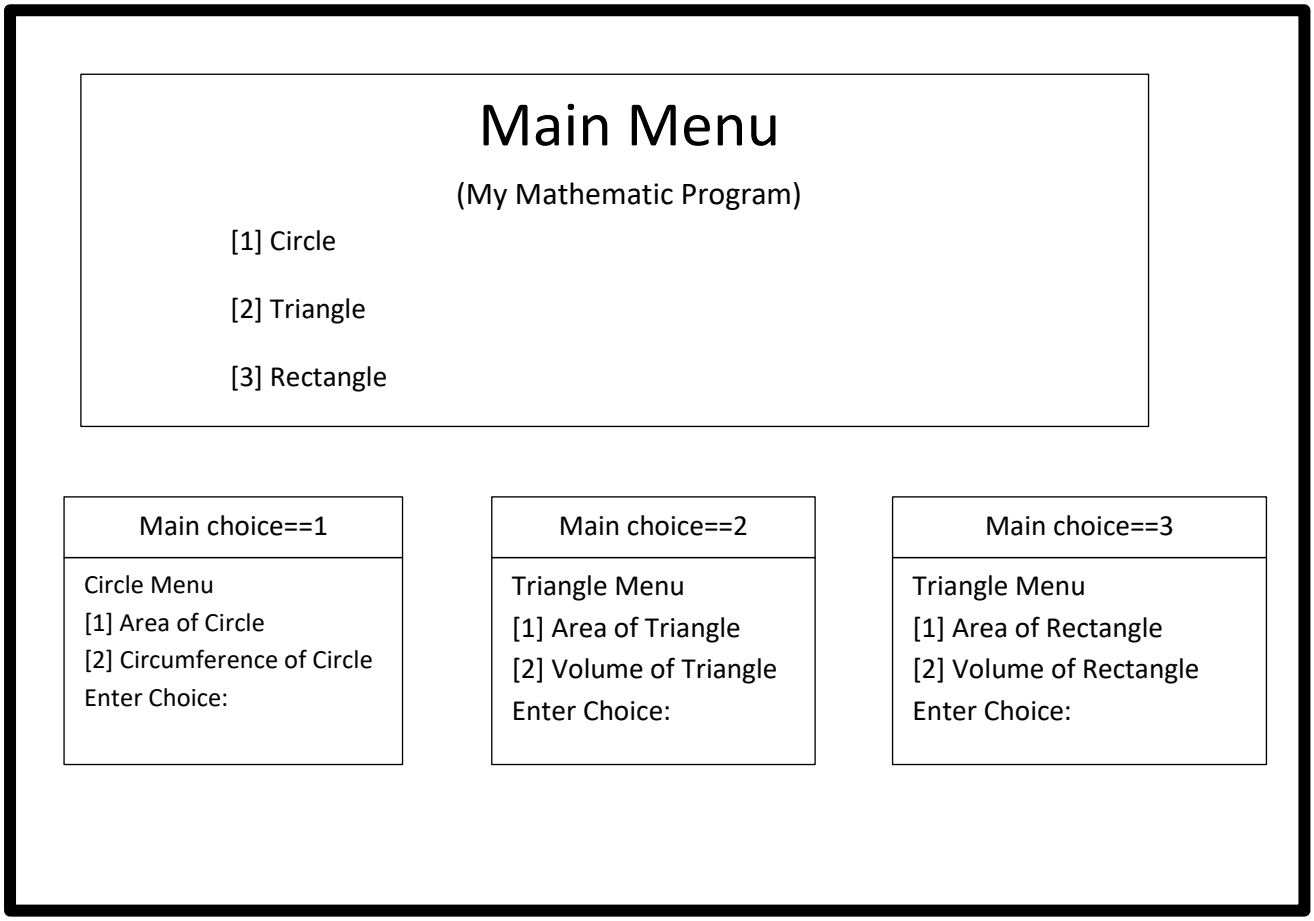
```
  Box Box2;       // Declare Box2 of type Box
  double volume = 0.0;     // Store the volume of a box here

  // box 1 specification
  Box1.height = 5.0;
  Box1.length = 6.0;
  Box1.breadth = 7.0;

  // box 2 specification
  Box2.height = 10.0;
  Box2.length = 12.0;
  Box2.breadth = 13.0;
  // volume of box 1
  volume = Box1.height * Box1.length * Box1.breadth;
  cout << "Volume of Box1 : " << volume <<endl;

  // volume of box 2
  volume = Box2.height * Box2.length * Box2.breadth;
  cout << "Volume of Box2 : " << volume <<endl;
  return 0;
}
```

C++ class array sample

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

class MyClass {
 int x;
public:
 void setX(int i) { x = i; }
 int getX() { return x; }
};

int main()
{
 MyClass obs[4];
 int i;

 for(i=0; i < 4; i++)
  obs[i].setX(i);

 for(i=0; i < 4; i++)
  cout << "obs[" << i << "].getX(): " << obs[i].getX() << "\n";
```

```
  return 0;
}
```

Do the Following Exercises



YOMA BANK

1] Open New Account
2] View Account
3] Deposit
4] Withdraw
5] Update Account
6] Close Account
Enter Choice:

Specifications:
1. Account no
2. Account name
3. Date
4. Branch Name
5. Initial Amount
6. NRC
7. Address
8. Phone

YUZANA CAR SALE SERVICE

1] Add New Car
2] View Car
3] Sell Car
5] Update Car Price
6] Delete Record
Enter Choice:

Specifications:
1. Car_ID
2. Brand
3. Price
4.Sale_Status

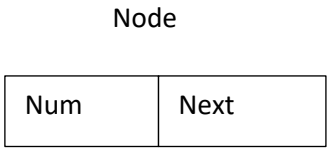
Linked Lists

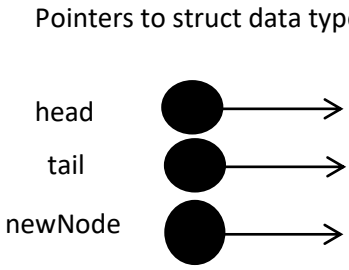
A **linked list** is a basic data structure where each item contains the information that we need to get to the next item.

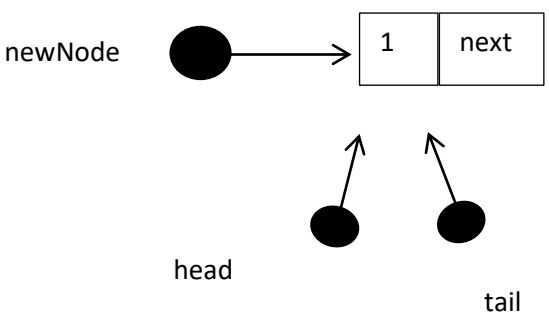
The main advantage of linked lists over arrays is that the links provide us with the capability to rearrange the item efficiently. This flexibility is gained at the expense of quick access to any arbitrary item in the list, because the only way to access to an item in the list is to follow links from the beginning.

Example 1

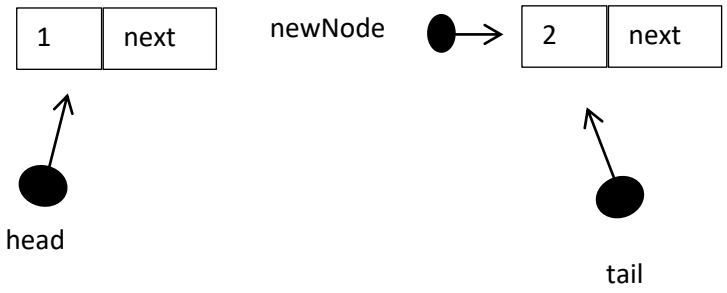
```
#include<iostream>
#include<string.h>

using namespace std;

struct Node{
        int num;
        Node *next;

};


struct Node *head=NULL;
struct Node *tail=NULL;
struct Node *newNode=NULL;

int main(){


newNode=new Node;
newNode->num=1;
head=newNode;
tail=newNode;
```



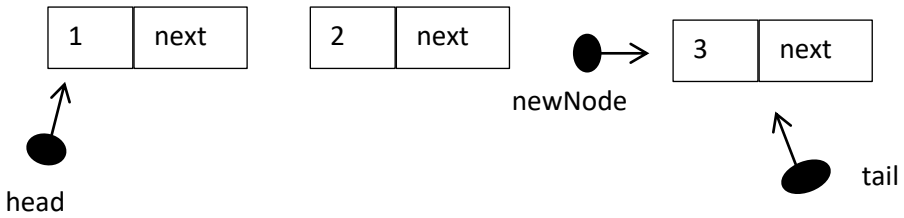




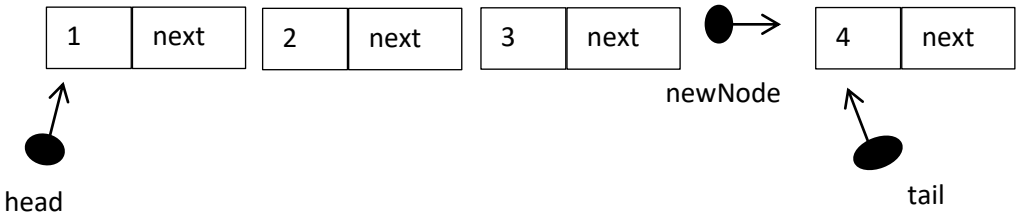
```
newNode=new Node;
newNode->num=2;
tail->next=newNode;
tail=tail->next;
```



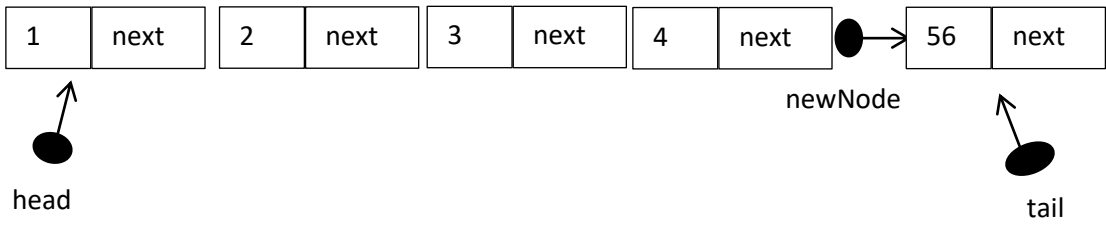

```
newNode=new Node;
newNode->num=3;
tail->next=newNode;
tail=tail->next;
```




```
newNode=new Node;
newNode->num=4;
tail->next=newNode;
tail=tail->next;
```




```
newNode=new Node;
newNode->num=56;
tail->next=newNode;
```




```
newNode->next=NULL;


while(head){

        cout<<head->num<<"   ";
        head=head->next;


}


        return 0;
}
```

Example 2

```
#include<iostream>
using namespace std;
struct node{
	int data;
	node *next;
};

struct node *head=NULL;

void addNode(int number);
void showlist(struct node *head);
void removeNode(int number);
void menu();
int getChoice();
int getNum();


void addNode(int number){

	node *temp=new node;
	temp->data=number;
	temp->next=NULL;

	if(head==NULL){

		head=temp;
		return;
	}

	node *curr=head;

	while(curr){

		if(curr->next==NULL){

			curr->next=temp;
			return;

		}

		curr=curr->next;
```

```
	}

}

struct node *searchNode(int number){

	if(head==NULL)  return head;

	node *curr=head;
	node *prev=head;

	while(curr){
			if(curr->data==number) return curr;

			prev=curr;
			curr=curr->next;

	}

}

void removeNode(int number){

	node *ptr=searchNode(number);

	if(ptr==NULL) cout<<"\nCould not delete ..";

	if(ptr==head){

		head=ptr->next;
		return;
	}

	node *curr=head;
	node *prev=head;
	while(curr){
			if(curr==ptr){
				prev->next=ptr->next;
				return;
			}
			prev=curr;
			curr=curr->next;

	}

}


void showlist(struct node *head){

	node *list=head;
```

```
	while(list){
		cout<<list->data<<"  ";
		list=list->next;
	}
	cout<<"\n"<<endl;

}

void menu(){

	cout<<"         M E N U          \n";
	cout<<" 1] add Number             \n";
	cout<<" 2] remove number           \n";
	cout<<" 3] show list of number              \n";
	cout<<" 4] exit program           \n";
}

int getChoice(){

	int choice;
	cout<<"Enter Choice:";
	cin>>choice;

	return choice;
}

int getNum(){

	int number;
	cout<<"Enter a Number:";
	cin>>number;

	return number;

}

int main(){

	int choice;
	int number;

	for(;;){

		menu();
		choice=getChoice();

		switch(choice){

			case 1:  number=getNum();
				 addNode(number);
```

```
                    break;
                case 2:   number=getNum();
                    removeNode(number);
                    break;

                case 3: showlist(head);
                    break;

                 case 4: cout<<"thanks for using my program";
                    return 1;
                default: cout<<"please press 4 to exit!!!!";


            }

        }

        return 0;

}
```

Example 3

```
#include<iostream>
#include<string.h>
using namespace std;

struct Node{
        string name;
        string age;
        string address;
        struct Node *next;
};

struct Node *head=NULL;
struct Node *tail=NULL;
struct Node *newNode=NULL;

void menu(){
        cout<<"       M E N U       \n";
        cout<<"  1] Add Record       \n";
        cout<<"  2] View Record      \n";
        cout<<"  3] Search Record    \n";
        cout<<"  4] Delete Record    \n";
```

```
        cout<<"  5] Exit Program       \n";


}


int getChoice(){
        int choice;
        cout<<"Enter Choice:";
        cin>>choice;
        return choice;
}

string getName(){
        string name;
        cout<<"Enter Name:";
        cin>>name;
        return name;
}

string getAge(){
        string age;
        cout<<"Enter Age:";
        cin>>age;
        return age;


}


string getAddress(){
        string address;
        cout<<"Enter Address:";
        cin>>address;
        return address;


}

void addRecord(string name,string age,string address){

        newNode=new Node;
        newNode->name=name;
        newNode->age=age;
        newNode->address=address;
        newNode->next=NULL;

        if(head==NULL){
```

```
                head=newNode;
                return;
        }

        Node *curr=head;
        while(curr){

                if(curr->next==NULL){

                        curr->next=newNode;
                        return;
                }

                curr=curr->next;
        }



}

void viewRecord(struct Node *head){

        Node *curr=head;

        if(curr==NULL){
                 cout<<"it is empty no record";

        }

        while(curr){

                cout<<"Name :"<<curr->name<<"\n";
                cout<<"Age  :"<<curr->age<<"\n";
                cout<<"Address  :"<<curr->address<<"\n";

                curr=curr->next;

        }


}

struct Node *searchNode(string name){

        if(head==NULL) return head;

        Node *curr=head;

        while(curr){

                if(curr->name==name){
```

```
                return curr;
                }

                curr=curr->next;

        }
}

void searchRecord(){

        string name=getName();

        Node *curr=searchNode(name);

        if(curr==NULL){

                cout<<"Sorry Not Found!!";
        }
        else{
                cout<<"-------------------------\n";
                cout<<"Record Found";
                cout<<"Name :"<<curr->name<<"\n";
                cout<<" Age :"<<curr->age<<"\n";
                cout<<"Address:"<<curr->address<<"\n";
                cout<<"-------------------------\n";

        }

}


void removeRecord(){

        string sname=getName();
        Node *sNode=searchNode(sname);

        if(sNode==NULL){
                cout<<"it is empty";
        }

        if(sNode==head){
                head=sNode->next;
                return;
        }

        Node *curr=head;
        Node *prev=head;

        while(curr){

                if(curr==sNode){
```

```
                        prev->next=sNode->next;
                        return;
                }
                prev=curr;
                curr=curr->next;
        }


}
int main(){
        int choice;
        string name="";
        string age="";
        string address="";

        for(;;){

                menu();
                choice=getChoice();
                switch(choice){

                        case 1: name=getName();
                                        age=getName();
                                        address=getAddress();
                                        addRecord(name,age,address);
                                        break;
                        case 2: viewRecord(head);
                                        break;

                        case 3: searchRecord();

                                break;
                        case 4: removeRecord();
                                break;


                }


        }

}
```

Do the following Exercises

Exercise 1

Northern City Computer Training Center
Myitkyina, Myanmar

1] Add new Student

2] View Students

3] Search Student

4] Delete Student

5]exit program

Enter Choice:_

Student record:

1. StudentID
2. Age
3. Gender
4. Phone

Exercise 2

City Express

Hle Dan, Yangon, Myanmar

1] Add new product
2] Sell Items
2] View Product
3] Search Product
4] Delete Product
5] Update Price
6] Exit program

Enter Choice:_

Product record:

1. ID
2. Name
3. Price
4. Items
5. Available

files

C++ provides the following classes to perform output and input of characters to/from files:

- **ofstream**: Stream class to write on files
- **ifstream**: Stream class to read from files
- **fstream**: Stream class to both read and write from/to files.

Open a file

The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to *open a file*. An open file is represented within a program by a *stream* (i.e., an object of one of these classes; in the previous example, this was `myfile`) and any input or output operation performed on this stream object will be applied to the physical file associated to it.

In order to open a file with a stream object we use its member function `open`:

```
open (filename, mode);
```

Where `filename` is a string representing the name of the file to be opened, and `mode` is an optional parameter with a combination of the following flags:

`ios::in`	Open for input operations.
`ios::out`	Open for output operations.
`ios::binary`	Open in binary mode.
`ios::ate`	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
`ios::app`	All output operations are performed at the end of the file, appending the content to the current content of the file.
`ios::trunc`	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

All these flags can be combined using the bitwise operator OR (`|`). For example, if we want to open the file `example.bin` in binary mode to add data we could do it by the following call to member function `open`:

```
1 ofstream myfile;
2 myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

Each of the `open` member functions of classes `ofstream`, `ifstream` and `fstream` has a default mode that is used if the file is opened without a second argument:

class	default mode parameter
`ofstream`	ios::out
`ifstream`	ios::in
`fstream`	ios::in | ios::out

For `ifstream` and `ofstream` classes, `ios::in` and `ios::out` are automatically and respectively assumed, even if a mode that does not include them is passed as second argument to the `open` member function (the flags are combined).

For `fstream`, the default value is only applied if the function is called without specifying any value for the mode parameter. If the function is called with any value in that parameter the default mode is overridden, not combined.

File streams opened in *binary mode* perform input and output operations independently of any format considerations. Non-binary files are known as *text files*, and some translations may occur due to formatting of some special characters (like newline and carriage return characters).

Since the first task that is performed on a file stream is generally to open a file, these three classes include a constructor that automatically calls the `open` member function and has the exact same parameters as this member. Therefore, we could also have declared the previous `myfile` object and conduct the same opening operation in our previous example by writing:

```
ofstream myfile ("example.bin", ios::out | ios::app | ios::binary);
```

Combining object construction and stream opening in a single statement. Both forms to open a file are valid and equivalent.

To check if a file stream was successful opening a file, you can do it by calling to member `is_open`. This member function returns a `bool` value of `true` in the case that indeed the stream object is associated with an open file, or `false` otherwise:

```
if (myfile.is_open()) { /* ok, proceed with output */ }
```

Closing a file

When we are finished with our input and output operations on a file we shall close it so that the operating system is notified and its resources become available again. For that, we call the stream's member function `close`. This member function takes flushes the associated buffers and closes the file:

```
myfile.close();
```

Once this member function is called, the stream object can be re-used to open another file, and the file is available again to be opened by other processes.

In case that an object is destroyed while still associated with an open file, the destructor automatically calls the member function `close`.

Text files

Text file streams are those where the `ios::binary` flag is not included in their opening mode. These files are designed to store text and thus all values that are input or output from/to them can suffer some formatting transformations, which do not necessarily correspond to their literal binary value.

Writing operations on text files are performed in the same way we operated with `cout`:

Sample 1:

```
1  // writing on a text file
2  #include <iostream>
3  #include <fstream>
4  using namespace std;
5
6  int main () {
7    ofstream myfile ("example.txt");
8    if (myfile.is_open())
9    {
10     myfile << "This is a line.\n";
11     myfile << "This is another line.\n";
12     myfile.close();
13   }
14   else cout << "Unable to open file";
15   return 0;
16 }
```

```
[file example.txt]
This is a line.
This is another line.
```

Reading from a file can also be performed in the same way that we did with cin:

Sample 2:

```
1  // reading a text file
2  #include <iostream>
3  #include <fstream>
4  #include <string>
5  using namespace std;
6
7  int main () {
8    string line;
9    ifstream myfile ("example.txt");
10   if (myfile.is_open())
11   {
12     while ( getline (myfile,line) )
13     {
14       cout << line << '\n';
15     }
16     myfile.close();
17   }
```

```
This is a line.
This is another line.
```

```
18
19  else cout << "Unable to open file";
20
21  return 0;
22 }
```

This last example reads a text file and prints out its content on the screen. We have created a while loop that reads the file line by line, using `getline`. The value returned by `getline` is a reference to the stream object itself, which when evaluated as a boolean expression (as in this while-loop) is `true` if the stream is ready for more operations, and `false` if either the end of the file has been reached or if some other error occurred.

Sample 3(Writing data to File):

```
#include <iostream>
#include <fstream>
#include<string.h>
using namespace std;

int main () {



 ofstream myfile ("example.txt");

 string name;
 string age;


 cout<<"Enter Name:";
 getline(cin,name);
 cout<<"Enter age:";
 getline(cin,age);

 if (myfile.is_open())
 {
  myfile << name<<"\n";
  myfile << age<<"\n";
  myfile.close();
 }
 else cout << "Unable to open file";
 return 0;
}
```

Sample 4:

#include <iostream>

```
#include <fstream>
#include<string.h>
#include<sstream>
#include<stdio.h>


using namespace std;

void addStudent() {



 ofstream myfile("student.txt",ios::app);

 string dummy;
 string name;
 string age;
 string address;

 getline(cin,dummy);

 cout<<"Enter Name:";
 getline(cin,name);

 cout<<"Enter age:";
 getline(cin,age);

 cout<<"Enter address:";
 getline(cin,address);


 if (myfile.is_open())
 {
  myfile << name<<"\n";
  myfile << age<<"\n";
  myfile << address<<"\n";


  myfile.close();
 }
 else {

        cout << "Unable to open file";

        }



}

void viewStudent(){
```

```
 string line;
 ifstream myfile("student.txt");
 if (myfile.is_open())
 {
  while ( getline (myfile,line) )
  {
    cout << line << '\n';
  }

  myfile.close();

 }

 else {

                cout << "Unable to open file";

        }


}

int getChoice(){

        int choice;
        cout<<"Enter Choice:";
        cin>>choice;
        return choice;
}

void main_menu(){

        cout<<"     Student Record      \n";
        cout<<" 1] add new Student      \n";
        cout<<" 2] view student records  \n";
        cout<<" 3] exit program         \n";


}

int main(){

        int choice;

        for(;;){

        main_menu();
        choice=getChoice();

        switch(choice){
```

```
                case 1: addStudent();
                        break;
                case 2: viewStudent();
                        break;
                case 3: cout<<" Goodbye ";
                        return 0;
                        break;
        }
}

}
```

Do the following Exercises using files

MENU

[1] ADD A NUMBER

[2] VIEW NUMBERS

[3] SORT NUMBERS

[4] DELETE NUMBERS

[5] EXIT PROGRAM

ENTER CHOICE:

MENU

[1] ADD NAME

[2] VIEW NAMES

[3] SORT NAMES

[4] DELETE NAME

[5] EXIT PROGRAM

ENTER CHOICE:

Northern City Computer Center

[1] REGISTER (**INPUTS: ID, NAME, COURSE,GENDER**)

[2] VIEW ALL STUDENTS

[4] DELETE STUDENT

[5]SEARCH STUDENT By ID

[6] UPDATE (ADDRESS, PHONE, EMAIL)

[7] EXIT PROGRAM

ENTER CHOICE:

YOMA BANK

(WELCOME TO YAMA BANK)

[1] ACCOUNT

[2] UPDATE

[3]SEARCH

[4] EXIT PROGRAM

ENTER CHOICE:

VIEW ACCOUNT SUBMENU	UPDATE ACCOUNT SUBMENU
[1] Open New Account* [2] Close Account ENTER CHOICE:	[1] DEPOSIT [2] WIDTHDRAW [3] PERSONAL INFORMATION UPDATE** ENTER CHOICE:

*(ACCCOUNT NO, ACCOUNT NAME, NRC, GENGER, PHONE, EMAIL, ADDRESS, DATE, INITIAL AMOUNT)

**(PHONE, EMAIL, ADDRESS)

C++ Programming Language

Object-Oriented Programming (OOP) in C++

1. Why OOP?

Suppose that you want to assemble your own PC, you go to a hardware store and pick up a motherboard, a processor, some RAMs, a hard disk, a casing, a power supply, and put them together. You turn on the power, and the PC runs. You need not worry whether the motherboard is a 4-layer or 6-layer board, whether the hard disk has 4 or 6 plates; 3 inches or 5 inches in diameter, whether the RAM is made in Japan or Korea, and so on. You simply put the hardware *components* together and expect the machine to run. Of course, you have to make sure that you have the correct *interfaces*, i.e., you pick an IDE hard disk rather than a SCSI hard disk, if your motherboard supports only IDE; you have to select RAMs with the correct speed rating, and so on. Nevertheless, it is not difficult to set up a machine from hardware *components*.

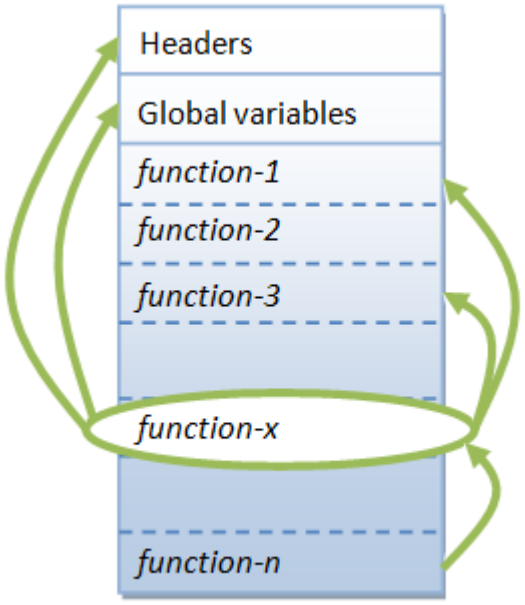
Similarly, a car is assembled from parts and components, such as chassis, doors, engine, wheels, brake, and transmission. The components are reusable, e.g., a wheel can be used in many cars (of the same specifications).

Hardware, such as computers and cars, are assembled from parts, which are reusable components.

How about software? Can you "assemble" a software application by picking a routine here, a routine there, and expect the program to run? The answer is obviously no! Unlike hardware, it is very difficult to "assemble" an application from *software components*. Since the advent of computer 60 years ago, we have written tons and tons of programs. However, for each new application, we have to re-invent the wheels and write the program from scratch.

Why re-invent the wheels?

1.1 Traditional Procedural-Oriented languages



A function (in C) is not well-encapsulated

Can we do this in traditional procedural-oriented programming language such as C, Fortran, Cobol, or Pascal?

Traditional procedural-oriented languages (such as C and Pascal) suffer some notable drawbacks in creating reusable software components:

1. The programs are made up of functions. Functions are often not *reusable*. It is very difficult to copy a function from one program and reuse in another program because the the function is likely to reference the headers, global variables and other functions. In other words, functions are not well-encapsulated as a self-contained *reusable unit*.
2. The procedural languages are not suitable of *high-level abstraction* for solving real life problems. For example, C programs uses constructs such as if-else, for-loop, array, function, pointer, which are low-level and hard to abstract real problems such as a Customer Relationship Management (CRM) system or a computer soccer game. (Imagine using assembly codes, which is a very low level code, to write a computer soccer game. C is better but no much better.)

In brief, the traditional procedural-languages *separate* the data structures and algorithms of the software entities.

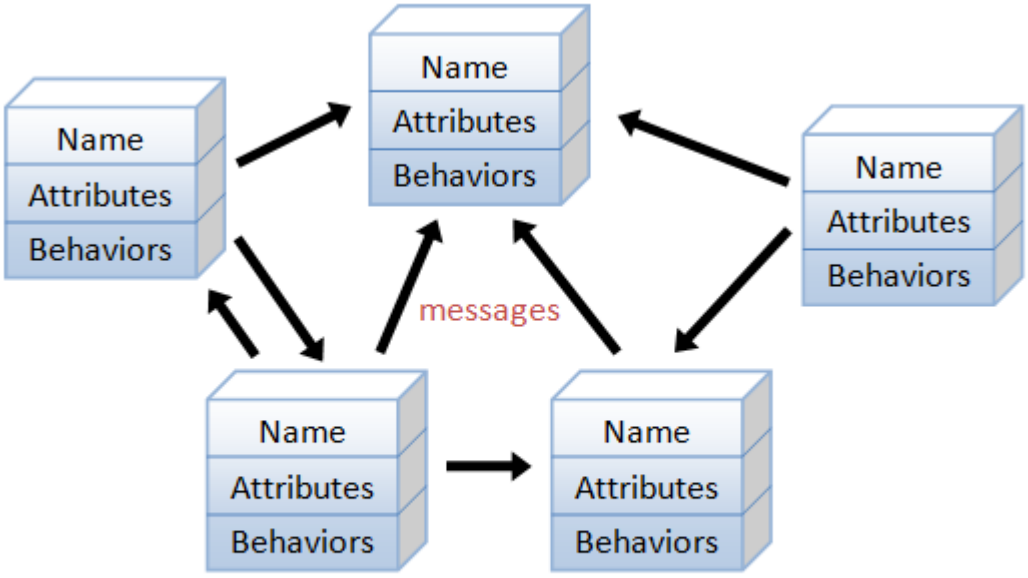
In the early 1970s, the US Department of Defense (DoD) commissioned a task force to investigate why its IT budget always went out of control; but without much to show for. The findings are:

1. 80% of the budget went to the software (while the remaining 20% to the hardware).
2. More than 80% of the software budget went to maintenance (only the remaining 20% for new software development).
3. Hardware components could be applied to various products, and their integrity normally did not affect other products. (Hardware can share and reuse! Hardware faults are isolated!)

4. Software procedures were often non-sharable and not reusable. Software faults could affect other programs running in computers.

The task force proposed to make software behave like hardware OBJECT. Subsequently, DoD replaces over 450 computer languages, which were then used to build DoD systems, with an object-oriented language called Ada.

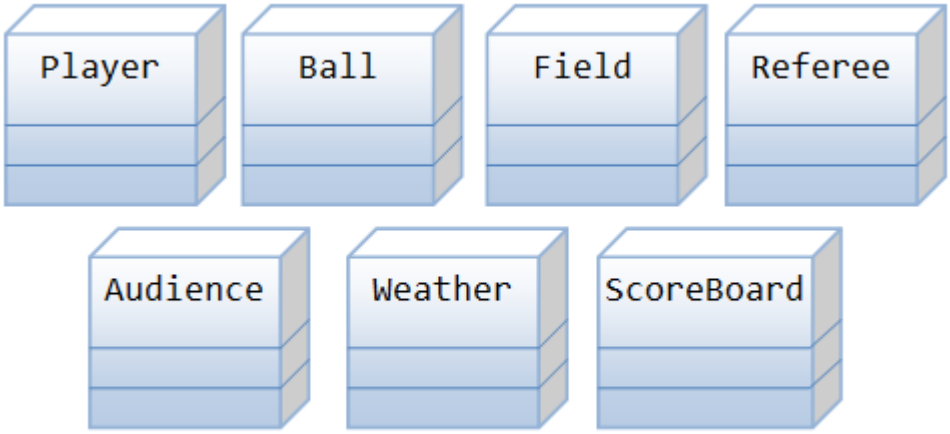
1.2 Object-Oriented Programming Languages



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

Object-oriented programming (OOP) languages are designed to overcome these problems.

1. The basic unit of OOP is a *class*, which encapsulates both the *static attributes* and *dynamic behaviors* within a "box", and specifies the public interface for using these boxes. Since the class is well-encapsulated (compared with the function), it is easier to reuse these classes. In other words, OOP combines the data structures and algorithms of a software entity inside the same box.
2. OOP languages permit *higher level of abstraction* for solving real-life problems. The traditional procedural language (such as C and Pascal) forces you to think in terms of the structure of the computer (e.g. memory bits and bytes, array, decision, loop) rather than thinking in terms of the problem you are trying to solve. The OOP languages (such as Java, C++, C#) let you think in the problem space, and use software objects to represent and abstract entities of the problem space to solve the problem.



Classes (Entities) in a Computer Soccer Game

As an example, suppose you wish to write a computer soccer games (which I consider as a complex application). It is quite difficult to model the game in procedural-oriented languages. But using OOP languages, you can easily model the program accordingly to the "real things" appear in the soccer games.

- Player: attributes include name, number, location in the field, and etc; operations include run, jump, kick-the-ball, and etc.
- Ball:
- Reference:
- Field:
- Audience:
- Weather:

Most importantly, some of these classes (such as `Ball` and `Audience`) can be reused in another application, e.g., computer basketball game, with little or no modification.

1.3 Benefits of OOP

The procedural-oriented languages focus on procedures, with function as the basic unit. You need to first figure out all the functions and then think about how to represent data.

The object-oriented languages focus on components that the user perceives, with objects as the basic unit. You figure out all the objects by putting all the data and operations that describe the user's interaction with the data.

Object-Oriented technology has many benefits:

- *Ease in software design* as you could think in the problem space rather than the machine's bits and bytes. You are dealing with high-level concepts and abstractions. Ease in design leads to more productive software development.
- *Ease in software maintenance*: object-oriented software are easier to understand, therefore easier to test, debug, and maintain.

- *Reusable software*: you don't need to keep re-inventing the wheels and re-write the same functions for different situations. The fastest and safest way of developing a new application is to reuse existing codes - fully tested and proven codes.

2. OOP Basics

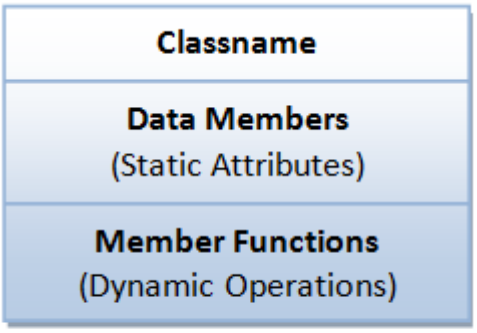
2.1 Classes & Instances

Class: A *class is a definition of objects of the same kind*. In other words, a *class* is a blueprint, template, or prototype that defines and describes the *static attributes* and *dynamic behaviors* common to all objects of the same kind.

Instance: An *instance* is *a realization of a particular item of a class*. In other words, an instance is an *instantiation* of a class. All the instances of a class have similar properties, as described in the class definition. For example, you can define a class called "`Student`" and create three instances of the class "`Student`" for "`Peter`", "`Paul`" and "`Pauline`".

The term "*object*" usually refers to *instance*. But it is often used quite loosely, which may refer to a class or an instance.

2.2 A Class is a 3-Compartment Box encapsulating Data and Functions



A class is a 3-compartment box encapsulating data and functions

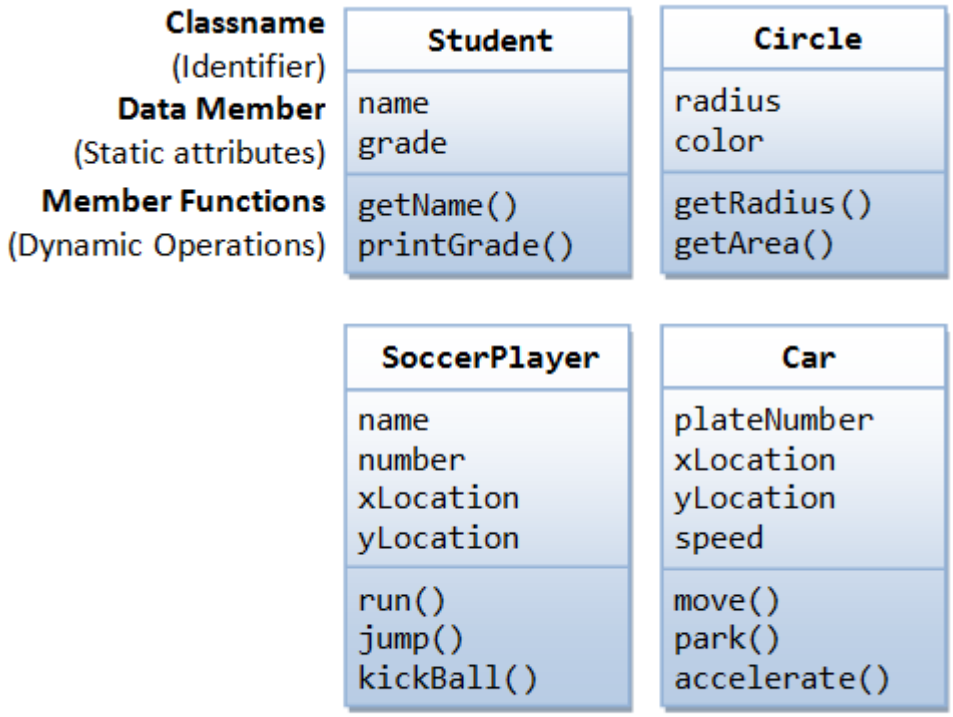
A class can be visualized as a three-compartment box, as illustrated:

1. ***Classname*** (or identifier): identifies the class.
2. ***Data Members*** or ***Variables*** (or *attributes*, *states*, *fields*): contains the *static attributes* of the class.
3. ***Member Functions*** (or *methods*, *behaviors*, *operations*): contains the *dynamic operations* of the class.

In other words, a class encapsulates the static attributes (data) and dynamic behaviors (operations that operate on the data) in a box.

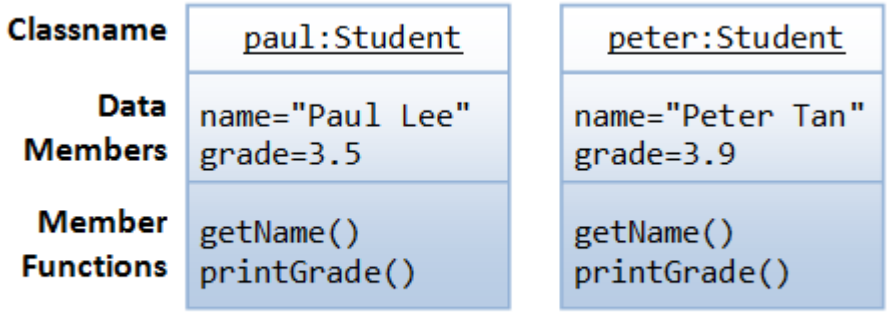
Class Members: The *data members* and *member functions* are collectively called *class members*.

The followings figure shows a few examples of classes:



Examples of classes

The following figure shows two instances of the class `Student`, identified as "`paul`" and "`peter`".



Two instances of the **Student** class

Unified Modeling Language (UML) Class and Instance Diagrams: The above class diagrams are drawn according to the UML notations. A class is represented as a 3-compartment box, containing name, data members (variables), and member functions, respectively. classname is shown in bold and centralized. An instance (object) is also represented as a 3-compartment box, with instance name shown as `instanceName:Classname` and underlined.

Brief Summary

1. A *class* is a programmer-defined, abstract, self-contained, reusable software entity that mimics a real-world thing.

2. A class is a 3-compartment box containing the name, data members (variables) and the member functions.
3. A class encapsulates the data structures (in data members) and algorithms (member functions). The values of the data members constitute its *state*. The member functions constitute its *behaviors*.
4. An *instance* is an instantiation (or realization) of a particular item of a class.

2.3 Class Definition

In C++, we use the keyword `class` to define a class. There are two sections in the class declaration: `private` and `public`, which will be explained later. For examples,

```
class Circle {          // classname
private:
   double radius;       // Data members (variables)
   string color;
public:
   double getRadius(); // Member functions
   double getArea();
}
class SoccerPlayer {    // classname
private:
   int number;          // Data members (variables)
   string name;
   int x, y;
public:
   void run();          // Member functions
   void kickBall();
}
```

Class Naming Convention: A classname shall be a noun or a noun phrase made up of several words. All the words shall be initial-capitalized (camel-case). Use a *singular* noun for classname. Choose a meaningful and self-descriptive classname. For examples, `SoccerPlayer`, `HttpProxyServer`, `FileInputStream`, `PrintStream` and `SocketFactory`.

2.4 Creating Instances of a Class

To create *an instance of a class*, you have to:

1. Declare an instance identifier (name) of a particular class.
2. Invoke a constructor to construct the instance (i.e., allocate storage for the instance and initialize the variables).

For examples, suppose that we have a class called `Circle`, we can create instances of `Circle` as follows:

```
// Construct 3 instances of the class Circle: c1, c2, and c3
Circle c1(1.2, "red");  // radius, color
Circle c2(3.4);         // radius, default color
Circle c3;              // default radius and color
```

Alternatively, you can invoke the constructor explicitly using the following syntax:

```
Circle c1 = Circle(1.2, "red");  // radius, color
```

```
Circle c2 = Circle(3.4);          // radius, default color
Circle c3 = Circle();             // default radius and color
```

2.5 Dot (.) Operator

To reference a *member of a object* (data member or member function), you must:

1. First identify the instance you are interested in, and then
2. Use the *dot operator* (`.`) to reference the member, in the form of *`instanceName.memberName`*.

For example, suppose that we have a class called `Circle`, with two data members (`radius` and `color`) and two functions (`getRadius()` and `getArea()`). We have created three instances of the class `Circle`, namely, `c1`, `c2` and `c3`. To invoke the function `getArea()`, you must first identity the instance of interest, says `c2`, then use the *dot operator*, in the form of `c2.getArea()`, to invoke the `getArea()` function of instance `c2`.

For example,

```
// Declare and construct instances c1 and c2 of the class Circle
Circle c1(1.2, "blue");
Circle c2(3.4, "green");
// Invoke member function via dot operator
cout << c1.getArea() << endl;
cout << c2.getArea() << endl;
// Reference data members via dot operator
c1.radius = 5.5;
c2.radius = 6.6;
```

Calling `getArea()` without identifying the instance is meaningless, as the radius is unknown (there could be many instances of `Circle` - each maintaining its own radius).

In general, suppose there is a class called *`AClass`* with a data member called *`aData`* and a member function called *`aFunction()`*. An instance called *`anInstance`* is constructed for *`AClass`*. You use *`anInstance.aData`* and *`anInstance.aFunction()`*.

2.6 Data Members (Variables)

A *data member* (*variable*) has a *name* (or *identifier*) and a *type*; and holds a *value* of that particular type (as descried in the earlier chapter). A data member can also be an instance of a certain class (to be discussed later).

Data Member Naming Convention: A data member name shall be a noun or a noun phrase made up of several words. The first word is in lowercase and the rest of the words are initial-capitalized (camel-case), e.g., `fontSize`, `roomNumber`, `xMax`, `yMin` and `xTopLeft`. Take note that variable name begins with an lowercase, while classname begins with an uppercase.

2.7 Member Functions

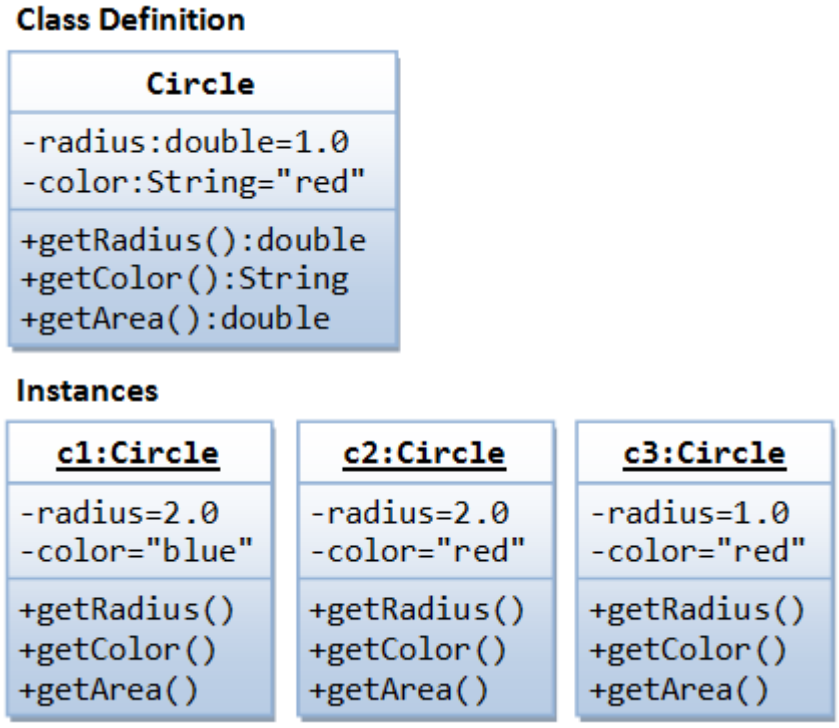
A member function (as described in the earlier chapter):

1. receives parameters from the caller,
2. performs the operations defined in the function body, and
3. returns a piece of result (or void) to the caller.

Member Function Naming Convention: A function name shall be a verb, or a verb phrase made up of several words. The first word is in lowercase and the rest of the words are initial-capitalized (camel-case). For example, `getRadius()`, `getParameterValues()`.

Take note that data member name is a noun (denoting a static attribute), while function name is a verb (denoting an action). They have the same naming convention. Nevertheless, you can easily distinguish them from the context. Functions take arguments in parentheses (possibly zero argument with empty parentheses), but variables do not. In this writing, functions are denoted with a pair of parentheses, e.g., `println()`, `getArea()` for clarity.

2.8 Putting them Together: An OOP Example



A class called `Circle` is to be defined as illustrated in the class diagram. It contains two data members: `radius` (of type `double`) and `color` (of type `String`); and three member functions: `getRadius()`, `getColor()`, and `getArea()`.

Three instances of `Circle`s called `c1`, `c2`, and `c3` shall then be constructed with their respective data members, as shown in the instance diagrams.

In this example, we shall keep all the codes in a single source file called `CircleAIO.cpp`.

CircleAIO.cpp

```
1  /* The Circle class (All source codes in one file) (CircleAIO.cpp) */
2  #include <iostream>    // using IO functions
```

```
3  #include <string>      // using string
4  using namespace std;
5
6  class Circle {
7  private:
8     double radius;      // Data member (Variable)
9     string color;       // Data member (Variable)
10
11 public:
12    // Constructor with default values for data members
13    Circle(double r = 1.0, string c = "red") {
14       radius = r;
15       color = c;
16    }
17
18    double getRadius() {  // Member function (Getter)
19       return radius;
20    }
21
22    string getColor() {   // Member function (Getter)
23       return color;
24    }
25
26    double getArea() {    // Member function
27       return radius*radius*3.1416;
28    }
29 };   // need to end the class declaration with a semi-colon
30
31 // Test driver function
32 int main() {
33    // Construct a Circle instance
34    Circle c1(1.2, "blue");
35    cout << "Radius=" << c1.getRadius() << " Area=" << c1.getArea()
36         << " Color=" << c1.getColor() << endl;
37
38    // Construct another Circle instance
39    Circle c2(3.4); // default color
40    cout << "Radius=" << c2.getRadius() << " Area=" << c2.getArea()
41         << " Color=" << c2.getColor() << endl;
42
43    // Construct a Circle instance using default no-arg constructor
44    Circle c3;      // default radius and color
45    cout << "Radius=" << c3.getRadius() << " Area=" << c3.getArea()
46         << " Color=" << c3.getColor() << endl;
47    return 0;
48 }
```

To compile and run the program (with GNU GCC under Windows):

```
> g++ -o CircleAIO.exe CircleAIO.cpp
   // -o specifies the output file name

> CircleAIO
Radius=1.2 Area=4.5239 Color=blue
Radius=3.4 Area=36.3169 Color=red
Radius=1 Area=3.1416 Color=red
```

2.9 Constructors

A *constructor* is a special function that has the *function name same as the classname*. In the above `Circle` class, we define a constructor as follows:

```
// Constructor has the same name as the class
Circle(double r = 1.0, string c = "red") {
   radius = r;
   color = c;
}
```

A constructor is used to construct and *initialize all the data members*. To create a new instance of a class, you need to declare the name of the instance and invoke the constructor. For example,

```
Circle c1(1.2, "blue");
Circle c2(3.4);      // default color
Circle c3;           // default radius and color
                     // Take note that there is no empty bracket ()
```

A constructor function is different from an ordinary function in the following aspects:

- The name of the constructor is the same as the classname.
- Constructor has no return type (or implicitly returns `void`). Hence, no `return` statement is allowed inside the constructor's body.
- Constructor can only be invoked *once* to initialize the instance constructed. You cannot call the constructor afterwards in your program.
- Constructors are not inherited (to be explained later).

2.10 Default Arguments for Functions

In C++, you can specify the default value for the trailing arguments of a function (including constructor) in the function header. For example,

```
1  /* Test function default arguments (TestFnDefault.cpp) */
2  #include <iostream>
3  using namespace std;
4
5  // Function prototype
6  int sum(int n1, int n2, int n3 = 0, int n4 = 0, int n5 = 0);
7
8  int main() {
9     cout << sum(1, 1, 1, 1, 1) << endl; // 5
10    cout << sum(1, 1, 1, 1) << endl;    // 4
11    cout << sum(1, 1, 1) << endl;       // 3
12    cout << sum(1, 1) << endl;          // 2
13 // cout << sum(1) << endl;  // error: too few arguments
14 }
15
16 // Function definition
17 // The default values shall be specified in function prototype,
18 //   not the function implementation
19 int sum(int n1, int n2, int n3, int n4, int n5) {
20    return n1 + n2 + n3 + n4 + n5;
21 }
```

2.11 "public" vs. "private" Access Control Modifiers

An *access control modifier* can be used to control the visibility of a data member or a member function within a class. We begin with the following two access control modifiers:

1. **`public`**: The member (data or function) is accessible and available to *all* in the system.
2. **`private`**: The member (data or function) is accessible and available *within this class only*.

For example, in the above `Circle` definition, the data member `radius` is declared `private`. As the result, `radius` is accessible inside the `Circle` class, but NOT outside the class. In other words, you cannot use "`c1.radius`" to refer to `c1`'s `radius` in `main()`. Try inserting the statement "`cout << c1.radius;`" in `main()` and observe the error message:

```
CircleAIO.cpp:8:11: error: 'double Circle::radius' is private
```

Try moving `radius` to the `public` section, and re-run the statement.

On the other hand, the function `getRadius()` is declared `public` in the `Circle` class. Hence, it can be invoked in the `main()`.

UML Notation: In UML notation, `public` members are denoted with a "`+`", while `private` members with a "`-`" in the class diagram.

2.12 Information Hiding and Encapsulation

A class encapsulates the static attributes and the dynamic behaviors into a "3-compartment box". Once a class is defined, you can seal up the "box" and put the "box" on the shelve for others to use and reuse. Anyone can pick up the "box" and use it in their application. This cannot be done in the traditional procedural-oriented language like C, as the static attributes (or variables) are scattered over the entire program and header files. You cannot "cut" out a portion of C program, plug into another program and expect the program to run without extensive changes.

Data member of a class are typically hidden from the outside word, with `private` access control modifier. Access to the private data members are provided via `public` assessor functions, e.g., `getRadius()` and `getColor()`.

This follows the principle of *information hiding*. That is, objects communicate with each others using well-defined interfaces (public functions). Objects are not allowed to know the implementation details of others. The implementation details are hidden or encapsulated within the class. Information hiding facilitates reuse of the class.

Rule of Thumb: Do not make any data member `public`, unless you have a good reason.

2.13 Getters and Setters

To allow other to *read* the value of a `private` data member says `xxx`, you shall provide a *get function* (or *getter* or *accessor function*) called `getXxx()`. A getter need not expose the data

in raw format. It can process the data and limit the view of the data others will see. Getters shall not modify the data member.

To allow other classes to *modify* the value of a `private` data member says `xxx`, you shall provide a *set function* (or *setter* or *mutator function*) called `setXxx()`. A setter could provide data validation (such as range checking), and transform the raw data into the internal representation.

For example, in our `Circle` class, the data members `radius` and `color` are declared `private`. That is to say, they are only available within the `Circle` class and not visible outside the `Circle` class - including `main()`. You cannot access the `private` data members `radius` and `color` from the `main()` directly - via says `c1.radius` or `c1.color`. The `Circle` class provides two public accessor functions, namely, `getRadius()` and `getColor()`. These functions are declared `public`. The `main()` can invoke these public accessor functions to retrieve the `radius` and `color` of a `Circle` object, via says `c1.getRadius()` and `c1.getColor()`.

There is no way you can change the `radius` or `color` of a `Circle` object, after it is constructed in `main()`. You cannot issue statements such as `c1.radius = 5.0` to change the `radius` of instance `c1`, as `radius` is declared as `private` in the `Circle` class and is not visible to other including `main()`.

If the designer of the `Circle` class permits the change the `radius` and `color` after a `Circle` object is constructed, he has to provide the appropriate setter, e.g.,

```
// Setter for color
void setColor(string c) {
   color = c;
}

// Setter for radius
void setRadius(double r) {
   radius = r;
}
```

With proper implementation of *information hiding*, the designer of a class has full control of what the user of the class can and cannot do.

2.14 Keyword "this"

You can use keyword "`this`" to refer to *this* instance inside a class definition.

One of the main usage of keyword `this` is to resolve ambiguity between the names of data member and function parameter. For example,

```
class Circle {
private:
   double radius;                 // Member variable called "radius"
   ......
public:
   void setRadius(double radius) { // Function's argument also called
"radius"
```

```
      this->radius = radius;
         // "this.radius" refers to this instance's member variable
         // "radius" resolved to the function's argument.
   }
   ......
}
```

In the above codes, there are two identifiers called `radius` - a data member and the function parameter. This causes naming conflict. To resolve the naming conflict, you could name the function parameter `r` instead of `radius`. However, `radius` is more approximate and meaningful in this context. You can use keyword `this` to resolve this naming conflict. "`this->radius`" refers to the data member; while "`radius`" resolves to the function parameter.

"`this`" is actually a *pointer* to this object. I will explain pointer and the meaning of "`->`" operator later.

Alternatively, you could use a prefix (such as `m_`) or suffix (such as _) to name the data members to avoid name crashes. For example,

```
class Circle {
private:
   double m_radius;  // or radius_
   ......
public:
   void setRadius(double radius) {
      m_radius = radius;  // or radius_ = radius
   }
   ......
}
```

C++ Compiler internally names their data members beginning with a leading underscore (`e.g., _xxx`) and local variables with 2 leading underscores (e.g., `__xxx`). Hence, avoid name beginning with underscore in your program.

2.15 "const" Member Functions

A `const` member function, identified by a `const` keyword at the end of the member function's header, cannot modifies any data member of this object. For example,

```
double getRadius() const {  // const member function
   radius = 0;
      // error: assignment of data-member 'Circle::radius' in read-only
structure
   return radius;
}
```

2.16 Convention for Getters/Setters and Constructors

The constructor, getter and setter functions for a `private` data member called `xxx` of type `T` in a class `Aaa` have the following conventions:

```
class Aaa {
private:
```

```
   // A private variable named xxx of type T
   T xxx;
public:
   // Constructor
   Aaa(T x) { xxx = x; }
   // OR
   Aaa(T xxx) { this->xxx = xxx; }
   // OR using member initializer list (to be explained later)
   Aaa(T xxx) : xxx(xxx) { }

   // A getter for variable xxx of type T receives no argument and return a
value of type T
   T getXxx() const { return xxx; }

   // A setter for variable xxx of type T receives a parameter of type T
and return void
   void setXxx(T x) { xxx = x; }
   // OR
   void setXxx(T xxx) { this->xxx = xxx; }
}
```

For a `bool` variable `xxx`, the getter shall be named `isXxx()`, instead of `getXxx()`, as follows:

```
private:
   // Private boolean variable
   bool xxx;
public:
   // Getter
   bool isXxx() const { return xxx; }

   // Setter
   void setXxx(bool x) { xxx = x; }
   // OR
   void setXxx(bool xxx) { this->xxx = xxx; }
```

2.17 Default Constructor

A default constructor is a constructor with no parameters, or having default values for all the parameters. For example, the above `Circle`'s constructor can be served as default constructor with all the parameters default.

```
Circle c1;   // Declare c1 as an instance of Circle, and invoke the default
constructor
Circle c1(); // Error!
             // (This declares c1 as a function that takes no parameter and
returns a Circle instance)
```

If C++, if you did not provide ANY constructor, the compiler automatically provides a default constructor that does nothing. That is,

```
ClassName::ClassName() { }  // Take no argument and do nothing
```

Compiler will not provide a default constructor if you define any constructor(s). If all the constructors you defined require arguments, invoking no-argument default constructor results in error. This is to allow class designer to make it impossible to create an *uninitialized* instance, by NOT providing an explicit default constructor.

2.18 Constructor's Member Initializer List

Instead of initializing the private data members inside the body of the constructor, as follows:

```
Circle(double r = 1.0, string c = "red") {
   radius = r;
   color = c;
}
```

We can use an alternate syntax called *member initializer list* as follows:

```
Circle(double r = 1.0, string c = "red") : radius(r), color(c) { }
```

Member initializer list is placed after the constructor's header, separated by a colon (`:`). Each initializer is in the form of *`data_member_name(parameter_name)`*. For fundamental type, it is equivalent to *`data_member_name = parameter_name`*. For object, the constructor will be invoked to construct the object. The constructor's body (empty in this case) will be run after the completion of member initializer list.

It is recommended to use member initializer list to initialize all the data members, as it is often more efficient than doing assignment inside the constructor's body.

2.19 *Destructor

A *destructor*, similar to constructor, is a special function that has the same name as the classname, with a prefix `~`, e.g., `~Circle()`. Destructor is called implicitly when an object is destroyed.

If you do not define a destructor, the compiler provides a default, which does nothing.

```
class MyClass {
public:
   // The default destructor that does nothing
   ~MyClass() { }
......
}
```

Advanced Notes

- If your class contains data member which is dynamically allocated (via `new` or `new[]` operator), you need to free the storage via `delete` or `delete[]`.

2.20 *Copy Constructor

A *copy constructor* constructs a new object by copying an existing object of the same type. In other words, a copy constructor takes an argument, which is an object of the same class.

If you do not define a copy constructor, the compiler provides a default which copies all the data members of the given object. For example,

```
Circle c4(7.8, "blue");
cout << "Radius=" << c4.getRadius() << " Area=" << c4.getArea()
```

```
     << " Color=" << c4.getColor() << endl;
                // Radius=7.8 Area=191.135 Color=blue

// Construct a new object by copying an existing object
// via the so-called default copy constructor
Circle c5(c4);
cout << "Radius=" << c5.getRadius() << " Area=" << c5.getArea()
     << " Color=" << c5.getColor() << endl;
                // Radius=7.8 Area=191.135 Color=blue
```

The copy constructor is particularly important. When an object is passed into a function *by value*, the copy constructor will be used to make a clone copy of the argument.

Advanced Notes

- Pass-by-value for object means calling the copy constructor. To avoid the overhead of creating a clone copy, it is usually better to pass-by-reference-to-`const`, which will not have side effect on modifying the caller's object.
- The copy constructor has the following signature:
- `class MyClass {`
- `private:`
- `   T1 member1;`
- `   T2 member2;`
- `public:`
- `   // The default copy constructor which constructs an object via memberwise copy`
- `   MyClass(const MyClass & rhs) {`
- `      member1 = rhs.member1;`
- `      member2 = rhs.member2;`
- `   }`
- `.....`
  `}`

- The default copy constructor performs *shadow copy*. It does not copy the dynamically allocated data members created via `new` or `new[]` operator.

2.21 *Copy Assignment Operator (=)

The compiler also provides a default assignment operator (=), which can be used to assign one object to another object of the same class via memberwise copy. For example, using the `Circle` class defined earlier,

```
Circle c6(5.6, "orange"), c7;
cout << "Radius=" << c6.getRadius() << " Area=" << c6.getArea()
     << " Color=" << c6.getColor() << endl;
                // Radius=5.6 Area=98.5206 Color=orange
cout << "Radius=" << c7.getRadius() << " Area=" << c7.getArea()
     << " Color=" << c7.getColor() << endl;
                // Radius=1 Area=3.1416 Color=red (default constructor)

c7 = c6; // memberwise copy assignment
cout << "Radius=" << c7.getRadius() << " Area=" << c7.getArea()
     << " Color=" << c7.getColor() << endl;
                // Radius=5.6 Area=98.5206 Color=orange
```

Advanced Notes

- You could overload the assignment opeator to override the default.
- The copy constructor, instead of copy assignment operator, is used in declaration:

```
Circle c8 = c6;  // Invoke the copy constructor, NOT copy assignment operator
                 // Same as Circle c8(c6)
```

- The default copy assignment operator performs *shadow copy*. It does not copy the dynamically allocated data members created via `new` or `new[]` operator.
- The copy assignment operator has the following signature:

```
class MyClass {
private:
   T1 member1;
   T2 member2;
public:
   // The default copy assignment operator which assigns an object via memberwise copy
   MyClass & operator=(const MyClass & rhs) {
      member1 = rhs.member1;
      member2 = rhs.member2;
      return *this;
   }
......
}
```

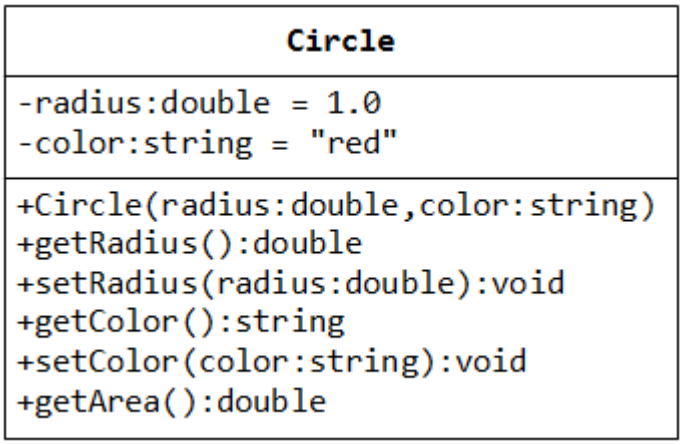
- The copy assignment operator differs from the copy constructor in that it must release the dynamically allocated contents of the target and prevent self assignment. The assignment operator shall return a reference of this object to allow chaining operation (such as `x = y = z`).
- The default constructor, default destructor, default copy constructor, default copy assignment operators are known as *special member functions*, in which the compiler will automatically generate a copy if they are used in the program and not explicitly defined.

3. Separating Header and Implementation

For better software engineering, it is recommended that the class declaration and implementation be kept in 2 separate files: declaration is a header file "`.h`"; while implementation in a "`.cpp`". This is known as separating the public interface (header declaration) and the implementation. Interface is defined by the designer, implementation can be supplied by others. While the interface is fixed, different vendors can provide different implementations. Furthermore, only the header files are exposed to the users, the implementation can be provided in an object file "`.o`" (or in a library). The source code needs not given to the users.

I shall illustrate with the following examples.

4. Example: The Circle Class



Instead of putting all the codes in a single file. We shall "separate the interface and implementation" by placing the codes in 3 files.

1. `Circle.h`: defines the public interface of the `Circle` class.
2. `List.h`: Dyanmic Array of the `Circle` class.
3. `CircleApp.cpp`: A test driver program for the `Circle` class.

Circle.h – Header

```
#ifndef  CIRCLE_H
#define  CIRCLE_H

#include<iostream>

using namespace std;

class Circle{

private :
        double radius;
        string color;
public:

        Circle(void);
        Circle(double,string);
        double getRadius();
```

```
        void setRadius(double radius);
        String getColor();
        Void setColor(string color);
        double getArea();
        Double getCircum();

};

#include"Circle.cpp"

#endif
```

Circle.cpp

```
    Circle:: Circle(void){

        This->radius=0;

        This->color="";

}

double Circle::getRadius(){

        return this->radius;

}


void Circle:: setRadius(double radius){

        this->radius=radius;

}


String Circle:: getColor(){

        return this->color;

}
```

```
void Circle::setColor(string color){
        this->color=color;
}
 double Circle:: getArea(){
   double pi=3.14159;
   return pi*radius*radius;
  }

   double Circle:: getCircum(){
        double pi=3.14159;
        return 2*pi*radius;
   }
```

CircleList.h – Header

```
#ifndef CIRCLE_LIST_H
#define  CIRCLE_LIST_H
#include<iostream>
using namespace std;
struct circle_struct{
```

```
        double area;
        double circum;
        string color;
        circle_struct *next;
};

struct circle_struct *head=NULL;
struct circle_struct *newCircle=NULL;

class CircleList{

public :
void add(double,double,string);
void view();
void remove();

};

#include"CircleList.cpp"

#endif
```

CircleList.h

```
void CircleList::add(double area,double circum,string color){

        newCircle=new circle_struct;
        newCircle->area=area;
        newCircle->circum=circum;
        newCircle->color=color;
        newCircle->next=NULL;
```

```
        if(head==NULL){

                head=newCircle;
                return;
        }

        circle_struct *curr=head;
        while(curr){

                if(curr->next==NULL){

                        curr->next=newCircle;
                        return;
                }

                curr=curr->next;
        }

}
void CircleList::view(){

        circle_struct *curr=head;

        if(curr==NULL){
                 cout<<"it is empty no record";

        }

        while(curr){

       cout<<"\n----------------------------\n";
                cout<<"Area :"<<curr->area<<"\n";
                cout<<"Circumference  :"<<curr->circum<<"\n";
                cout<<"Color  :"<<curr->color<<"\n";
```

```
            cout<<"\n----------------------------\n";
            curr=curr->next;

        }


}


void CircleList::remove(){

        if(head==NULL){
            cout<<"it is empty";
        }

        if(head->next==NULL){
            head=NULL;
            return;
        }

        circle_struct *curr=head;
        circle_struct *prev=head;

        while(curr){

            if(curr->next==NULL){
                prev->next=NULL;
                return;
            }
            prev=curr;
            curr=curr->next;
        }

}
```

CircleApp.cpp - Application

```
#include<iostream>

#include "CircleList.h"



using namespace std;

//circle List

CircleList circles;

void mainMENU(){

        cout<<"\n    Circle MENU              \n";
        cout<<" [ 1 ] Add New Circle          \n";
        cout<<" [ 2 ] View Circles            \n";
        cout<<" [ 3 ] Remove Circle           \n";
        cout<<" [ 4 ]Exit Program             \n";

}

int getChoice(){

        int choice;

        cout<<"Enter Choice:";
        cin>>choice;
        return choice;
}

double getRadius(){
```

```
        double radius;
        cout<<"Enter Radius:";
        cin>>radius;
        return radius;
}

string getColor(){

        string color;
        cout<<"Enter Color:";
        cin>>color;
        return color;


}

void addCircle(){

        double radius=getRadius();
        string color=getColor();
        Circle c(radius,color);
        double area=c.getArea();
        double circum=c.getCircum();



        circles.add(area,circum,color);
        cout<<"Successfully Added A new Circle Record...";



}

void viewCircles(){
```

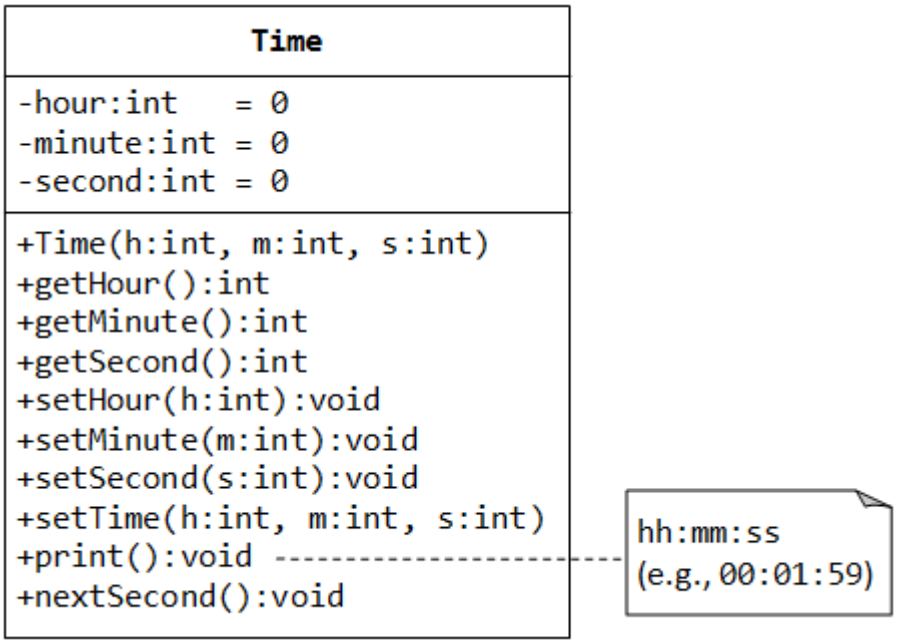
```
        cout<<"\n ---------------View Circle Records----------------------------\n";
        circles.view();


}

void removeCircle(){
        cout<<"\n ---------------Delete Circle Record----------------------------\n";
        circles.remove();
        cout<<"\nSuccessfully Deleted...\n";
}

int main(){

        while(true){
        mainMENU();
        switch(getChoice()){

                case 1:addCircle();break;
                case 2:viewCircles();break;
                case 3:removeCircle();break;
                case 4:cout<<"Goodbye...\n";return 0; break;
        }


    }
}
```

Program Notes:

- The implementation file provides the *definition* of the functions, which are omitted from the *declaration* in the header file.
- #include "Circle.h"
  The compiler searches the headers in double quotes (such as `"Circle.h"`) in the *current directory* first, then the system's include directories. For header in angle bracket (such as `<iostream>`), the compiler does NOT searches the current directory, but only the system's include directories. Hence, use double quotes for user-defined headers.
- Circle::Circle(double r, string c) {
  You need to include the *`className`*`::` (called *class scope resolution operator*) in front of all the members names, so as to inform the compiler this member belong to a particular class.
  (Class Scope: Names defined inside a class have so-called *class scope*. They are visible within the class only. Hence, you can use the same name in two different classes. To use these names outside the class, the class scope resolution operator *`className`*`::` is needed.)
- You CANNOT place the default arguments in the implementation (they shall be placed in the header). For example,

```
Circle::Circle(double r = 1.0, string c = "red") {   // error!
```

5. Example: The Time Class



Let's write a class called Time, which models a specific instance of time with hour, minute and second values, as shown in the class diagram.

The class Time contains the following members:

- Three private data members: hour (0-23), minute (0-59) and second (0-59), with default values of 0.
- A public constructor Time(), which initializes the data members hour, minute and second with the values provided by the caller.
- public getters and setters for private data members: getHour(), getMinute(), getSecond(), setHour(), setMinute(), and setSecond().
- A public member function setTime() to set the values of hour, minute and second given by the caller.
- A public member function print() to print this Time instance in the format "hh:mm:ss", zero-filled, e.g., 01:30:04.
- A public member function nextSecond(), which increase this instance by one second. nextSecond() of 23:59:59 shall be 00:00:00.

Let's write the code for the Time class, with the header and implementation separated in two files: Time.h and Time.cpp.

Header – Time.h

```
 // Include this "block" only if TIME_H is NOT defined
#ifndef TIME_H
#define TIME_H

 #include<iostream>
 #include<iomanip>

 using namespace std;


class Time{


        private:

                int hour;
                int minute;
                int second;
        public:

                void setHour(int hour){
                        this->hour=hour;
                }
                void setMinute(int minute){
                        this->minute=minute;
                }
                void setSecond(int second){
                        this->second=second;
                }

                int getHour(){
```

```
            return this->hour;
        }
        int getMinute(){

            return this->minute;
        }
        int getSecond(){
            return this->second;
        }

        Time();
        Time(int);
        Time(int,int,int);
        void setTime(int,int,int);
        void nextSecond();
        void print();



};


#include"Time.h"

#endif
```

Time.cpp

```
Time::Time(){
        this->hour=0;
        this->minute=0;
        this->second=0;
}

Time::Time(int h){
        this->hour=h;

}

Time::Time(int h,int m,int s){
        this->hour=h;
        this->minute=m;
        this->second=s;
}

void Time::setTime(int h,int m,int s){
        hour=h;
        minute=m;
        second=s;
}

void Time::nextSecond() {
  ++second;
  if (second >= 60) {
    second = 0;
    ++minute;
  }
  if (minute >= 60) {
    minute = 0;
    ++hour;
```

```
  }
  if (hour >= 24) {
    hour = 0;
  }
}


void Time::print(){

        // zero-filled, need <iomanip>, sticky
       cout << setfill('0');
        // set width to 2 spaces, need <iomanip>, non-sticky
   cout << setw(2) << hour << ":" << setw(2) << minute << ":" << setw(2) <<
second << endl;

}
```

Dissecting Time.h

```
#ifndef TIME_H
#define TIME_H
......
#endif
```

To prevent an header file from included *more than once* into a source file (which could result in compilation error if an entity is declared twice, e.g., int i), we wrap the header codes within a pair of *preprocessor directives* #ifndef (if not define) and #endif. The codes within the if-block will only be included if the identifier TIME_H has not been defined. This is true for the first inclusion, which also defines the identifier TIME_H (the first directive in body of the if-block). No subsequent inclusion is possible, since TIME_H has been defined during the first inclusion. By convention, use the identifier XXX_H (or XXX_H_INCLUDED) for header Xxx.h.

```
class Time {
private:
......
public:
......
```

};
The header `Time.h` contains the *class declaration* for the class `Time`. It is divided into two sections: `private` and `public`. The `private` members (data or functions) are accessible by members of this class only, while `public` members are visible by all (such as the `main()` function which is outside the class). The class declaration must be terminated by a semicolon.

```
private:
   int hour;
   int minute;
   int second;
public:
   ......
```

TestTime.cpp

```
/* Test Driver for the Time class (TestTime.cpp) */
#include <iostream>
#include<stdlib.h>

// include header of Time class
#include "Time.h"
using namespace std;
Time t;
int getHour(){

        int h;
        cout<<"Enter Hour:";
        cin>>h;
        return h;
}

 int getMinute(){

        int m;
        cout<<"Enter Minute:";
        cin>>m;
        return m;
```

```
}

 int getSecond(){

        int s;
        cout<<"Enter Second:";
        cin>>s;
        return s;
}

 void timeMENU(){

        cout<<"   Time MENU       \n";
        cout<<" 1] Input Time     \n";
        cout<<" 2] View Time      \n";
        cout<<" 3] Next Second    \n";
        cout<<" 4] Exit Program   \n";
 }

 int getChoice(){

        int choice;
        cout<<"Enter Choice:";
        cin>>choice;
        return choice;

 }
 void inputTime(){

        int h=getHour();
        int m=getMinute();
        int s=getSecond();
        t.setTime(h,m,s);
   cout<<"Successfully inputed new Time...\n";
```

```
}

void viewTime(){
        cout<<"\n -----View Current Time----\n";
        t.print();
        cout<<"\n -------------------------\n";

}
void viewNextSecond(){

        cout<<"\n-------Next Second--------\n";
        for(int i=0;i<10;i++){

        t.nextSecond();
        t.print();
   }
        cout<<"\n -------------------------\n";
}

void exitProgram(){
        cout<<"Thanks for using my program...\n";
        exit(0);


}
int main() {

        while(true){

        timeMENU();
        switch(getChoice()){
                case 1:inputTime();break;
                case 2:viewTime();break;
                case 3:viewNextSecond();break;
```

```
                case 4:exitProgram();break;

        }


    }
}
```

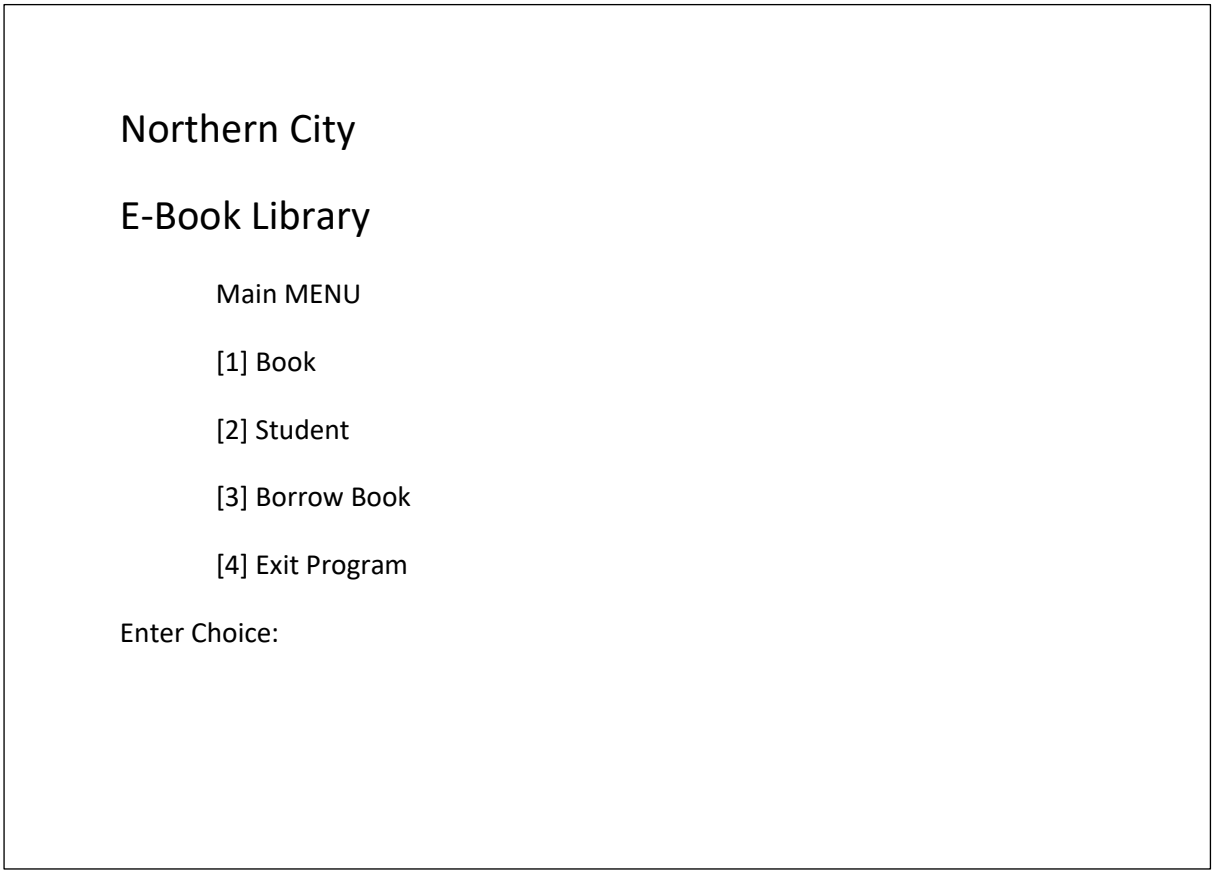
Do the following Exercise

Header files

- **Book.h**
- **BookList.h**
- **Student.h**
- **StudentList.h**
- **Borrow.h**
- **BorrowList.h**

Application File

- **BookStoreApp.cpp**



Northern City

==========

Book Menu

1] Add new Book

2] View Book

3] Update Book Info

4] Search Book Info

5] Delete Book

6] Exit Book Menu

Enter Choice:

Northern City

==========

Student Menu

1] Add new Student

2] View Students

3] Update Student Info

4] Search Student Info

5] Delete Student

6] Exit Student Menu

Enter Choice:

Book Class

- Book ID
- Title
- Author
- Published date
- Book Value

Student Class

- Student ID
- Student Name
- Phone
- Address
- Course

Borrow Class

- Borrow ID
- Borrow Date
- Borrow Fees
- Book ID
- Book Title
- Student ID
- Student Name
- Fine
- Total Amount

Project Sample

IMCS College, Student Enrolment System

Identification of Objects, Attributes, Data types and methods for Student Enrolment System

Required Objects	: 1) IMCS
	2) Student
Required Data Types	:1)Integer
	2)Double
	3)String
	4)Boolean
Required Attributes	: 1) centerName
	2) centerNumber
	3)location
	3)studentRollNumber
	4)studentName
	5) studentAge
	6)studentMajor
	7)enrollDate
	8)fees
	9)duration
Required Methods	: 1) getCenterNumber()

2) setCenterNumber(int centerNumber)

3) getCenerName()

4) setCenterName(String centerNumber)

5) getCenterLocation()

6) setCenterLocation(String centerLocation)

7) setCenterDetailInfo(int centerNo,String centerName,String location)

8)getStudentRollNumber()

9) setStudentRollNumer(String rollNumber)

10) getStudnetName()

11) setStudentName(String studentName)

12) getAge()

13) setAge(int age)

14) getMajor()

15) setMajor(String major)

16) getEnrollDate()

17) setEnrollDate(String enrollDate)

18) getFees()

19) setFees(Double fees)

20) getDurationMonths()

21) setDurationMonths(int durationMonths)

22) setStudentDetailInfo(int rollNo,String name,int age,String major,String enrollDate,double fees,int months)

23)getStudentDetailInfo()

24)printCenterDetailInfo()

25)printStudentDetailInfo()

26) registerStudent()

27)enrollStudentFile()

28) deleteStudentRecord()

29) updateStudentRecord()

30) findStudentRecord()

31) sortStudentRecord()

32) isRegistered()

33) isRollNumberExists()

An Object Oriendted Programming solution

Class : IMCS

Attributes:
centerNumber
centerName
location

Methods:
setCenterNumber()
getCenterNumber()
setCenterName()
getCenterName()
setLocation()
getLocation()
setCenterDetailInfo(int centerNo,String centerName,String location)

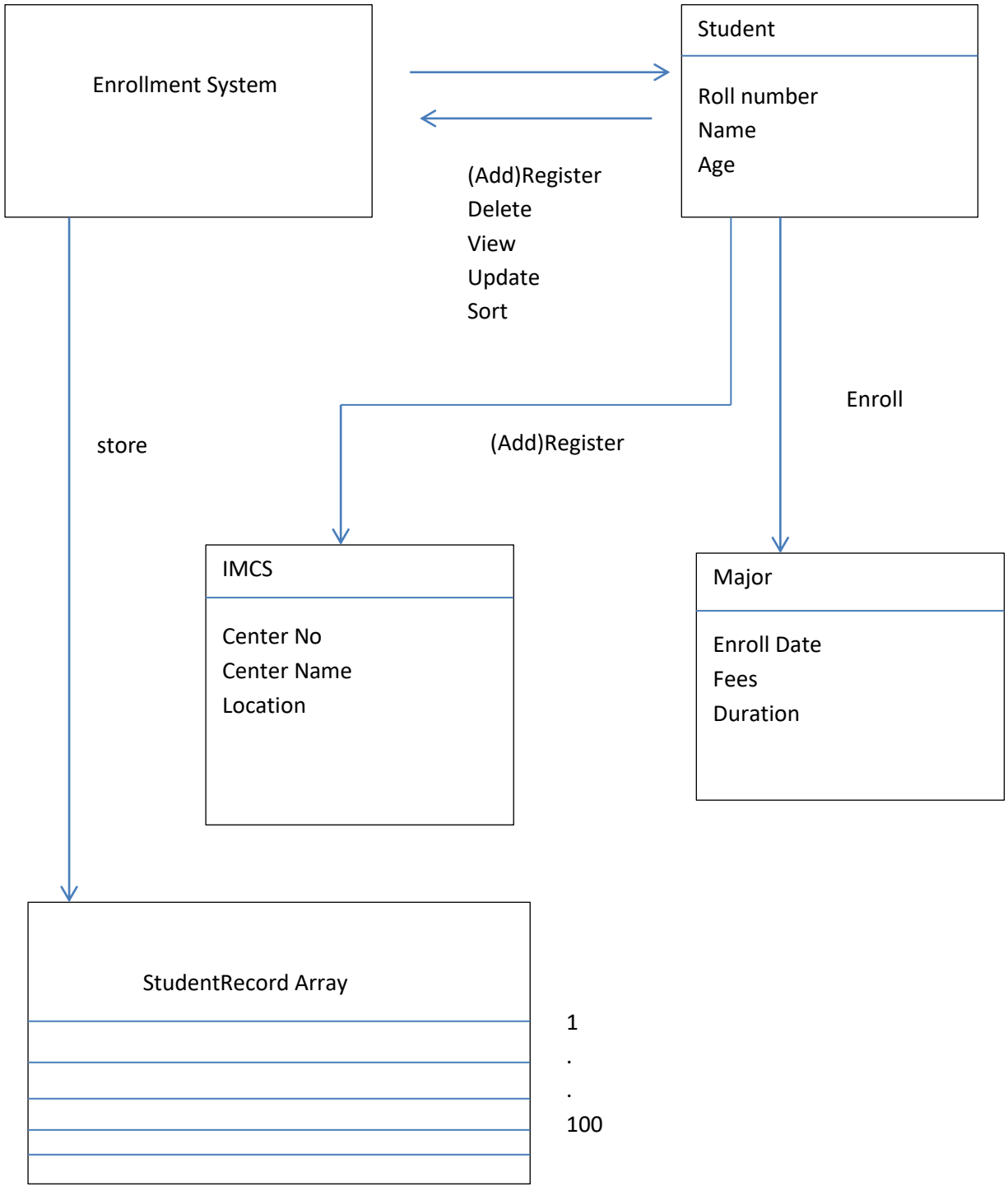
printCenterDetailInfo()

Class : Student

Attributes:
rollNumber
Name
Age
Major
enrollDate
Fees
Duration

Methods:
setStudentRollNumber()
getStudentRollNumber()
setName()
getName()
setAge()
getAge()
setMajor()
getMajor()
setEnrollDate()
getEnrollDate()
setFees()
getFees()
setDuration()
getDurtaion()
setStudentDetailInfo()
printStudentDetailInfo()

Relationship



Gantt chart Diagram

Tasks	Week 1	Week 2	Week 3	Week4
Phase 1	Do Required Data and Informtion Gathering Regarding Enrollment System			
Phase 2		Do feasibility study - Budget and Technical Knowledge Scope		
Phase 3			Application Coding using C++	
Phase 4				Testing in Live Environemnt And refurbishment

The Effectivenes of BloodShed Dev-C++ IDE

Feature Rich

As with any development studio Bloodshed Dev-C++ IDE helps with the routine and tedius tasks of programming to let a programmer focus on the actual design and building work. The editor assists with syntax and the autocomplete will suggest answers to whatever we are trying to code. Forget the next part of a code chunk? Start typing what we think it should be and this IDE will help us find what we need. Search through the libraries for functions and other useful code snippets or ask the community for help. We will be coding our project in no time.

Projects from Start to Finish

This compiler will allow us to stay in Dev-C++ from start to finish. Compose code and run functions through the debugger line by line in order to find problems. Once the code is compiling correctly and bug free use the compiler to create an exe for distribution and use. Bloodshed Dev-C++ is one shot free IDE to take any project to development.

Source Code:

```
//import library for cin/ cout function
#include<iostream>
//import library for setw() function
#include<iomanip>
//import library for system
#include<stdlib.h>

using namespace std;

class Major{

	private:
		string majorName;
		string enrollDate;
		double fees;
		int duration;
	public:
		//default constructor or constructor initialization
		Major(){
			majorName="";
			enrollDate="";
			fees=0;
			duration=0;
		}
		//constructor with parameters

		Major(string m,string ed,double fe,int du){
			majorName=m;
			enrollDate=ed;
			fees=fe;
			duration=du;
		}

		//setters and getters
		void setMajorName(string m){
			majorName=m;
		}
		string getMajorName(){
			return majorName;
		}
```

```
		void setEnrollDate(string date){
			enrollDate=date;
		}
		string getEnrollDate(){
			return enrollDate;
		}
		void setFees(double fe){
			fees=fe;
		}
		double getFees(){
			return fees;
		}
		void setDuration(int du){
			duration=du;
		}
		int getDuration(){
			return duration;
		}
};

//declare class student
class Student:public Major{

	private:
	int Sr;
	string rollNumber;
	string name;
	int age;


	public:
		//default Constructor
		Student(){
			Sr=1;
			rollNumber="";
			name="";
			age=0;

		}
		//Constructor with student record parameters
		Student(int sr,string rn,string n,int a,string mj,string ed,double fe,int
months):Major(mj,ed,fe,months){


			Sr=sr;
			rollNumber=rn;
			name=n;
			age=a;

		}
```

```
		//setters and getters

		int getSr(){
			return Sr+1;
		}

		void setSr(int sr){
			Sr=sr;
		}
		string getRollNumber(){
			return rollNumber;
		}
		void setRollNumber(string rn){
			rollNumber=rn;
		}
		string getName(){
			return name;

		}
		void setName(string n){
			name=n;
		}

		int getAge(){
			return age;
		}
		void setAge(int a){
			age=a;
		}
		//printing values to console
		void print(int sr){


	cout<<setw(10)<<left<<sr<<setw(10)<<left<<getRollNumber()<<setw(10)<<left<<getName()
<<setw(10)<<left<<getAge()<<setw(20)<<left<<getMajorName()<<setw(10)<<left<<getEnrollDate()<
<setw(10)<<left<<getFees()<<setw(10)<<left<<getDuration()<<"\n";
		}

};

//declare object array
static Student s[100];
//student record array index initialization
int index=0;
void mainMENU(){


	cout<<"\n	INFO Myanmar College		\n";
```

```
        cout<<"      Student Enrolment  System       \n";
        cout<<" ---------------------------------------\n";
        cout<<" [1] Add New Student Record            \n";
        cout<<" [2] View Student Record              \n";
        cout<<" [3] Delete Student Record            \n";
        cout<<" [4] Update Student Record            \n";
        cout<<" [5] Sort Student Record              \n";
        cout<<" [6] Exit Program                    \n";
        cout<<"---------------------------------------\n";
}
int getChoice(){

        int choice;
        cout<<"Enter Choice:";
        cin>>choice;
        return choice;
}

string getRollNumber(){
        string dummy;
        getline(cin,dummy);
        string rn;
        cout<<"Enter Student Roll Number:";
        getline(cin,rn);
        return rn;
}

string getName(){
        string name;
        cout<<"Enter Studnet Name:";
        getline(cin,name);
        return name;
}
int getAge(){
        int age;
        cout<<"Enter Student Age:";
        cin>>age;
        return age;
}

string getMajor(){
        string dummy;
        getline(cin,dummy);
        string major;
        cout<<"Enter Student Major Name:";
        getline(cin,major);
        return major;
}

string getEnrollDate(){
        string enrollDate;
```

```
	cout<<"Enter Enroll Date:";
	getline(cin,enrollDate);
	return enrollDate;
}

double getFees(){

	double fees;
	cout<<"Enter Fees:";
	cin>>fees;
	return fees;
}

int getDuration(){

	int duration;
	cout<<"Enter Duration in Months:";
	cin>>duration;
	return duration;
}
void addNewStudentRecord(){


	string rn=getRollNumber();
	string name=getName();
	int age=getAge();
	string major=getMajor();
	string enrollDate=getEnrollDate();
	double fees=getFees();
	int duration=getDuration();
	s[index]=Student(0,rn,name,age,major,enrollDate,fees,duration);
	index++;

}

void viewStudentRecords(){
	int sr=0;
	cout<<"     \n INFO Myanmar College Enrolment System  \n";
	cout<<"          Veiw Student Record              \n";
	for(int i=0;i<index;i++){

		cout<<"-------------------------------------------------------------------------------------------------------
-------------\n";
		sr=i+1;
		s[i].print(sr);
		cout<<"-------------------------------------------------------------------------------------------------------
-------------\n";
	}
}

void deleteStudentRecords(){
```

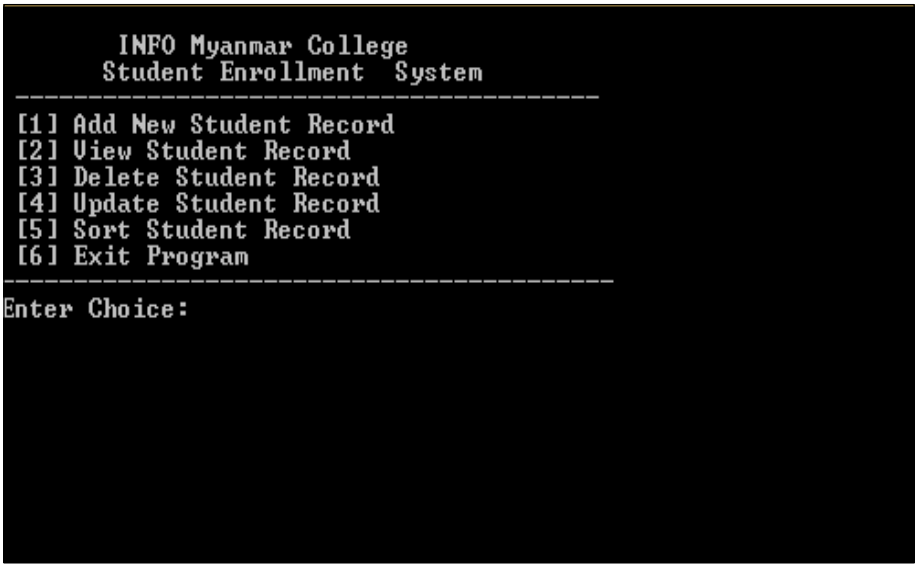
```
        string delRollNum;
        bool found=false;
        int delIndex=0;
        cout<<"     \n INFO Myanmar College Enrolment System  \n";
        cout<<"          Deleting Student Record             \n";
        cout<<"Enter Student Roll Number to Delete Record:";
        cin>>delRollNum;


        for(int i=0;i<index;i++){
                found=false;
                //search and compare student roll number from the records
                if(delRollNum.compare(s[i].getRollNumber())==0){

                        //if roll number found, store the
                        delIndex=i;
                        found=true;
                        break;
                }

        }

        if(found==true){
                //clear record contents
                s[delIndex]=Student(0,"","",0,"","",0,0);
                cout<<"Successfully Deleted a Student Record...\n";

        }
        else{

                cout<<"Sorry Student Roll Number not found...Try again...\n";
        }

}


void updateStudentRecords(){
        string updateRollNum;
        bool found=false;
        int updateIndex=0;
        cout<<"     \n INFO Myanmar College Student Enrolment System  \n";
        cout<<"          Updating Student Record             \n";
        cout<<"Enter Student Roll Number to Update Record:";
        cin>>updateRollNum;


        for(int i=0;i<index;i++){
                found=false;
                //search and compare student roll number from the records
                if(updateRollNum.compare(s[i].getRollNumber())==0){
```

```
                    //if roll number found, store the
                    updateIndex=i;
                    found=true;
                    break;
                }

        }

        if(found==true){
                //get New Inputs from the user and update each attributes
                        string rn=getRollNumber();
                        string name=getName();
                        int age=getAge();
                        string major=getMajor();
                        string enrollDate=getEnrollDate();
                        double fees=getFees();
                        int duration=getDuration();

                        s[updateIndex].setRollNumber(rn);
                        s[updateIndex].setName(name);
                        s[updateIndex].setAge(age);
                        s[updateIndex].setMajorName(major);
                        s[updateIndex].setEnrollDate(enrollDate);
                        s[updateIndex].setFees(fees);
                        s[updateIndex].setDuration(duration);


                cout<<"Successfully updated a Student Record...\n";

        }
        else{

                cout<<"Sorry Student Roll Number not found...Try again...\n";
        }

}

void sortStudentRecords(){

        Student temp;


        cout<<"     \n INFO Myanmar College Student Enrolment System  \n";
        cout<<"          Sorting Student Record              \n";



        for(int i=0;i<index-1;i++){

                for(int j=i+1;j<index;j++){
```

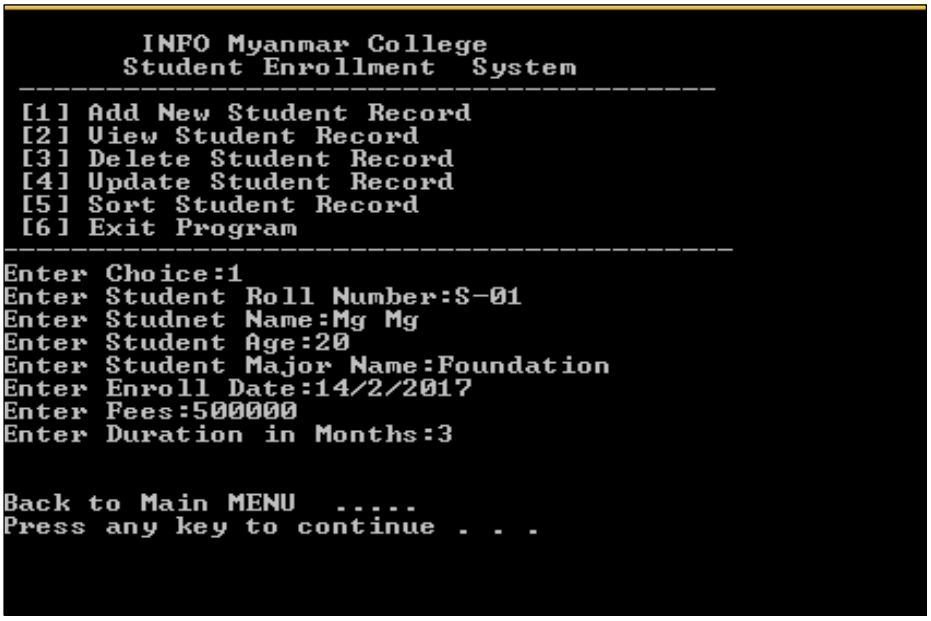
```
                        if(s[i].getName().compare(s[j].getName())>0){

                                temp=s[i];
                                s[i]=s[j];
                                s[j]=temp;
                        }
                }

        }


        cout<<"Successfully Sorted Alphabetically...";

}

void exitProgram(){
        cout<<"\n-------------------------------------\n";
        cout<<"\n Thank you very much...Exiting Now....\n";
}
void message(){
        cout<<"\n\nBack to Main MENU  .....\n";
}


int main(){

        //the program keep looping until the user press 5
           for(;;){

                mainMENU();

                switch(getChoice()){

                        case
1:addNewStudentRecord();message();system("pause");system("cls");break;
                        case
2:viewStudentRecords();message();system("pause");system("cls");break;
                        case
3:deleteStudentRecords();message();system("pause");system("cls");break;
                        case
4:updateStudentRecords();message();system("pause");system("cls");break;
                   case 5:sortStudentRecords();message();system("pause");system("cls");break;
                   case 6:exitProgram();system("pause");return 0;break;
                }

        }

}
```

Output Screen Displays:

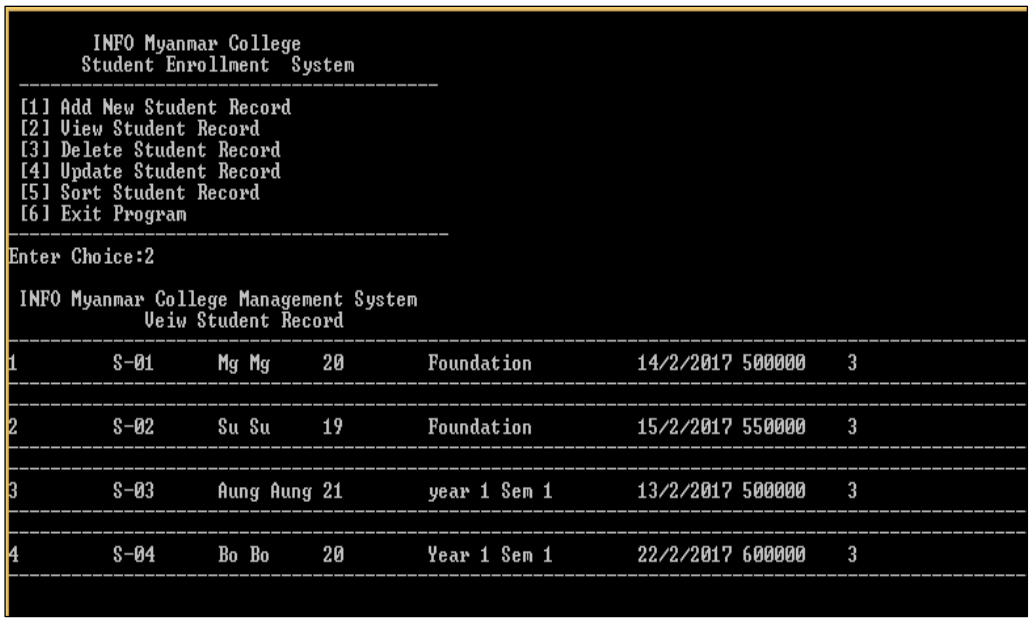
Main MENU



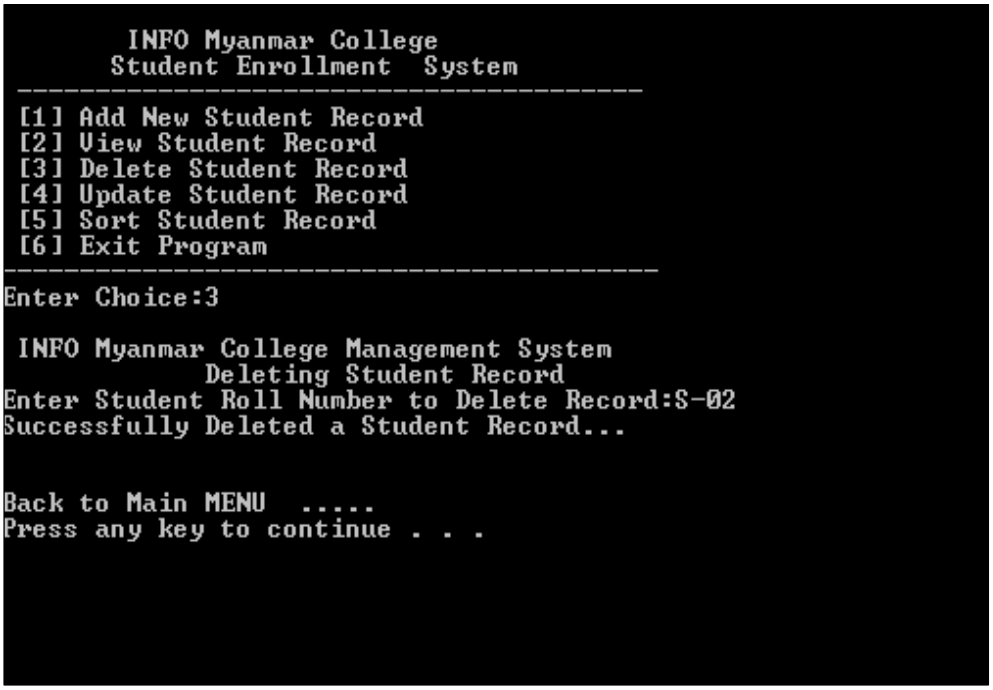
1] Add new Student Record



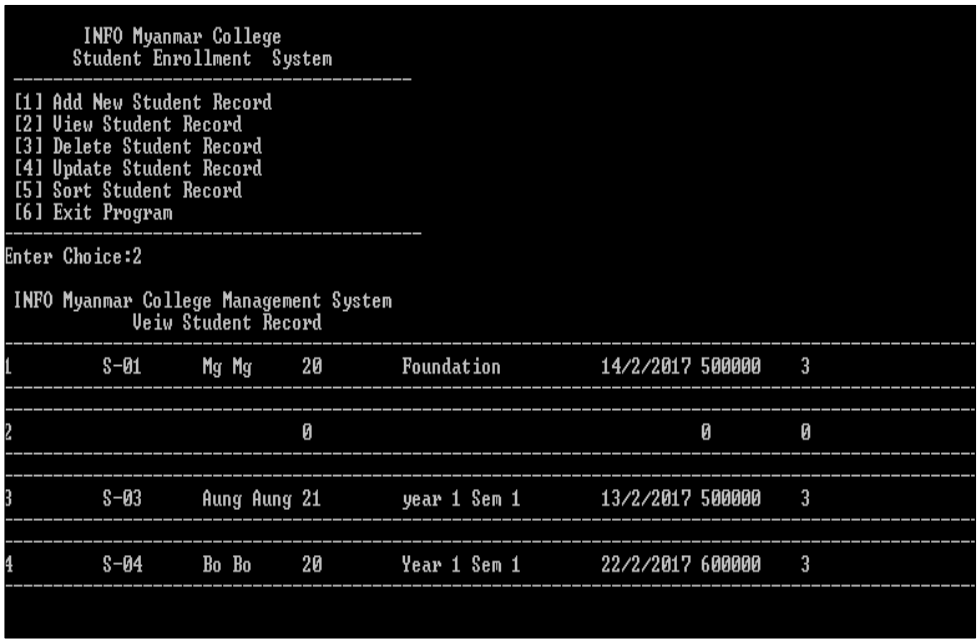
2] View Student Record

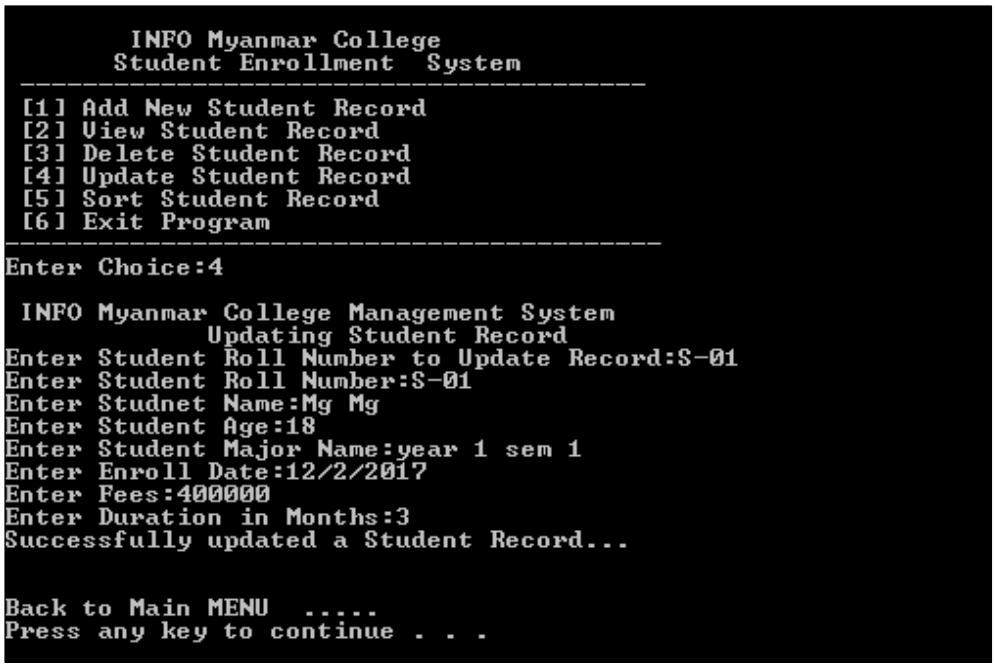


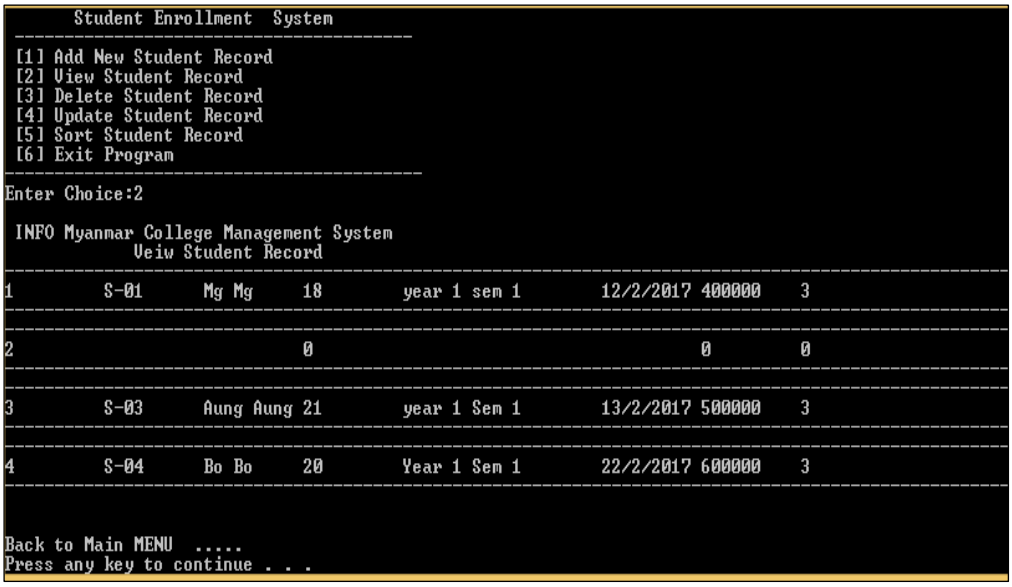
3] Delete Student Record



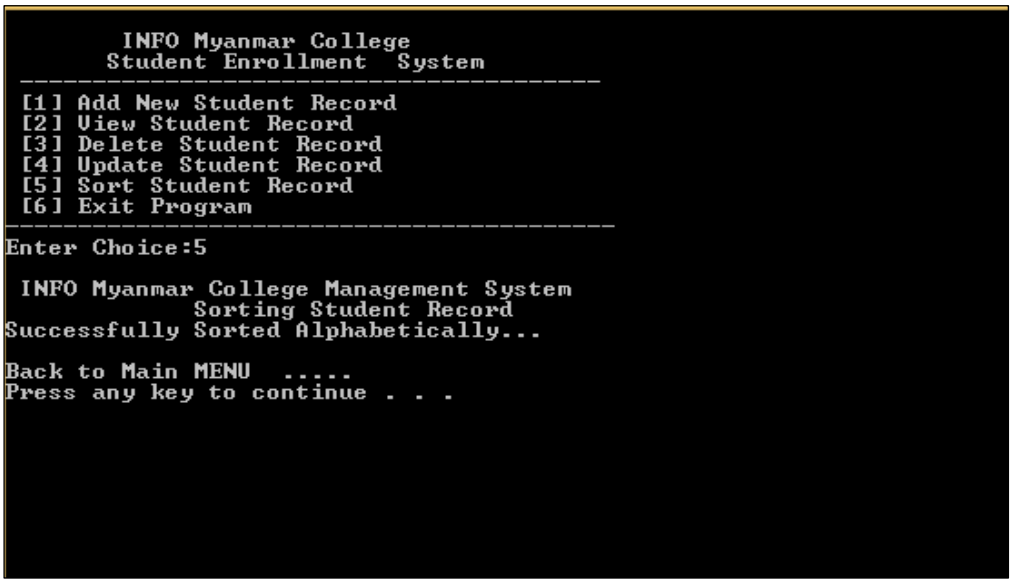
4] Update Student Record

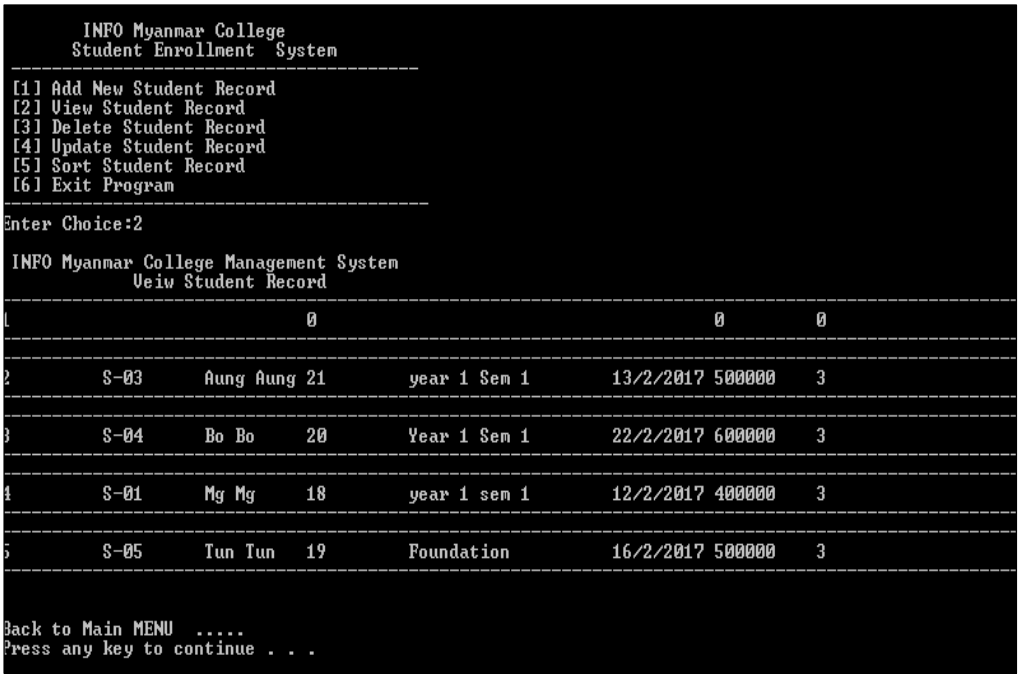




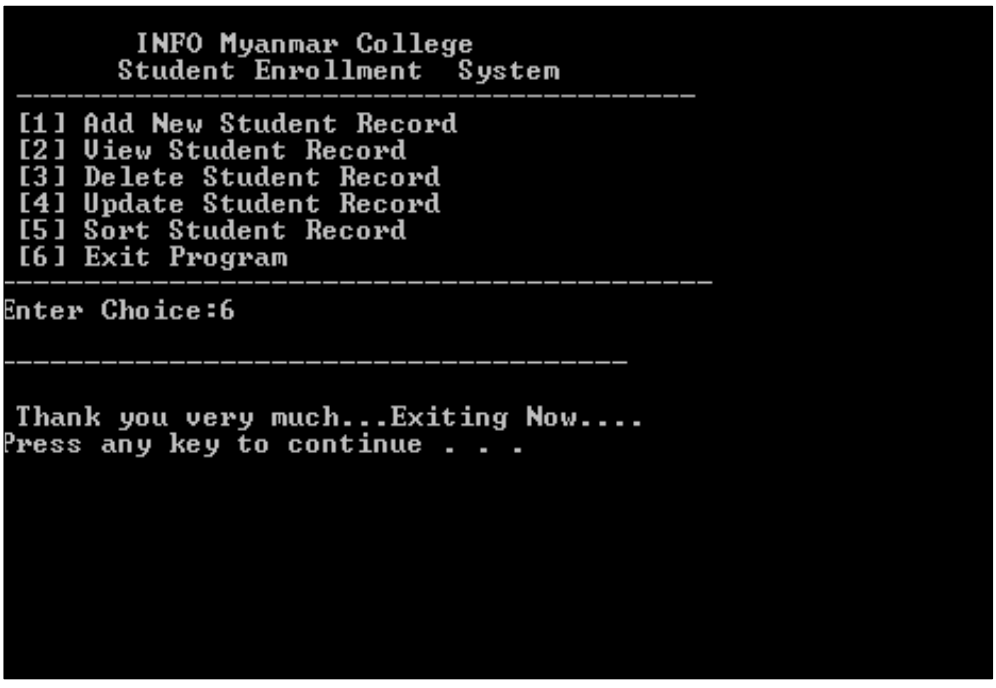


5] Sort Student Record





6] Exit Program



Do the Following Exercise

Yangon Hotel

Main MENU

==========

1] Guest List

2] Rooms

3] Reservation

5] Invoice

6] Exit Program

==========

Enter Choice:

Guest Class
- Guest ID
- Full Name
- BirthDate
- Passport No
- Religion
- Nationality
- Home Country

Room Class
- Room No
- Room Type
- Room Price
- Floor

Reservation
- Reserve No
- Guest ID
- Guest Name
- Floor

Guest.h

GuestList.h

Reservation.h

ReserveList.h

Yangonhotel.cpp

C++ Extra Exercises

1) Vector
2) Map
3) Generic (Template)
4) Tic Tac Toe Game
5) Snake Game

Vector

```
/For my C++ Students
//Vector Example

#include <vector>
#include <iostream>
#include<stdlib.h>
using namespace std;
class Contact{
private:
int id;
string name;
string phone;
public:
Contact(){
this->id=0;
this->name="";
this->phone="";
}
Contact(int id,string name,string phone){
this->id=id;
this->name=name;
this->phone=phone;
}
void setId(int id){
this->id=id;
}
int getId(){
return id;
}
void setName(string name){
this->name=name;
}
string getName(){
return name;
}
```

```
void setPhone(string phone){
this->phone=phone;
}
string getPhone(){
return phone;
}
void print(){
cout<<"\n================================================\n";
cout<<"\t"<<getId()<<"\t"<<getName()<<"\t"<<getPhone()<<"\n";
cout<<"================================================\n";
}

};
//creating contact list vector array and iterator
std::vector<Contact> contact_list;
std::vector<Contact>::iterator it;
void mainMenu(){
cout<<"========== My Phone Contact List============\n";
cout<<"= Main MENU =\n";
cout<<"============================================\n";
cout<<"= [1] Add New Contact =\n";
cout<<"= [2] View Contacts =\n";
cout<<"= [3] Update Contact =\n";
cout<<"= [4] Delete Contact =\n";
cout<<"= [5] Exit Program =\n";
cout<<"============================================\n";
}
int getChoice(){
int choice;
cout<<"Enter Choice:";
cin>>choice;
return choice;
}
int getId(){
int id;
cout<<"Enter Contact ID:";
cin>>id;
return id;
}
string getName(){
string name;
cout<<"Enter Cotact Name:";
getline(cin,name);
return name;
}
string getPhoneNo(){
string phoneno;
cout<<"Enter Phone No:";
getline(cin,phoneno);
return phoneno;
}
```

```
void addContact(){
string dummy;
int id=getId();
getline(cin,dummy);
string name=getName();
string phone=getPhoneNo();
//get id,name and phone no and put them into a class constructor
Contact contact(id,name,phone);
//put the contact class into contact list vector array
contact_list.push_back(contact);
cout<<"Successfully added a new contact \n";
}
void viewContact(){

cout<<"============ View Contact List ==============\n";
if(contact_list.empty()){
cout<<"No Contacts in the list....\n";
}
else{

for(int i=0; i<contact_list.size(); i++)
contact_list[i].print();
cout << "\n";
}
}
void updateMENU(){
cout<<" ====== Update Contact Info ======\n";
cout<<" Update MENU \n";
cout<<" [1] Update Name \n";
cout<<" [2] Update Phone No \n";
cout<<"=====================================\n";
}
void updateName(){
string dummy;
bool found=false;
int update_index=0;
string newName;
int updateID;
cout<<"Enter Contact ID to update Name:";
cin>>updateID;
for(int i=0;i<contact_list.size();i++){
if(contact_list[i].getId()==updateID){
found=true;
update_index=i;
break;
}
}
if(found==true){
```

```
cout<<"Congratualtions..Contact ID found and ready to update contact name....\n";
cout<<"Your Current Contact Name is:"<<contact_list[update_index].getName()<<"\n";
getline(cin,dummy);
cout<<"Enter New Contact Name:";
getline(cin,newName);
contact_list[update_index].setName(newName);
cout<<"Successfully updated....\n";
}
else{
cout<<" ID not found...Sorry 😟 \n";
}
}
void updatePhoneNo(){
string dummy;
string newPhone;
bool found=false;
int update_index=0;
int updateID;
cout<<"Enter Contact ID to update Phone No:";
cin>>updateID;
for(int i=0;i<contact_list.size();i++){
if(contact_list[i].getId()==updateID){
found=true;
update_index=i;
break;
}
}
if(found==true){
cout<<"Congratualtions..Contact ID found and ready to update contact phone no....\n";
cout<<"Your Current Contact Name is:"<<contact_list[update_index].getPhone()<<"\n";
getline(cin,dummy);
cout<<"Enter New Contact Phone No:";
getline(cin,newPhone);
contact_list[update_index].setName(newPhone);
cout<<"Successfully updated....\n";
}
else{
cout<<" ID not found...Sorry 😟 \n";
}
}
void updateContact(){
updateMENU();
switch(getChoice()){
case 1:updateName();break;
case 2:updatePhoneNo();break;
}
}
void deleteContact(){
bool found=false;
```

```
int del_ID;
int update_index=0;
cout<<" ================ Deleting Contact ===============\n";
cout<<"Enter Contact ID to delete contact:";
cin>>del_ID;
for(int i=0;i<contact_list.size();i++){
if(contact_list[i].getId()==del_ID){
found=true;
update_index=i;
break;
}
}
if(found==true){
it=contact_list.begin() + update_index;
contact_list.erase(it);
cout<<" found and deleted... \n";
}
else{
cout<<"Sorry...Contact ID not found..try again...\n";
}
}
void exitProgram(){
cout<<"Thanks for using my program...Goodbye...\n";
exit(0);
}
int main(){
while(true){
mainMenu();
switch(getChoice()){
case 1:addContact();break;
case 2:viewContact();break;
case 3:updateContact();break;
case 4:deleteContact();break;
case 5:exitProgram();break;
}
}
}
```

Practice your own Exercise

Main MENU
Todo List

1] Add Task
2] View Task
3] Delete Task
4] Exit Program

Map<Key,Value>

```
/*
Objectives: to be able to understand hashing or using pair memory allocation unit <Key,Value>

*/
#include <iostream>
#include <map>
#include <string>
#include <iterator>
using namespace std;
//write all the prototypes
void addCountryAndCity();
void viewCountryAndCity();
void updateCapitalCity();
void searchCapitalCity();
void deleteCountry();
//declaration of map to be used
map<string, string> worldMap;
```

```
int mainMENU(){
int choice=0;
cout<<" World Map MENU \n";
cout<<"[1] Add Country and Capital City \n";
cout<<"[2] view Country and Capital City \n";
cout<<"[3] Update Country and Capital City \n";
cout<<"[4] Search Country and Capital City \n";
cout<<"[5] Dlete Country and Capital City \n";
cout<<"[6] Exit Program \n";
cout<<"----------------------------------\n";
cout<<"Enter choice:";
cin>>choice;
return choice;
}
int main()
{
int menuChoice=0;
do{
menuChoice=mainMENU();
switch(menuChoice){
case 1:addCountryAndCity();break;
case 2:viewCountryAndCity();break;
case 3:updateCapitalCity();break;
case 4:searchCapitalCity();break;
case 5:deleteCountry();break;
}

if(menuChoice==6){
cout<<"Thanks for using my program...exit now...\n";
}
}while(menuChoice!=6);
}
string getCountryName(){

string dummy;
string country_name;

getline(cin,dummy);
cout<<"Enter Country Name:";
getline(cin,country_name);
return country_name;
}
string getCapitalCity(){

string capital_city;
cout<<"Enter Capital City:";
getline(cin,capital_city);
return capital_city;
```

```
}
void addCountryAndCity(){
string country_name=getCountryName();
string capital_city=getCapitalCity();

// Check if Country is already existed or not
if(worldMap.insert(make_pair(country_name, capital_city)).second == false)
{
cout<<"Country is already existed "<<endl;
}
else{
//insert country name and capital city into worldmap which is of Map data type

worldMap.insert(make_pair(country_name,capital_city));
cout<<"Successfully added a new country and capital city...\n";
}
}
void viewCountryAndCity(){
map<string, string>::iterator it = worldMap.begin();
while(it != worldMap.end())
{
cout<<"Country Name: => "<<it->first<<" and Capital City:=> "<<it->second<<std::endl;
it++;
}
}
void searchCapitalCity(){
cout<<"Searching Capital City...\n";
string country=getCountryName();
if(worldMap.find(country) != worldMap.end()){
cout<<"Congratulaions!!!...\n";
cout<<"Found Result:\n";
cout<<"Country is:"<<country<<" and Capital City =>"<<worldMap[country]<<"\n";
}

}
void updateCapitalCity(){
string updateCity;
cout<<"Updating Capital City...\n";
string country=getCountryName();
if(worldMap.find(country) != worldMap.end()){
cout<<"Congratulaions!!!...\n";
cout<<"Found Result:\n";
cout<<"Country is:"<<country<<" and your current Capital City is =>"<<worldMap[country]<<"\n";
cout<<"Enter a new name of Capital City to update:";
getline(cin,updateCity);
worldMap[country]=updateCity;
cout<<"Successfully updated a new capital City...\n";
```

```
}
else{
cout<<"Sorry...Country not found..try again...\n";
}
}
void deleteCountry(){
cout<<"Deleting Country and Capital City...\n";
string del_country=getCountryName();
if(worldMap.find(del_country) != worldMap.end()){

worldMap.erase(del_country);
cout<<"Successfully deleted...\n";

}
else{
cout<<"Sorry...Country not found..try again...\n";
}
}
```

Practice the following:

Contact List

[1] Add Contact

[2] View Contacts

[3] Delete Contact

[4] Update Contact

[5] Exit Program

Map <Key, Value>

Contact <Name, PhoneNo>

Generic Function

Traditional Function Example

(This function to be converted into Generic Form)

```
#include<iostream>
using namespace std;
//traditional functions
int add1(int x,int y){
return x+y;
}
float add2(float x,float y){
return x+y;
}
double add3(double x,double y){
return x+y;
}
int main(){
int a=9;
int b=3;
float c=8.5;
double d=3.14;

int ans1=add1(a,b);
cout<<"The answer 1 is:"<<ans1<<"\n";
float ans2=add2(b,c);
cout<<"THe answer 2 is:"<<ans2<<"\n";
double ans3=add3(c,d);
cout<<"The answre 3 is:"<<ans3<<"\n";
}
```

Example 2

```
//generic functions in C++
//This program is rewritten in template or generic form to the first sample

#include<iostream>
using namespace std;
template <class A,class B>
B add(A x,B y){
return x+y;
}
int main(){
int a=9;
int b=3;
float c=8.5;
double d=3.14;

int ans1=add(a,b);
cout<<"The answer 1 is:"<<ans1<<"\n";
float ans2=add(b,c);
cout<<"THe answer 2 is:"<<ans2<<"\n";
double ans3=add(c,d);
cout<<"The answre 3 is:"<<ans3<<"\n";
}
```

Tic Tac Toe Game

```
#include<stdlib.h>
#include <iostream>
using namespace std;
char box[9] = {'0','0','0','0','0','0','0','0','0'};
bool moreMove(char b[]);
int checkwin(int);
void board();
int main()
{
int player = 0;
int i;
int choice;
char symbol;
int count=0;
for(;;)
{
board();

player=(count%2)?1:2;
cout << "Player " << player << " turn , enter a number(1-9): ";
cin >> choice;
symbol=(player == 1) ? 'X' : 'O';
if (choice == 1 && box[0] == '0')
box[0] = symbol;
else if (choice == 2 && box[1] == '0')
box[1] = symbol;
else if (choice == 3 && box[2] == '0')
box[2] = symbol;
else if (choice == 4 && box[3] == '0')
box[3] = symbol;
else if (choice == 5 && box[4] == '0')
box[4] = symbol;
else if (choice == 6 && box[5] == '0')
box[5] = symbol;
else if (choice == 7 && box[6] == '0')
box[6] = symbol;
else if (choice == 8 && box[7] == '0')
box[7] = symbol;
else if (choice == 9 && box[8] == '0')
```

```
box[8] = symbol;
else
{
cout<<"Invalid move ";
count--;
cin.ignore();
cin.get();
}

if(checkwin(player)==1){
board();
cout<<"\n Game Over !! ";
cout<<"==>\a Player 1 win \n";
}
if(checkwin(player)==2){
board();
cout<<"\n Game Over !.... ";
cout<<"==>\a Player 2 win \n";
break;
}
//check more moves are still possible in box array
if(moreMove(box)==false){
board();
cout<<"==>\a Game Over!!\n it is Draw!!... \n";
break;
}
count++;
}

}
bool moreMove(char b[]){
bool empty_found=false;
for(int i=0;i<9;i++){
empty_found=false;
if(b[i]=='0'){
empty_found=true;
break;
}
}
return empty_found;
}
/*********************************************
FUNCTION TO RETURN GAME STATUS
---------------------------
if function return 1 => Player One Win
if function return 2=> Palyer Two Win
*********************************************/
int checkwin(int player)
{
switch(player){
```

```
case 1: if ( ( box[0] == 'X' && box[1] == 'X' && box[2]=='X') ||

(box[3]=='X' && box[4]=='X' && box[5]=='X') ||

(box[6] == 'X' && box[7] =='X'&& box[8] =='X') ||

(box[0] == 'X' && box[3]=='X' && box[6] == 'X') ||
(box[1] =='X' && box[4] =='X' && box[7]=='X') ||

(box[2] =='X' && box[5]=='X' && box[8]=='X') ||

(box[0] =='X' && box[4]=='X' && box[8]=='X') ||

(box[2] == 'X' && box[4]=='X' && box[6] == 'X') )
return 1;
case 2: if ((box[0] == 'O' && box[1] == 'O' && box[2]=='O') ||

(box[3]=='O' && box[4]=='O' && box[5]=='O') ||

(box[6] == 'O' && box[7] =='O'&& box[8] =='O') ||

(box[0] == 'O' && box[3]=='O' && box[6] == 'O') ||
(box[1] =='O' && box[4] =='O' && box[7]=='O') ||

(box[2] =='O' && box[5]=='O' && box[8]=='O') ||

(box[0] =='O' && box[4]=='O' && box[8]=='O') ||

(box[2] == 'O' && box[4]=='O' && box[6] == 'O') )
return 2;

}
}
/*********************************************************************
FUNCTION TO DRAW BOARD OF TIC TAC TOE WITH PLAYERS SYMBOLS( PLAYER 1- X
AND PLAYER 2- O )
*********************************************************************/

void board()
{
system("CLS");
cout << "\n\n\tTic Tac Toe Game Volume 1.0 \n\n";
cout << "Player 1 (symbol- X ) - Player 2 (symbol- O )" << endl << endl;
cout << endl;
cout << "     |     |     " << endl;
cout << "  " << box[0] << "  |  " << box[1] << "  |  " << box[2] << endl;
cout << "_____|_____|_____" << endl;
cout << "     |     |     " << endl;
```

```
cout << " " << box[3] << " | " << box[4] << " | " << box[5] << endl;
cout << "_____|_____|_____" << endl;
cout << " | | " << endl;
cout << " " << box[6] << " | " << box[7] << " | " << box[8] << endl;
cout << " | | " << endl << endl;
}
```

Create your own version of TIC TAC TOE Game

Board 4x4

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Snake Game

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

using namespace std;

bool gameover;
const int width=20;
const int height=20;
int x,y,fruitx,fruity;

enum directions{TOP=0,LEFT,RIGHT,UP,DOWN};
directions dir;

int nTail;
int tailX[100],tailY[100];


int score;

void setup();
void draw();
void intput();
void logic();

int main(){

	setup();

	while(!gameover){

		draw();
		intput();
		logic();
	}


}

void setup(){
```

```
	gameover=false;
	dir=TOP;
	x=width % 2;
	y=height % 2;
	fruitx=rand() % width;
	fruity=rand() % height;
	score=0;
	nTail=0;

}

void draw(){

	system("cls");
	cout<<"    Snake Game 1.0  "<<endl;
	cout<<"-----------------------------"<<endl;
	cout<<"Score   :"<<score<<endl;
	cout<<"---------------------------------"<<endl;


	//top bar

	for(int i=0;i<width+2;i++)
		cout<<"#";
		cout<<endl;

	for(int i=0;i<height;i++){
		for(int j=0;j<width;j++){


			if(j==0)
				cout<<"#";

			if(j==x && i==y){

				cout<<"O";
			}
			else if(fruity==i && fruitx==j){
				cout<<"F";
			}
			else{

				bool print=false;
				for(int k=0;k<nTail;k++){
					if(j==tailX[k] && i==tailY[k]){
						cout<<"0";
						print=true;
```

```
                                        }
                                }

                                if(!print){
                                        cout<<" ";
                                }
                        }



                        if(j==width-1)
                                cout<<"#";

                }

                cout<<endl;
        }




        //bottom bar

        for(int j=0;j<width+2;j++)
                cout<<"#";
                cout<<endl;

}

void intput(){

        if(_kbhit()){
                switch(_getch()){
                        case 'a': dir=LEFT;break;
                        case 's': dir=DOWN;break;
                        case 'd': dir=RIGHT;break;
                        case 'w': dir=UP;break;
                        case 'x': cout<<"Exit Game"<<endl;gameover=true;break;
                }
        }
}

void logic(){

        int tempX=tailX[0];
        int tempY=tailY[0];
        int currX,currY;
```

```
        tailX[0]=x;
        tailY[0]=y;

        for(int i=1;i<=nTail;i++){

                currX=tailX[i];
                currY=tailY[i];
                tailX[i]=tempX;
                tailY[i]=tempY;
                tempX=currX;
                tempY=currY;

        }

        switch(dir){

                case LEFT:x--;break;
                case RIGHT:x++;break;
                case UP:y--;break;
                case DOWN:y++;break;
                default:break;
        }

        if(x==fruitx && y==fruity){
                        fruitx=rand() % width;
                        fruity=rand() % height;
                        score+=10;
                        nTail++;
        }
}
```

Discussions and Results

Basic Project Structure

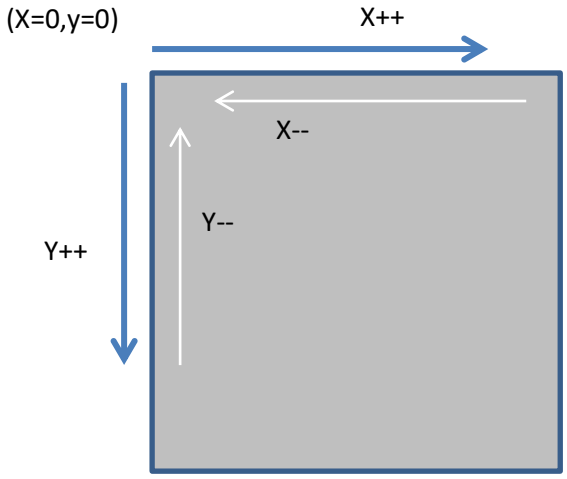
```
Setup()

While(!gameOver){

        Draw();

        Input();

        Logic();

}
```

```
//Initialization of settings

void setup(){

        //initialize all the variables here

}
```

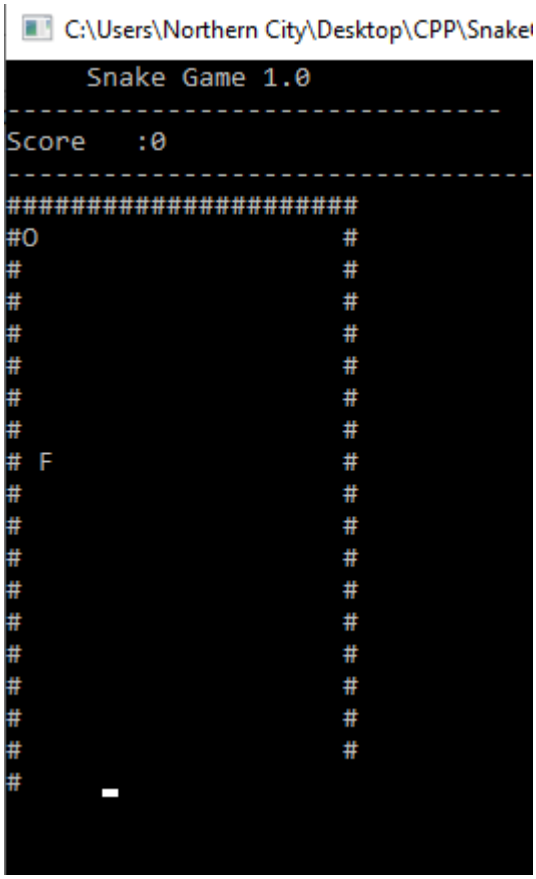
//drawing game board and characters

```
void draw(){
//draw top bar
//draw body parts
        ⇨ Draw Snake
        ⇨ Draw Fruit
        ⇨ Check Collision and reposition and draw  Snake and Fruit
        ⇨ draw tails

//draw bottom bar
```

```
}


void input(){
        If(_kbhit()){
                switch(_getch()){
                        assign andy key for the directions ( LEFT,RIGHT,UP and DOWN);
                }
        }
}


void logic(){
        switch(dir){
                case LEFT:x--;break;
                case RIGHT:x++;break;
                case UP:y--;break;
                case DOWN:y++;break;
                default:break;
        }
}
```



Output:


C:\Users\Northern City\Desktop\CPP\Snake
Snake Game 1.0
Score :0

Northern City C++ Course Exam

Exam Type: ***Take Home Exam*** Date: _____

Examinee Info:

Name: ----------
Email: ----------
Signature : ----------
Phone: ----------
Submission Date: _____

Conducted By:
Andrew Tin Mai Zaw
B.Scs. Computer Science
(University of the Philippines -Los Banos)
zautingmai@gmail.com
09425026458

Marking Scheme

Part 1 (True or False)	10 Marks
Part 2(Multiple Choice)	10 Marks
Part 3(Challenge Codes)	40 Marks
Part 4 (Project Code)	40 Marks
Total	100 marks

Passing marks: **60 Marks**

==========

===☞ Suggested Learning Tool References

1. www.google.com
2. www.youtube.com
3. http://cppreference.com/
4. www.freecodecamp.org
5. www.sololearn.com
6. https://stackoverflow.com/
7. https://hackr.io/tutorials/learn-c-plus-plus

Part 1 (True or false)

Shade the correct Answers:

1. In a switch statement every case must contain a break.

   ○ true ○ false

2. A constructor is the body of a class
   ○ true ○ false

3. A derived class inherits the constructor of the base class.
   ○ true ○ false

4. private members of a class are accessible in derived classes.
   ○ true ○ false

5. There can be a switch statement without a default clause.
   ○ true ○ false

6. C++ is was developed by Bjarne Stroustrup.
   ○ true ○ false

7. A static member function can be called using the class name instead of its objects.
   ○ true ○ false

8. The first OSI standard C++ edition was released in 1998

   ○ true ○ false

9. C++ is general purpose programming language

   ○ true ○ false

10. C++ is considered as High Level Programming Language
    ○ true ○ false

Part 2 (Multiple Choice)

Shade the correct Answers:

1. Are the following two statements equal for any value of a,b,c,d?

    1. (a==b) &&(c==d)
    2. a==b && c==d

    ☐ Yes ☐ No

2. What is the correct answer?

```
class A{
        public:
        static int cnt;
        A(int a){ cnt++; }
};
int  A::cnt=0;
int main(){
        vector<A> v(5,1);
        cout<<A::cnt<<endl;
        return 0;
}
```

    ☐ 5 ☐ 1 ☐ Runtime error ☐ compilation error

3. What is the output of this code?

```
for(;;){
        int i=0;
        if(i==1){
                cout<<I;
                break;
        }
        i++;
}
```

    ☐ 1 ☐ 0 ☐ infinite loop ☐ compilation error

4. Which type of inheritance does not exist in C++?

- ☐ Protected
- ☐ Private
- ☐ Friend
- ☐ Public

5. What is the output of this code?

```
# include<iostream>
using namespace std;
int main(){
        for(int i=0;i<7;i++){  }
                cout<<i;

}
```

☐ 6 ☐ error ☐ 0 ☐ 7

6. A recursive function needs to have a:

- ☐ Pointer
- ☐ Base Case
- ☐ If statement
- ☐ Parameter

7. What is (void *)O?

- ☐ Representation of a Null pointer
- ☐ Representation of an error
- ☐ None of the options
- ☐ Representation of an in pointer

8. Which of the following correctly declears an array?

- ☐ int array[10];
- ☐ Array array[];
- ☐ Int array;
- ☐ Array[10];

9. What is the output of this code?

```
int a=40;
int *ptr=&a;
a++;
cout<<*ptr;
```

☐ Address of a ☐ Compilation error
☐ 41 ☐ 40

10. What is the output of this code?

```
Class MyClass{
        Public:
        ~MyClass();
        MyClass();
};
MyClass :: ~MyClass(){
        cout<<"b";
}
MyClass :: MyClass(){
        cout<<"a";
}
int main(){
        MyClass obj;
}
```

☐ a ☐ ba ☐ b ☐ ab

Part 3 (Challenge Codes)

Run the following code and fill up the correct answers:

1. What is the output ?

```
int fun1(int p){
        ++p;
        return p++;
}

int fun2(int &p){
        ++p;
        return p++;
}
int main(){
        int a=1,b,c;
        b=fun1(a);
        c=fun2(b);
        cout<<a+b+c;
        return 0;
}
```

Answer : ----------

2. What is the output?

```
#include<iostream>
using namespace std;
namespace alpha{int var=1;}
namespace beta{
        Int var=alpha::var+1;
}
int main(){
        Beta::var+=alpha::var;
        {
                using namespace beta;
                cout<<var;
}
}
```

Answer : ----------

3. What is the output of this code?

```
int i=2;
for(int x=0;x<3;x++){
        I=i*2;
}
cout<<i;
```

Answer: _____

4. What is the output?

```
char *p=”Abcd”;
int len=0;
while(len<*p++)
        len++;
cout<<len;
```

Answer:_____

5. What is the output of this code?

```
void func(int &x){
        x=6;
}
int main(){
        int x=3;
        func(x);
        cout<<x;
}
```

Answer:_____

6. What is the output of this code?

```
Int a=5;
Int &p=a;
p+=3;  cout<<a;
```

Answer: _____

True False

7. What is the output of this code?

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

class A{
        public:
        static int cnt;
        A(int a){cnt++; }
        };
int A::cnt=0;
int main(){
        vector<A> v(4,1);
```

```
        v.push_back(1);
        cout<<A::cnt<<endl;
        return 0;
}
```
Answer: _____

8. What is the output of this code?
```
class A{
        int a;
        public :
                A(void) {a=1;}
                int b(void) {return ++a;}
};
int main(){
        A a;
        a.b();
        cout<<a.b();
}
```
Answer: _____

9. What is the output of this code?
```
int *t[2]={new int[2], new int[2]};
for(int i=0;i<4;i++)
        t[i%2][i/2]=I;
cout<<t[0][1]+t[1][0];
delete [] t[0];
delete [] t[1];
```
Answer: _____

10. What is the output this code?
```
char c[]={'a','b','c','f'};
++c[2];
c[2]+=c[1]-c[0];
for(unsigned i=0;i<4;i++)
        cout<<c[i];
```
Answer: _____

11. What is the output of this code?
```
class A{
```

```
        public:
        A()  {cout<<1;}
};
class B:A{
        public:
        B()   {cout<<2;}
};

Int main(){
        B b;
}
```

Answer : _____

12. What is the output of this code?

```
class A{
        public:
        A(){cout<<1;}
};
class B:A{
        public:
        B(){cout<<2;}
};
class C:B{
        public:
        C(){cout<<3;}
};
int main(){
        C c;
}
```

Answer: _____

13. What is the output of this code?

```
int add(a,b){
return a+b;
}

int sub(a,b){

return a-b;
}

int main(){
```

```
int x=99;
int y=22;
ans1=add(x,y);
ans2=sub(ans1,y);
cout<<ans2;


}
```

Answer : _____

14. What is the output of this code?

```
int x(int &a, int &b){
        a=3;
        b=4;
        return a+b;
}
int main(){
        int a=2;
        int b=7;
        int c=x(a,a);
        cout<<a<<b<<c;
}
```

Answer: _____

15. What is the output of this code?

```
int x=123;
int sum=0;
while(x>0){
        sum+=x%10;
        x=(x-x%10)/10;
}
cout<<sum;
```

Answer: _____

16. What is the output of this code?

```
int x=21, y=31, z;
Z=(x%5)*9/(y%6)*9;
cout<<z;
```

Answer: _____

17. What is the output of this code?

```
#include<iostream>
int foo(){
        return 10;
}
struct foobar{
        static int x;
        static int foo(){
                return 11;
}
};
int foobar :: x=foo();
int main(){
        std::cout<<foobar::xx;
}
```

Answer : _____

18. What is the output of this code?

```
bool CheckIt(int x){
        if(x==5)
                return false;
        else
                return true;
}
void incr(int &y)
        {  ++y;   }
int  main(){
        Int i=0;
        for(; CheckIt(i);incr(i));
        cout<<I;
}
```

Answer: _____

19. What is the output?

```
# include<iostream>
Using namespace std;
int main(){
        for(int i=0;i<7;i++){  }
                cout<<i;

}
```

20. What is the output?

```
int a=40;
int *ptr=&a;
a++;
cout<<*ptr;
```

Answer: _____

Part 4 (Project Coding)

Do the following POS (Point of Sale Program) on your own

Main MENU

Mini Project

Point of Sale Program (POS)

1] Product

2] Employee

3] Payment

4] Exit Program

Product MENU

1] Add Product

2] View Product

3] Update Product

4] Delete Product

5] Return to Main MENU

Employee MENU

1] Add Employee

2] View Employee

3] Update Employee Info

4] Delete Employee

5] Return to main MENU

Payment MENU

1] Sell Product

2] View Sold Product List

3] Delete Payment List

4] Return to Main MENU

Enter choice:

Suggested Coding Plans

Classes:

Product

Employee

Payment

POSApp

LinkedList:

Products

Employees

Payments

Files:

Products.txt

Employees.txt

Payments.txt

Product class

Attributes:

- Id
- Name
- Item
- Price

constructor
Setters/getters
toString

employee class

Attributes:

1. Id
2. Name
3. Position
4. Salary

constructor
Setters/getters
toString

payment class

Attributes:

1. Payment Id
2. Product ID
3. Employee ID
4. Sell Date
5. Product Name
6. Product Item
7. Product Price
8. total

constructor
Setters/getters
toString

မှတ်ချက်

၁။ sell product ပြုလုပ်ရာတွင် product id နှင့် employee id ရှိမရှိကို ဦးစွာ စစ်ဆေးရန်။ product id နှင့် employee id ရှိမှ သာ sell product function ကို ဆက်လက် လုပ်ဆောင်ရန်ဖြစ်သည်။

၂။ Project Source Code အား CD အချပ်ဖြင့် burn ၍ Northern City သို့ပြန်လည် ပေးအပ်ရန်

၃။ အထူးကျေးဇူးတင်ရှိပါသည်။ ကျန်းမာချမ်းသာကြပါစေ။ ☺

=Thank you=

(END OF CPlus Plus Course)

Whatever you do
Work at it with all your heart
As working for the Lord
Not for human masters
Colossians 3:23