CSE101-Lec#14

Function Call

Outline

- Function call
  - Passing arguments by value
  - Passing arguments by reference

Formal Arguments and Actual Arguments

- **Argument:** An argument is an expression which is passed to a function by its caller in order for the function to perform its task.
- **Actual arguments:** The arguments that are passed in a **function call** are called actual arguments. These arguments are defined in the calling function.
- **Formal arguments:** The formal arguments are the parameters/arguments in a **function declaration**. Formal arguments are a copy of the actual arguments.

```c
#include <stdio.h>
void sum(int i, int j, int k); /*function prototype*/
int main()
{
int a = 5;
sum(3, 2 * a, a); // actual arguments
return 0;
}
void sum(int i, int j, int k)//formal arguments
{
int s;
s = i + j + k;
printf("sum is %d", s);
}
```

Methods of passing arguments

- There are two ways to call a function/to pass arguments to a function:

1. Call by value
2. Call by reference

Call by Value

- Call by value
  - In this method the values of actual arguments are copied to the formal arguments of the function.
  - Changes in function do not effect original
  - Use when function does not need to modify argument
    - Avoids accidental changes
  - The method of passing arguments by value is know as call by value

```
#include <stdio.h>
int callByValue(int n); // prototype / declaration
int main( void )
{
   int number = 5; // initialize number
   printf("The original value of number is %d", number);
   callByValue(number); // pass number by value
   printf( "\nThe new value of number is %d\n", number );
} // end main

int callByValue( int n )
{
   return n*n*n; //cube local variable n and return value
}
```

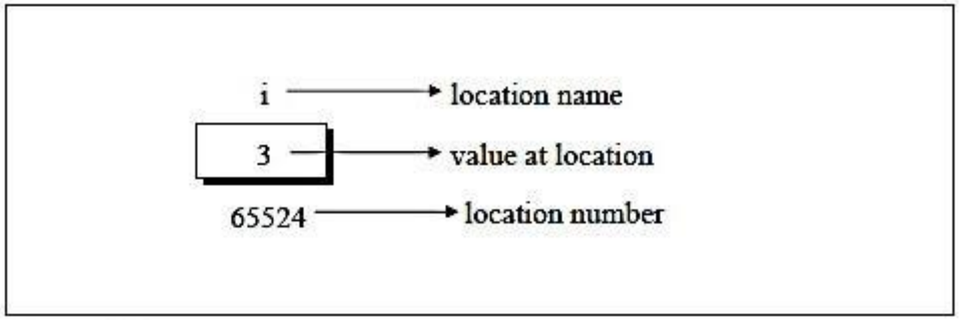
```
The original value of number is 5
The new value of number is 5
```

Call by reference

- The address of actual argument are copied to the formal arguments.
- The called function uses the address to refer to the actual location.
- Changes made by the function are effective when the control returns to the calling function.
  - If we want to make changes even in the actual arguments, then we use call by address

Address of variable

```
int i=3;
```



If we want to print the address of variable:

```
#include<stdio.h>
void main()
{
int i=3;
printf("address of i=%d", &i);
printf("value of i =%d", i);
}
```

Pointers

- A variable which stores address of another variable
- Example:

```
int *p;
int i;
p= &i;
```

- ➢ *p gives **value** at address stored in p.
- ➢ int *p means p is containing an address of variable on which an integer is stored

Calling Functions by Reference

- Call by reference with pointer arguments
  - Pass address of argument using & operator
  - Allows you to change actual location in memory
- * operator
  - Used as alias/nickname for variable inside of function

```
void double( int *number )
 {
  *number = 2 * ( *number );
 }
```

  - `*number` used as nickname for the variable passed

```
#include <stdio.h>
void callByReference(int *nPtr); //function prototype
int main( void )
{
   int number = 5;
    printf("The original value of number is %d", number);
   callByReference( &number ); //pass address of number
   printf("\nThe new value of number is %d\n", number);
} // end main

//calculate cube of *nPtr; actually modifies number in main
void callByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;   //cube *nPtr
}
```

```
The original value of number is 5
The new value of number is 125
```

Header Files-Review

- Header files
  - Contain function prototypes for library functions
  - `<stdlib.h>` , `<math.h>`, etc
  - Load with **`#include <filename>`**

    `#include <math.h>`
- Custom header files
  - Create file with functions
  - Save as `filename.h`
  - Load in other files with **#**`include "filename.h"`
  - Reuse functions
    - Example `#include<square.h>`

Header Files

Standard library header	Explanation
`<ctype.h>`	Contains function prototypes for functions that test characters for certain properties, and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa.
`<errno.h>`	Defines macros that are useful for reporting error conditions.
`<float.h>`	Contains the floating point size limits of the system.
`<limits.h>`	Contains the integral size limits of the system.
`<math.h>`	Contains function prototypes for math library functions.
`<stddef.h>`	Contains common definitions of types used by C for performing certain calculations.
`<stdio.h>`	Contains function prototypes for the standard input/output library functions, and information used by them.
`<stdlib.h>`	Contains function prototypes for conversions of numbers to text and text to numbers, memory allocation, random numbers, and other utility functions.
`<string.h>`	Contains function prototypes for string processing functions.

Next Class: Recursive Functions and Scope Rules

cse101@lpu.co.in