National University of Computer and Emerging Sciences

Laboratory Manual
for
Computer Programming

Course Instructor	Mr. Sarim Baig
Lab Instructor(s)	Zara Asif, Ahsan Rehman
Section	E
Semester	Spring 2017

Department of Computer Science
FAST-NU, Lahore, Pakistan

Pointers

Question 1:

Introduce int variables x and y and int* pointer variables p and q. Set x to 2, y to 8, p to the address of x, and q to the address of y. Then print the following information:
(1) The address of x and the value of x.
(2) The value of p and the value of *p.
(3) The address of y and the value of y.
(4) The value of q and the value of *q.
(5) The address of p (not its contents!).
(6) The address of q (not its contents!).

Question 2:

Declare three integers x, y and sum and three pointers xPtr, yPtr, sumPtr. Point three pointers to their respective variable.

- Take input in x and y using xPtr and yPtr. **Do not use direct references i.e. x and y integers**
- Add x and y and save the result in sum. **Do not use direct references i.e. x, y and sum integers**
- Print addition's result. For example, if user entered x = 5 and y = 9 your program should print: 5 + 9 = 14. **Do not use direct references i.e. x, y and sum integers**

Note: You have to do all the processing using pointers i.e. indirect references to variables.

Question 3:

Introduce int variables x, y, z and int* pointer variables p, q, r. Set x, y, z to three distinct values. Set p, q, r to the addresses of x, y, z respectively.
(1) Print with labels the values of x, y, z, p, q, r, *p, *q, *r.
(2) Print the message: Swapping values.
(3) Execute the swap code: z = x; x = y; y = z;
(4) Print with labels the values of x, y, z, p, q, r, *p, *q, *r.

Question 4:

Introduce int variables x, y, z and int* pointer variables p, q, r. Set x, y, z to three distinct values. Set p, q, r to the addresses of x, y, z respectively.
(1) Print with labels the values of x, y, z, p, q, r, *p, *q, *r.
(2) Print the message: Swapping pointers.
(3) Execute the swap code: r = p; p = q; q = r;
(4) Print with labels the values of x, y, z, p, q, r, *p, *q, *r.

Structure

Structure is a collection of variables of different types under a single name.

For example: You want to store some information about a person: his/her name, citizenship number and salary. You can easily create different variables name, citNo, salary to store these information separately.

However, in the future, you would want to store information about multiple persons. Now, you'd need to create different variables for each information per person: name1, citNo1, salary1, name2, citNo2, salary2

You can easily visualize how big and messy the code would look. Also, since no relation between the variables (information) would exist, it's going to be a daunting task.

A better approach will be to have a collection of all related information under a single name Person, and use it for every person. Now, the code looks much cleaner, readable and efficient as well.

This collection of all related information under a single name Person is a structure.

Structure Definition

Keyword **struct** is used for creating a structure.

Syntax of structure

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type member;
};
```

We can create the structure for a person as mentioned above as:

```
struct person
{
    char name[50];
    int citNo;
    float salary;
};
```

Structure variable declaration

When a structure is defined, it creates a user-defined type but, no storage or memory is allocated.
For the above structure of a person, variable can be declared as:

```
struct person
{
    char name[50];
    int citNo;
    float salary;
};

int main()
{
    struct person person1, person2, person3[20];
    return 0;
}
```

Another way of creating a structure variable is:

```
struct person
{
    char name[50];
    int citNo;
    float salary;
} person1, person2, person3[20];
```

In both cases, two variables person1, person2 and an array person3 having 20 elements of type struct person are created.

Accessing members of a structure

There are two types of operators used for accessing members of a structure.

```
Member operator (.)
Structure pointer operator (->)
```

Any member of a structure can be accessed as:

```
structure_variable_name.member_name
```

Suppose, we want to access salary for variable person2. Then, it can be accessed as:

```
person2.salary
```

Example

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

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

void printmovie (movies_t movie);

int main ()
{
  mine.title = "2001 A Space Odyssey";
  mine.year = 1968;

  cout << "My favorite movie is:\n ";
  printmovie (mine);

  return 0;
}

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

Statement :

Points and Rectangle

The following structs and functions illustrate the functionality that you will find in the graphics libraries for your system. Many of the basic graphics functions will use points and rectangles; for example, a line drawing function might take a "start point" and an "end point" as its parameters.

The graphics packages typically use short integers to represent coordinate values. A "point" will need two short integer data fields:

```
struct Point {
short fx;
short fy;
};
```

A struct to represent a rectangle can be defined as:

```
struct Rectangle {
short ftop;
short fleft;
short fwidth;
short fheight;
};
```

Part 1:
Write a function that returns "true" (1) if points p1 and p2 are the same.

```
int Equal_Points(Point p1, Point p2);
```

Part 2:
Write a function that changes the "output parameter" res to be the 'vector sum' of points p1 and p2.

```
void Add_Point(Point p1, Point p2, Point& res);
```

Part 3:
Write a function that returns the mid point of the given points.

```
Point MidPoint(Point p1, Point p2);
```

Part 4:
Write a function that return "true" if r's width and height are both zero.

```
int ZeroRect(Rectangle r);
```

Part 5:
Write a function that returns "true" if point “pt” (passed as parameter) lies within Rectangle “r”.

```
int Point_In_Rect(Point pt, Rectangle  r);
```