FUNCTIONS AND INTRODUCTION TO POINTERS

Passing arguments – Call by Value

Designed by Parul Khurana, LIECA.

Passing Arguments to a Function – Call By Value

- When a single value is passed to a function via an actual argument, the value of the actual argument is **copied** into the function.
- *Therefore, the value of the corresponding formal argument can be altered within the function*, but the value of the actual argument within the calling routine will not change.
  - This procedure for passing the value of an argument to a function is known as **passing by value**.

Passing Arguments to a Function – Call By Value - Example

- Here is a simple C program containing a function that alters the value of its argument:

```
# include <stdio.h>
void modify ( int a );
void  main (void) {
int a = 2;
printf (“\na=%d ( from main, before calling the function)”,a);
modify(a);
printf (“\na=%d ( from main, after calling the function)”,a);
}
```

```
void modify ( int a)
{
a *= 3;
printf(“\n\na = %d ( from the function, after being modified )”, a);
return;
}
```

Passing Arguments to a Function – Call By Value - Example

- The original value of a (i.e., a = 2) is displayed when main begins execution.
- This value is then passed to the function modify, where it is multiplied by 3 and the **new value** will be displayed.

Passing Arguments to a Function – Call By Value - Example

- Note that it is the *altered value of the formal argument* that is displayed within the function.
- Finally, the value of a within main (i.e., the actual argument) is again displayed, after control is transferred back to the main from modify.

Passing Arguments to a Function – Call By Value - Example

- When the program is executed, the following output is generated:

a =2 (from main, before calling the function)
a =6 (from the function, after being modified)
a =2 (from main, after calling the function)

- These results show that a is not altered within the main, even though the corresponding value of a is changed within modify.

Passing Arguments to a Function – Call By Value

- Passing an argument by value has advantages and disadvantages.
- On the plus side, it allows a single – valued actual argument to be written as an expression rather than being restricted to a single variable.
- Moreover, if the actual argument is expected simply as a single variable, it protects the value of this variable from alterations within the function.

Passing Arguments to a Function – Call By Value

- On the other hand, it does not allow information to be transferred back to the calling portion of the program via arguments.

- Thus, passing by value is restricted to a one-way transfer of information.

Passing Arguments to a Function – Call By Value - Important

- **The following points must be noted about passing arguments using call by value mechanism:**

- The actual arguments can be constants, variables, or expressions.
- When the control is transferred from the calling function to the called function, the memory for formal arguments and local variables is **allocated**, *values of the actual arguments are substituted in the corresponding formal arguments*, and the statements in the function body are executed.

Passing Arguments to a Function – Call By Value - Important

- **The following points must be noted about passing arguments using call by value mechanism:**

- As soon as the called function finishes its execution, the **memory allocated for it is de-allocated**, i.e., the values of formal arguments and local variable are destroyed, and finally the control is transferred back to the calling function.
- Any change made to the formal arguments will have no effect on actual arguments, since the function will only be using the local copy of the arguments.

Practice Questions

- Predict the behavior of the following code segment:

```
void main(void)
{
        int s = sum(10,20);
        printf("%d", s);
}
void sum(int a, int b)
{
    int s;
    s = a + b;
}
```

Practice Questions

- Predict the result of the following code segment:

```
int x = 5;
void change()
{
    x = x + 5;
}
void main()
{
    printf("\nInitially: %d",x);
    change();
    printf("\nFinally: %d", x);
}
```

Practice Questions

- Predict the result of the following code segment:

```
void change(Int x)
{
    int y = x + 5;
    x = y * (x++);
}
void main()
{
    int y = 10;
    printf("\nInitially: %d",y);
    change(y);
    printf("\nFinally: %d", y);
}
```

Practice Questions

- Predict the result of the following code segment:

```
getnew(int x, float y, int z)
{
    x++;
    y = (x * z)/10;
    z – – ;
    return y;
}
void main()
{
    float a = getnew(3,4.5,6);
    int b = getnew(a+1, a, a-1);
    printf(“%f and %d”,a,b);
}
```

Practice Questions

- Write functions using call by value to:
  - The main() has 3 variables x, y, z with values 10, 20, 30 respectively. Define three functions chx(), chy(), chz() which tend to increase the value of variables x, y, z of main() by 5, 10, 15.
  - The main() has 3 variables x, y, z with values 10, 20, 30 respectively. Define a single function changeany() which tend to increase the value of variables x, y, z of main() by 5, 10, 15.