Object Oriented & Programming Language

Lab#01

Table of Contents

Functions:

Functions are like building blocks. They let you divide complicated programs into manageable pieces. They have other advantages, too:

1. While working on one function, you can focus on just that part of the program and construct it, debug it, and perfect it.
2. Different people can work on different functions simultaneously.
3. If a function is needed in more than one place in a program or in different programs, you can write it once and use it many times.
4. Using functions greatly enhances the program's readability because it reduces the complexity of the function main

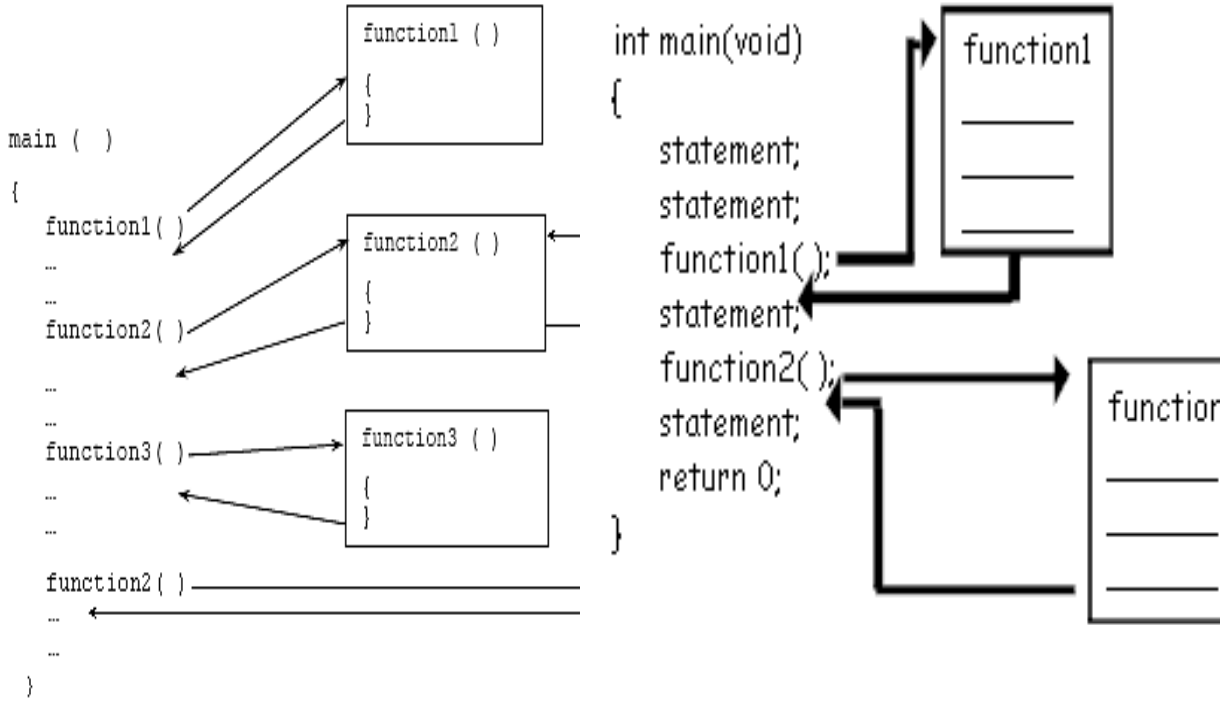

Figure 1: Functions

```
#include <iostream>
#include <cctype>
#include <iomanip>
using namespace std;
double getNumber();
char identifyInput();
double getKilos (double);
double getMeters (double);
int main()
{
```

```
double numberIn = 0.0, kilos = 0.0, meters = 0.0, answer = 0.0 ;
char choice;
numberIn = getNumber();
choice = identifyInput();
if (choice =='P')
{
answer = getKilos (numberIn);
cout << numberIn << " pounds are equivalent to " << answer << "
Kilograms.\n";
}
else
{
answer = getMeters (numberIn);
cout << numberIn << " yards are equivalent to " << answer << "
Meters.\n";
}
system("pause");
return 0;
}
double getNumber()
{
double input;
do
{
cout << "Enter a number greater than 0 that represents a measurement
in either pounds or yards: ";
cin >> input;
}while (input <= 0);
return input;
}
char identifyInput()
{
char option;
do
{
cout << "Enter a P if the number you input was pounds and Y if the
number input was yards: ";
cin >> option;
option = toupper(option);
}while(!(option == 'P' || option == 'Y'));
return option;
}
double getKilos (double lbs)
{
double result;
result = lbs * 0.45359237;
```

```
return result;
}
double getMeters ( double yds)
{
double result;
result = yds * 0.9144;
return result;
}
```

Output

```
/* Sample Run-I:
Enter a number greater than 0 that represents a measurement in either pounds or yards: 512
Enter a P if the number you input was pounds and Y if the number input was yards: P
512.00 pounds are equivalent to 232.24 Kilograms.

Sample Run-II:
Enter a number greater than 0 that represents a measurement in either pounds or yards: 512
Enter a P if the number you input was pounds and Y if the number input was yards: Y
512.00 yards are equivalent to 468.17 Meters.
Press any key to continue . . .
*/
```

Functions Overloading

Creating several functions with the same name but different formal parameters.
Two functions are said to have different formal parameter lists if both functions have:

- A different number of formal parameters or
- If the number of formal parameters is the same, then the data type of the formal parameters, in the order you list them, must differ in at least one position.

If a function's name is overloaded, then all of the functions in the set have the same name. Therefore, all of the functions in the set have different signatures if they have different formal parameter lists. Thus, the following function headings correctly overload the function functionXYZ:

```
void functionXYZ()
void functionXYZ(int x, double y)
void functionXYZ(double one, int y)
void functionXYZ(int x, double y, char ch)
```

Consider the following function headings to overload the function functionABC:

```
void functionABC(int x, double y)
int functionABC(int x, double y)
```

Both of these function headings have the same name and same formal parameter list. Therefore, these function headings to overload the function functionABC are incorrect. In this case, the compiler will generate a syntax error.

```
#include <iostream>
#include <conio.h>
using namespace std;
void repchar();
void repchar(char);
void repchar(char, int);
void main()
{
repchar();
repchar('=');
repchar('+', 30);
system("pause");
}
void repchar()
{
for(int j=0; j<45; j++)
cout << '*';
cout << endl;
}
void repchar(char ch)
{
for(int j=0; j<45; j++)
cout << ch;
cout << endl;
}
void repchar(char ch, int n)
{
for(int j=0; j<n; j++)
cout << ch;
cout << endl;
}
```

Output

```
*********************************************
=============================================
++++++++++++++++++++++++++++++
```

Functions with Default Parameters

```
#include <iostream>
#include <conio.h>
using namespace std;
void repchar(char='*', int=45);
int main()
{
repchar();
repchar('=');
repchar('+', 30);
system("pause");
}
void repchar(char ch, int n)
{
for(int j=0; j<n; j++)
cout << ch;
cout << endl;
}
```

Output

```
*********************************************
=============================================
++++++++++++++++++++++++++++++
Press any key to continue . . .
```

Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type, specify the name of the array followed by **square brackets** and specify the number of elements it should store:

1D Arrays

To create an array of five integers, you could write:

```
int myNum[5] = {10, 20, 30, 40, 50};
```

2D Arrays

The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x,y, you would write something as follows –

```
type arrayName [ r ][ c ];
```

Where **type** can be any valid C++ data type and **arrayName** will be a valid C++ identifier.

A two-dimensional array can be think as a table, which will have r number of rows and c number of columns. A 2-dimensional array **a**, which contains three rows and four columns can be shown as below −

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in array a is identified by an element name of the form **a[r][c]**, where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

Arrays Operations

Initialize

How to initialize the list elements with default values to avoid strange results.
void inilitiaze_array(int[]); // performs this needful job

Insert At

Remove At

Create Vacancy

Empty Test

```
#include<iostream>
#include <conio.h>
using namespace std;
const int array_size=15;
int length = 0;
```

```
void inilitiaze_array(int[]);
void print_array(int[]);
bool isFull();
bool isEmpty();
void insert(int[]);
void insertAt(int[]);
void remove(int[]);
void removeAt(int []);
void creatVacancy(int [], int );
void main()
{
int a[array_size];
inilitiaze_array(a);
print_array(a);
insert(a);
print_array(a);
remove(a);
print_array(a);
insertAt(a);
print_array(a);
removeAt(a);
print_array(a);
cout<<endl;
system("PAUSE");
}
//To inialitize the array by 1/3 of total.
void inilitiaze_array(int temp[])
{
for(int i=0; i<array_size/3; i++)
{
insert(temp);
}
}
// To print the whole array
void print_array(int temp[])
{
cout<<"\n-------------------------List-------------------------
\n";
for(int i=0; i<length; i++)
{
cout<<temp[i]<<", ";
}
```

```
cout<<"\n\n\t\t[List Length = "<<length<<" & List Total Size :
"<<array_size<<"]";
cout<<"\n_____________________________________________________
\n\n";
}
//It checks either array is full or not
bool isFull()
{
if(array_size==length)
{
return true;
}
return false;
//return (array_size==length);
}
// To check either list is empty or not.
bool isEmpty()
{
return (length==0);
}
//To insert at last of array
void insert(int test[])
{
if(isFull()!= true)
{
cout<<"Enter any integer value:> ";
//length++;
cin>>test[length];
length++;
}
else
{
cout<<"List is already full";
}
}
// To ceate vacancy at intended index...
void creatVacancy(int test[], int at)
{
for(int i=length-1;i>=at;i--)
{
test[i+1] = test[i];
}
```

```
}
// To insert at intended location
void insertAt(int test[])
{
if(isFull()!= true)
{
int at,value;
cout<<"Enter Location for entry:> ";
cin>>at;
cout<<"Enter some integer value:> ";
cin>>value;
creatVacancy(test,at-1);
test[at-1]=value;
length++;
}
else
{
cout<<"List is already full";
}
}
// To remove from last of list
void remove(int test[])
{
if(!isEmpty())
{
test[--length]=-1;
cout<<"\n One Last Emlement is removed from the list...\n";
}
else
{
cout<<"List is empty. So, NO element to remove from the list...";
}
}
// To remove from intended location from list
void removeAt(int test[])
{
if(!isEmpty())
{
int at;
cout<<"Enter Location to remove element: ";
cin>>at;
for(int i=at;i<length;i++)
```

```
{
test[i-1] = test[i];
}
length--;
}
else
{
cout<<"List is empty";

}
}
```

Output:

Run and see result

Searching in Array

A list is a set of values of the same type. Because all values are of the same type, lists can be stored in arrays. Because lists may be of varying sizes, the size of the array should be declared as the maximum size of the list. You can perform the following basic operations on a list:

- Input a list.
- Output a list.
- Search the list for a given item.
- Sort the list.
- Insert an item in the list.
- Delete an item from the list.

To search a list, you need to know the length of the list and the item for which you are searching. After searching a list, you need to know whether the item was found. If the item was found, you need to know where it was found.

Searching each location of a list until an item is found is called a sequential search or a binary search.

```
#include <iostream>
using namespace std;
int seqSearch(const int list[], int searchItem);
void printArray(int[]);
```

```
const int SIZE=15;
void main()
{
int list[SIZE] = {3,4,2,7,8,0,2,6,8,1,4,8,9,99,7};
int key;
cout<<"List elements are: ";
printArray(list);
cout<<"\n Enter an integer to search in the list :";
cin>> key;
int response;
response = seqSearch(list,key);
if(response>-1)
{
cout<<"Key found at index "<<response<<endl;
}
else
{
Cout<<"Key not found."<<endl;
}
system("PAUSE");
}
int seqSearch(const int list[], int searchItem)
{
int loc;
bool found = false;
loc = 0;
while (loc < SIZE && !found)
if (list[loc] == searchItem)
found = true;
else
loc++;
if (found)
return loc;
else
return -1;
}
void printArray(int list[])
{
cout<<endl;
for(int i = 0; i< SIZE; i++)
{
cout<<list[i]<<", ";
}
cout<<endl;
}
```

Output

```
List elements are:
3, 4, 2, 7, 8, 0, 2, 6, 8, 1, 4, 8, 9, 99, 7,

Enter an integer to search in the list :99
Key found at index 13Press any key to continue . . .
```

Counting Duplication of a Number

Modify the above given program to find the duplication of every searched key.

```
int repeatCount(const int list[], int searchItem)
{
int count=0;
bool found = false;
int loc = 0;
while (loc < SIZE)
{
if (list[loc] == searchItem)
{
found = true;
count++;
}
loc++;
}
if (found)
     return count;
else
     return -1;
}
```

Complete listing of program:

```
#include <iostream>
using namespace std;
int seqSearch(const int list[], int searchItem);
int repeatCount(const int list[], int searchItem);
void printArray(int[]);
const int SIZE=15;
void main()
{
int list[SIZE] = {3,4,2,7,8,5,2,6,8,1,4,8,9,99,7};
```

```
int key;
cout<<"List elements are: ";
printArray(list);
cout<<"\n Enter an integer to search in the list (-1 to exit):";
cin>> key;
do
{
     int response;
     response = seqSearch(list,key);
     if(response>-1)
     {
          cout<<"\tKey found at index "<<response<<" & repeated
"<<repeatCount(list,key)<<" times";
     }
     else if(response == -1)
     {
          cout<<"\tKey not found.";
     }
     cout<<"\n\n Enter an integer to search in the list (-1 to
exit):";
     cin>> key;
     cout<<endl;
}while(key != -1);
cout<<"Exiting the processing...\n";
system("PAUSE");
}
int seqSearch(const int list[], int searchItem)
{
     int loc;
     bool found = false;
     loc = 0;
     while (loc < SIZE && !found)
     {
          if (list[loc] == searchItem)
               found = true;
          else
               loc++;
     }
if (found)
     return loc;
else
     return -1;
```

```
}

int repeatCount(const int list[], int searchItem)
{
int count=0;
bool found = false;
int loc = 0;
while (loc < SIZE)
{
     if (list[loc] == searchItem)
     {
          found = true;
          count++;
}
loc++;
}
if (found)
     return count;
else
     return -1;
}
void printArray(int list[])
{
cout<<endl;
for(int i = 0; i< SIZE; i++)
{
     cout<<list[i]<<", ";
}
cout<<endl;
}
```

Output

```
List elements are:
3, 4, 2, 7, 8, 5, 2, 6, 8, 1, 4, 8, 9, 99, 7,
Enter an integer to search in the list (-1 to exit):5
Key found at index 5 & repeated 1 times
Enter an integer to search in the list (-1 to exit):3
Key found at index 0 & repeated 1 times
Enter an integer to search in the list (-1 to exit):0
Key not found.
```

```
Enter an integer to search in the list (-1 to exit):2
Key found at index 2 & repeated 2 times
Enter an integer to search in the list (-1 to exit):-1
Exiting the processing...
Press any key to continue . . .
*/
```

Searching in Two-Dimensional Array

Modify he given program to search in tow dimensional array of 7X5.

```
#include<iostream>
using namespace std;
const int COLS=5, ROWS=7;
void init(int[][COLS]);
void print(int[][COLS]);
bool search(int temp[][COLS], int searchItem, int&, int&);
void main()
{
int mat[ROWS][COLS];
init(mat);
print(mat);
int key;
cout<<"Enter an integer to search : ";
cin>>key;
int row = 0,col = 0;
if(search(mat,key,row,col))
{
     cout<<" \n Element found @ Row: "<<row<< " Col: "<< col;
}
else
{
     cout<<" \nElement not found";
}
cout<<endl;
system("PAUSE");
}
void init(int temp[][COLS])
{
     for(int i=0; i<ROWS; i++)
     {
          for(int j = 0; j< COLS; j++ )
```

```
            {
                  temp[i][j] = (i+1)*(j+2*i)*i;
            }
      }
}
void print(int temp[][COLS])
{
      for(int i=0; i<ROWS; i++)
      {
            for(int j = 0; j< COLS; j++ )
            {
                  cout<<"\t"<<temp[i][j];
            }
            cout<<endl;
      }
}
bool search(int temp[][COLS], int searchItem, int &row, int &col)
{
      bool found=false;
      for(int i=0; i<ROWS; i++)
      {
            for(int j = 0; j< COLS; j++ )
            {
                  if(temp[i][j] == searchItem)
                  {
                        found = true;
                        row = i + 1;
                        col = j + 1;
                  }
            }
      }
return found;
}
```