SSBT's Art, Commerce & Science College
Bambhori, Jalgaon

Bachelor of Computer Applications

LAB MANUAL

Class: FY BCA **Semester: II**

Subject: Lab on C Programming-II

Academic Year: 2022-2023

Name of Faculty: Mr. Krunal C. Pawar.

SSBT's Art, Commerce and Science College Bambhori, Jalgaon Bachelor of Computer Applications

Practical: 01

DOP: **DOC:**

Title: Write a program to illustrate concept of function using call by value.

Theory:

Call by Value

We can invoke a function in two ways which are call by value and call by reference. These two ways are usually identified by the kind of values that pass to them as parameters. Thus, the parameters which pass to function are known as actual parameters. On the other hand, the parameters that function receives are known as formal parameters.

Call By Value Function

In this parameter passing method, the values of actual parameters get copied to the function's formal parameters and the two kinds of parameters store in various memory locations. Thus, any changes which get made inside functions do not reflect in actual parameters of the caller.

In other words, the changes which happen to the parameter inside the function have no impact on the argument. By default, the C++ programming language makes use of call by value for passing arguments.

Above all, it means that code within a function cannot alter the argument which comes in use for calling the function. You might want to take a look at the function swap () definition given below:

C program to illustrate

```
// call by value
#include <stdio.h>
// Function Prototype
void swapx(int x, int y);
// Main function
int main()
{
    int a = 10, b = 20;
    // Pass by Values
    swapx(a, b);
    printf("a=%d b=%d\n", a, b);
    return 0;
}
// Swap functions that swaps
// two values
void swapx(int x, int y)
{
    int t;
    t = x;
    x = y;
    y = t;
    printf("x=%d y=%d\n", x, y);
}
```

Output:

x=20 y=10

a=10 b=20

Submitted By : **Checked By :**

Sign :

Name : **Mr. Krunal C. Pawar**

Roll No :

SSBT's Art, Commerce and Science College Bambhori,

Jalgaon Bachelor of Computer Applications

Practical: 02

DOP: **DOC:**

Title: Write a program to illustrate concept of function using call by reference.

Theory:

What is Function Call By Reference?

When we call a function by passing the addresses of actual parameters then this way of calling the function is known as call by reference. In call by reference, the operation performed on formal parameters, affects the value of actual parameters because all the operations performed on the value stored in the address of actual parameters. It may sound confusing first but the following example would clear your doubts.

```
// C program to illustrate Call by Reference


#include <stdio.h>
// Function Prototype
void swapx(int*, int*);
// Main function
int main()
{
int a = 10, b = 20;
 // Pass reference
swapx(&a, &b);
printf("a=%d b=%d\n", a, b);
return 0;
}
// Function to swap two variables
// by references
void swapx(int* x, int* y)
{
        int t;
        t = *x;
         *x = *y;
         *y = t;
```

```
        printf("x=%d y=%d\n", *x, *y);
}
```

Output:

```
x=20 y=10
a=20 b=10
```

Questions:

1) **Write query to create record of 5 students.**
2) **Explain DBMS languages?**

Submitted By : **Checked By :**

Sign :

Name :

Roll No : **Mr. Krunal C. Pawar.**

SSBT's Art, Commerce and Science College Bambhori, Jalgaon Bachelor of Computer Applications

Practical: 03

DOP: **DOC:**

Title: Write a program to illustrate concept of recursion.

Theory:

Recursion

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.

Recursion may be a bit difficult to understand. The best way to figure out how it works is to experiment with it.

Syntax

```
void recursion() {
  recursion(); /* function calls itself */
}

int main() {
  recursion();
}
```

The algorithmic steps for implementing recursion in a function are as follows:

Step1 - Define a base case: Identify the simplest case for which the solution is known or trivial. This is the stopping condition for the recursion, as it prevents the function from infinitely calling itself.

Step2 - Define a recursive case: Define the problem in terms of smaller subproblems. Break the problem down into smaller versions of itself, and call the function recursively to solve each sub problem.

Step3 - Ensure the recursion terminates: Make sure that the recursive function eventually reaches the base case, and does not enter an infinite loop.

step4 - Combine the solutions: Combine the solutions of the subproblems to solve the original problem.

Program

```
#include<stdio.h>
int fact(int);
int main()
{
   int x,n;
   printf(" Enter the Number to Find Factorial :");
   scanf("%d",&n);
   x=fact(n);
   printf(" Factorial of %d is %d",n,x);
   return 0;
}
int fact(int n)
{
   if(n==0)
      return(1);
   return(n*fact(n-1));
}
```

Output

```
Enter the Number to Find Factorial: 5
Factorial of 5 is 120
```

Submitted By : **Checked By :**

Sign :

Name :

Roll No : **Mr. Krunal C. Pawar.**

SSBT's Art, Commerce and Science College Bambhori, Jalgaon
Bachelor of Computer Applications

Practical: 04

DOP: **DOC:**

Title: Write a program to extern, static variable

Theory:

A storage class defines the scope (visibility) and life-time of variables and/or functions within a C Program. They precede the type that they modify. We have four different storage classes in a C program −

- auto
- register
- static
- extern

Static Storage class

The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

In C programming, when static is used on a global variable, it causes only one copy of that member to be shared by all the objects of its class.

```
#include <stdio.h>

/* function declaration */
void func(void);
static int count = 5; /* global variable */
main()
{
  while(count--) {
    func();
 }
  return 0;
}

/* function definition */
void func( void )
```

```
{
  static int i = 5; /* local static variable */
  i++;
  printf("i is %d and count is %d\n", i, count);
}
```

Output

```
i is 6 and count is 4
i is 7 and count is 3
i is 8 and count is 2
i is 9 and count is 1
i is 10 and count is 0
```

Extern storage class

The **extern** storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

When you have multiple files and you define a global variable or function, which will also be used in other files, then *extern* will be used in another file to provide the reference of defined variable or function. Just for understanding, *extern* is used to declare a global variable or function in another file.

The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions as explained below.

```
#include <stdio.h>

int count ;
extern void write_extern();
main()
{
  count = 5;
  write_extern();
}
```

Output

```
count is 5
```

Submitted By : **Checked By :**

Sign :

Name :

Roll No : **Mr. Krunal C. Pawar.**

SSBT's Art, Commerce and Science College Bambhori, Jalgaon
Bachelor of Computer Applications

Practical: 05

DOP: **DOC:**

Title: Write a program to demonstrate pointer to array.

Theory

Pointers are variables which stores the address of another variable. When we allocate memory to a variable, pointer points to the address of the variable. Unary operator (*) is used to declare a variable and it returns the address of the allocated memory. Pointers to an array points the address of memory block of an array variable.

The following is the syntax of array pointers.
datatype *variable_name[size];

Here,
datatype − The datatype of variable like int, char, float etc.
variable_name − This is the name of variable given by user.
size − The size of array variable.
The following is an example of array pointers.

```
#include <stdio.h>
int main () {
   int *arr[3];
   int *a;
   printf( "Value of array pointer variable : %d
", arr);
   printf( "Value of pointer variable : %d
", &a);
   return 0;
}
```

Output
Value of array pointer variable : 1481173888
Value of pointer variable : 1481173880

In the above program, an array pointer *arr and an integer *a are declared.

```
int *arr[3];
int *a;
```

The addresses of these pointers are printed as follows −

```
printf( "Value of array pointer variable : %d
", arr);
printf( "Value of pointer variable : %d
", &a);
```

Submitted By :	**Checked By :**
Sign :	
Name :	
Roll No :	**Mr. Krunal C. Pawar.**

SSBT's Art, Commerce and Science College Bambhori, JalgaonBachelor of Computer Applications

Practical: 06

DOP: **DOC:**

Title: Write a program to demonstrate pointer to function.

Theory

C Function Pointer

As we know that we can create a pointer of any data type such as int, char, float, we can also create a pointer pointing to a function. The code of a function always resides in memory, which means that the function has some address. We can get the address of memory by using the function pointer.

Declaration of a function pointer

Till now, we have seen that the functions have addresses, so we can create pointers that can contain these addresses, and hence can point them.

Syntax of function pointer

1. return type (*ptr_name)(type1, type2…);

```
#include <stdio.h>

void test()
{
   // test function that does nothing
   return ;
}
int main()
{
   int a = 5;
   // printing the address of variable a
   printf("Address of variable = %p\n", &a);
   // printing the address of function main()
   printf("Address of a function = %p", test);
   return 0;
}
```

Output

Address of variable = 0x7ffd7f36a224
Address of a function = 0x55f8e8abb169

Submitted By : **Checked By :**

Sign :

Name :

Roll No : **Mr. Krunal C. Pawar**

SSBT's Art, Commerce and Science College Bambhori, Jalgaon
Bachelor of Computer Applications

Practical: 07

DOP: **DOC:**

Title: Write a program to pointer to pointer.

A pointer to a pointer is a form of multiple indirection, or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.

A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, the following declaration declares a pointer to a pointer of type int −

```
int **var;
```

When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example −

```
#include <stdio.h>
int main ()
{
   int  var;
   int  *ptr;
   int  **pptr;
   var = 3000;
   /* take the address of var */
   ptr = &var;
   /* take the address of ptr using address of operator & */
   pptr = &ptr;
   /* take the value using pptr */
   printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n", **pptr);
   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000

Submitted By : **Checked By :**

Sign :

Name :

Roll No : **Mr. Krunal C. Pawar**

SSBT's Art, Commerce and Science College Bambhori, JalgaonBachelor of Computer Applications

Practical: 08

DOP: **DOC:**

Title: Write a program to demonstrate structure.

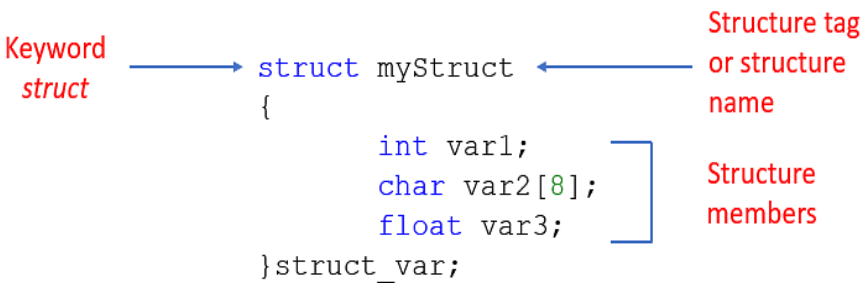
Theory

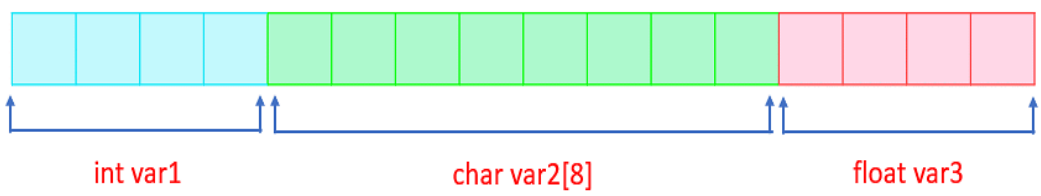
Arrays allow to define type of variables that can hold several data items of the same kind. Similarly structure is another user defined data type available in C that allows to combine data items of different kinds.

Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −

- Title
- Author
- Subject
- Book ID

At times, during programming, there is a need to store multiple logically related elements under one roof. For instance, an employee's details like name, employee number, and designation need to be stored together. In such cases, the C language provides structures to do the job for us.





Memory representation for struct_var

A structure can be defined as a single entity holding variables of different data types that are logically related to each other. All the data members inside a structure are accessible to the functions defined outside the structure. To access the data members in the main function, you need to create a structure variable.

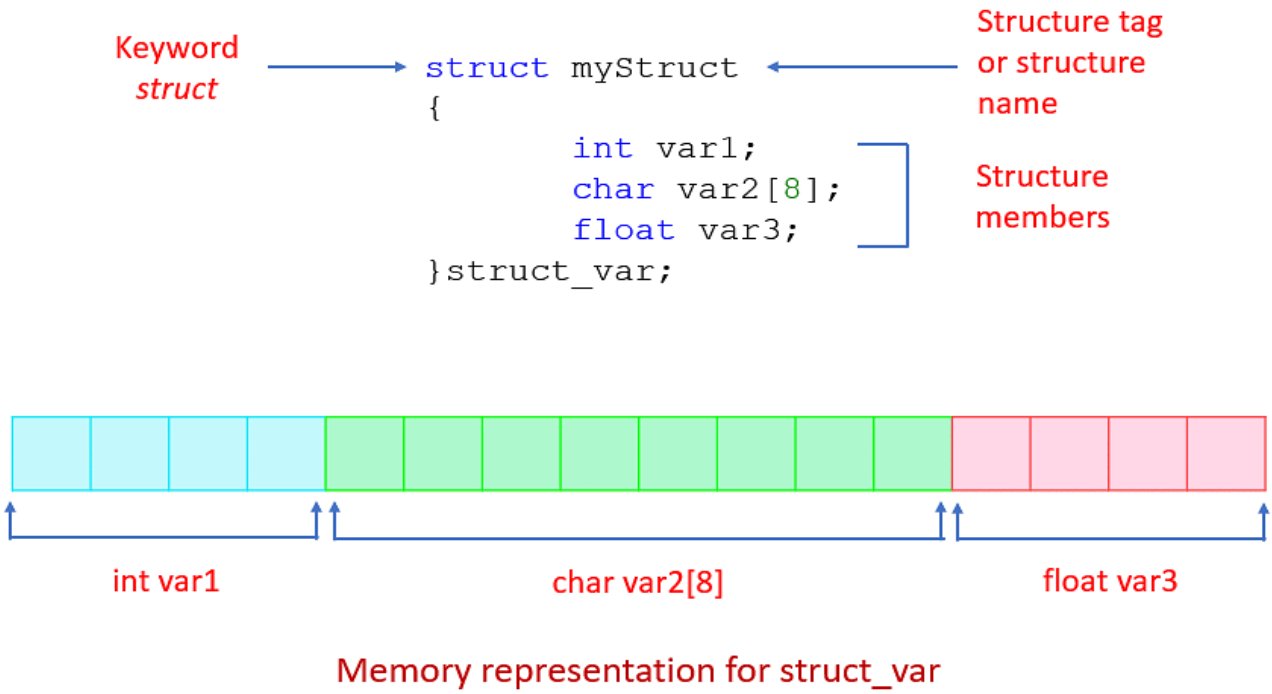
What Is Structures In C and How to Create It?

By Ravikiran A S

Last updated on Feb 15, 202359819

Table of Contents

At times, during programming, there is a need to store multiple logically related elements under one roof. For instance, an employee's details like name, employee number, and designation need to be stored together. In such cases, the C language provides structures to do the job for us.



Memory representation for struct_var

A structure can be defined as a single entity holding variables of different data types that are logically related to each other. All the data members inside a structure are accessible to the functions defined outside the structure. To access the data members in the main function, you need to create a structure variable.

Now, lets get started with Structure in C programming.

Why Do We Use Structures in C?

Structure in C programming is very helpful in cases where we need to store similar data of multiple entities. Let us understand the need for structures with a real-life example. Suppose you need to manage the record of books in a library. Now a book can have properties like book_name, author_name, and genre. So, for any book you need three variables to store its records. Now, there are two ways to achieve this goal.

The first and the naive one is to create separate variables for each book. But creating so many variables and assigning values to each of them is impractical.

So what would be the optimized and ideal approach? Here, comes the structure in the picture.

We can define a structure where we can declare the data members of different data types according to our needs. In this case, a structure named BOOK can be created having three members book_name, author_name, and genre. Multiple variables of the type

BOOK can be created such as book1, book2, and so on (each will have its own copy of the three members book_name, author_name, and genre).

Moving forward, let’s he a look at the syntax of Structure in C programming

Syntax to Define a Structure in C

```
struct structName
{
   // structure definition
   Data_type1 member_name1;
   Data_type2 member_name2;
   Data_type2 member_name2;
};
#include <stdio.h>
#include <string.h>

// create struct with person1 variable
struct Person {
  char name[50];
  int citNo;
  float salary;
} person1;

int main() {
```

```
  // assign value to name of person1
  strcpy(person1.name, "George Orwell");

  // assign values to other person1 variables
  person1.citNo = 1984;
  person1. salary = 2500;

  // print struct variables
  printf("Name: %s\n", person1.name);
  printf("Citizenship No.: %d\n", person1.citNo);
  printf("Salary: %.2f", person1.salary);

  return 0;
}
```

Output

Name: George Orwell

Citizenship No.: 1984

Salary: 2500.00

Submitted By : **Checked By :**

Sign :

Name : **Mr. Krunal C. Pawar**

Roll No :

SSBT's Art, Commerce and Science College Bambhori, Jalgaon
Bachelor of Computer Applications

Practical: 09

DOP: **DOC:**

Title: Write a program to demonstrate union.

Theory

A **union** is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

Defining a Union

To define a union, you must use the **union** statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows –

```
union [union tag] {
   member definition;
   member definition;
   ...
   member definition;
} [one or more union variables];
```

The **union tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the union's definition, before the final semicolon, you can specify one or more union variables but it is optional. Here is the way you would define a union type named Data having three members i, f, and str −

```
union Data {
   int i;
   float f;
   char str[20];
} data;
```

Now, a variable of **Data** type can store an integer, a floating-point number, or a string of characters. It means a single variable, i.e., same memory location, can be used to store multiple types of data. You can use any built-in or user defined data types inside a union based on your requirement.

The memory occupied by a union will be large enough to hold the largest member of the union. For example, in the above example, Data type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by a character string. The following example displays the total memory size occupied by the above union −

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

union Data
{
   int i;
   float f;
   char str[20];
};
int main( )
{
   union Data data;
   printf( "Memory size occupied by data : %d\n", sizeof(data));
   return 0;
}
```

Output

Memory size occupied by data : 20

Submitted By : **Checked By :**

Sign :

Name :

Roll No : **Mr. Krunal C. Pawar**

SSBT's Art, Commerce and Science College Bambhori, JalgaonBachelor of Computer Applications

Practical: 10

DOP: **DOC:**

Title: Write a program to demonstrate various graphics functions.
Theory

Graphics functions in C

here are so many built in graphics functions defined in C library. The basic graphics functions are described as below. Graphics functions are text mode graphics functions as well as graphical mode functions.

Text mode graphics functions: The text mode graphics functions place the text in certain area of the screen. Those functions are included in the header file <conio.h>. Some of those text mode graphics functions are:

window(): It sets particular area on the output screen and displays the output text on that area.

The syntax of calling this window function is

window(10,10, 80,25);

where co-ordinate (10,10) represents the upper left corner of displaying window and (80,25) is the lower right corner of the display window.

clrscr(): It is used to clear the screen and locates the curser at the beginning of the screen.

gotoxy(x,y): it moves the curser at specified co-ordinates (x,y) position.

cputs(string): it writes a string given to function to the user defined window screen.

putch(char) : it writes a character given to function to the user defined area of window.

Graphical mode functions:

There are so many graphical functions in C graphics library. All the graphics mode functions are in <graphics.h> file. So we must include the graphics.h file to use those functions. To run a c program in graphical mode the most important functions are as below.

initgraph(): it is one of the function that is used to initialize the computer in graphics mode. Its syntax is as

initgraph(&gdriver, &gmode, "graphics driver path");

closegraph(): It is the graphical function that is used to close the graphics mode initialized by the initgraph() function.

graphresult(): it is a graphical function that returns the value 0 if the graphics is detected correctly and the driver is correctly initialized to correct graphics mode other wise it returns some error code than 0.

grapherrormsg(errorcode): This function returns the message string corresponding to the errorcode returned by the graphresult() function.

cleardevice(): This function is used to clear the screen in graphical mode as clrscr() in text mode.

After correctly initialized in graphics mode, we can use the different graphical functions

available in the c- library in graphics.h. Following are some basic graphical functions for

drawing geometrical objects.

setcolor(color): It is a function that is used to set the color of the drawing object with given color. The color is indicated by the integer from 0 to 15 (16 color)

moveto(x, y): it is used to move the cursor in display screen at specified co-ordinates by the value (x,y).

outtext(" text string "): prints the text string in the current position of screen.

outtextxy(x, y, "string"): prints the string from the co-ordinate (x,y) position.

lineto(x, y): draws the line from current position to (x,y) position.

putpixel(x, y, color): displays a pixel(point) at (x,y) with given color.

line(a, b, c, d): draws line from (a,b) to (c,d).

circle(x, y, r): prints circle with center (x,y) and radius r.

rectangle(a, b, c, d): prints rectangle where (a,b) is upper left coordinate and (c,d) is lower right co-ordinates of rectangle.

Some Examples of C graphics:

```
// Drawing a Circle
#include
void main()
{
int gd= DETECT, gm;
initgraph(&gd,&gm,"c:\\tc\\bgi");
circle(200,100,10);
setcolor(WHITE);
getch();
closegraph();
}
```

```
//Drawing Lines
#include
void main()
{
int gd= DETECT, gm;
initgraph(&gd,&gm,"c:\\tc\\bgi");
line(90,70,60,100);
line(200,100,150,300);
setcolor(WHITE);
getch();
closegraph();
}
```

```
// Drawing rectangle
#include
void main()
{
int gd= DETECT, gm;
initgraph(&gd,&gm,"c:\\tc\\bgi");
rectangle(200,100,150,300);
setcolor(WHITE);
getch();
closegraph();
}
```

```
//Constructing triangle
```

```
#include
void main()
{
int gd= DETECT, gm;
initgraph(&gd,&gm,"c:\\tc\\bgi");
line(200,100,10,20);line(10,20,50,60);line(50,60,200,100);
setcolor(WHITE);
getch();
closegraph();

}
```

Submitted By : **Checked By :**

Sign :

Name :

Roll No : **Mr.Krunal C. Pawar**