FUNCTIONS

SDN 150S – Week 4

Lecturer: Mr. Stephen Ekwe

Cape Peninsula University of Technology

creating futures

Outline

SCOPE RULES

HEADER FILES

FUNCTIONS

Scope Rules

- A scope in programming is a region of the program where a defined variable can exist (declared & called).
- However, beyond that region the variable cannot be accessed.
- There are three places where variables can be declared in C programming language:

Variable Type	Description
Local Variable	Inside a function or a block which is called local variables
Global Variable	Outside of all functions which is called global variables
Formal Parameter	In the definition of function parameters which are called formal parameters.

- Local Variables can only be used by statements within the same function or block of code.
- Local variables are not known to functions outside their own.

```
1. #include <stdio.h>
2. int main ()
3. {
4. /* local variable declaration */
5. int a, b;
6. int c;
7. /* actual initialization */
8. a = 10;
9. b = 20;
10.c = a + b;
11.printf ("value of a = %d,"
12."b = %d and c = %d\n", a, b, c);
13.return 0;
14.}
```

Declare Local Variable

- Global variables are declared outside a function, usually on top of the program, They hold their values throughout the lifetime of your program and can be accessed or used by any statement inside any of the functions defined in the program.

Global Variable (g)

```
1. #include <stdio.h>
2. /* global variable declaration */
3. int g;
4. int main ()
5. {
6. /* local variable declaration */
7. int a, b;
8. /* actual initialization */
9. a = 10;
10.b = 20;
11.g = a + b;
12.printf ("value of a = %d,"
13."b = %d and g = %d\n", a, b, g);
14.return 0;
15.}
```

```
1. #include <stdio.h>
2. /* global variable declaration
3. int g = 20;
4. int main ()
5. {
6. /* local variable declaration */
7. int g = 10;
8. printf ("value of g = %d\n", g)
9. return 0;
10. }
```

When a local and global variable have the same name, the local takes precedence in the function

- Formal parameters are treated as local variables, however, within a function they take precedence over global variables. See the example below:

```c
#include <stdio.h>
/* global variable declaration */
int a = 20;
/* function definition to sum values */
int sum(int a, int b);

int main ()
{
/* local variable declaration in main function */
int a = 10;
int b = 20;
int c = 0;
printf ("value of a in main() = %d\n", a);
c = sum(a, b);
printf ("value of c in main() = %d\n", c);
return 0;
}

/* function to add two integers */
int sum(int a, int b)
{
printf ("value of a in sum() = %d\n", a);
printf ("value of b in sum() = %d\n", b);
return a + b;
}
```

OUTPUT

value of a in main() = 10
value of a in sum() = 10
value of b in sum() = 20
value of c in main() = 30

- When a local variable is defined, it is not initialized by the system, you must initialize it yourself.
- Global variables are initialized automatically by the system when you define them, as follows:

Data Type	Initial Default Value
Int	0
Char	'\0'
float	0
double	0
pointer	NULL

Headers

- A header file is a file with extension .h which contains C function declarations and macro definitions, which is to be shared between several source files.
- There are two types of header files: user defined header files and the standard library header files that come with your compiler.
- Each standard library has a corresponding header containing the function prototypes for all the functions in that library, and the definitions of various data types and constants needed by those functions
- Both standard and user header files are included using the preprocessing directive `#include`, as shown below respectively:
  - `#include <file>` and `#include "file"`

- The following table alphabetically lists several standard library headers that may be included in programs

Headers we discuss	Explanation
<assert.h>	Contains information for adding diagnostics that aid program debugging.
<ctyipe.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.
<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.
<signal.h>	Contains function prototypes and macros to handle various conditions that may arise during program execution.

Headers we discuss	Explanation
<stdarg.h>	Defines macros for dealing with a list of arguments to a function whose number and types are unknown.
<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.
<time.h>	Contains function prototypes and types for manipulating the time and date.

Functions

- A function is a collection of statements grouped together to perform some specific task.
- The C standard library provides numerous built-in functions that your program can call.
- A function declaration tells the compiler about a function's name, return type, and parameters.
- A function definition provides the actual body of the function.
- A function can also be referred to as a method or sub-routine or procedure, etc.

Advantages of Functions

- Function support reusability of code: Functions once defined can be used several times. Meaning you can use the same function(s) in several unrelated programs, which saves time and effort.
- Functions provides abstraction: To use any function you only need its name and arguments. Its not necessary to know how it works internally.
- Function allows modular design of code: We can divide program into small modules. Modular programming leads to better code readability, maintenance and reusability.
- Functions simplify codes: You can write code for separate task individually in separate function.
- Functions allow for easy maintenance and debugging of code: In case of errors in a function, you only need to debug that particular function instead of debugging entire program.

Defining A Function

- The general form of a function definition in C programming language is as follows:

```
return_type function_name( parameter list )
{
   //body of the function
}
```

- A function definition in C programming consists of a function header and a function body. Here are all the parts of a function:
- Return Type: A function may return a value of a specific data type or may perform the desired operations without returning a value. In this case, the return_type is the keyword void.

- Function Name: This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- Parameter: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument.
- Parameter list: This refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- Function Body: The function body contains a collection of statements that define what the function does.

- Given below is a function called max().
- This function takes two parameters num1 and num2 and returns the maximum value between the two:

```
/* function returning the max between two numbers */
int max(int num1, int num2)
{
/* local variable declaration */
int result;
if (num1 > num2)
   result = num1;
else
   result = num2;
return result;
}
```

FUNCTION DECLARATIONS

- This tells the compiler about a function name and how to call the function.
  - `return_type function_name( parameter list );`
- For the max() function, the function declaration is as follows:
  - `int max(int num1, int num2); or int max(int, int);`
- Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

Calling a Function

- To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function.
- The called function then performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.
- To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

- For example:

```
1.  #include <stdio.h>
2.  /* function declaration */
3.  int max(int num1, int num2);
4.  int main ()
5.  {
6.  /* local variable definition */
7.  int a = 100;
8.  int b = 200;
9.  int ret;
10. /* calling a function to get max value */
11. ret = max(a, b);
12. printf( "Max value is : %d\n", ret );
13. return 0;
14. }
15. /* function returning the max between two numbers */
16. int max(int num1, int num2)
17. {
18. /* local variable declaration */
19. int result;
20. if (num1 > num2)
21. result = num1;
22. else
23. result = num2;
24. return result;
25. }
```

OUTPUT

Max value is : 200

Call by Value

- This is a method of passing arguments to a function, copying the actual value of an argument into the formal parameter of the function.
- In this case, changes made to the parameter inside the function have no effect on the argument.
- By default, C programming uses call by value to pass arguments.
- In general, it means the code within a function cannot alter the arguments used to call the function.

- Consider the function swap() definition as follows.

Declare & Call Swap Function

```
1.  #include <stdio.h>
2.  /* function declaration */
3.  void swap(int x, int y);
4.
    int main ()
5.  {
6.  /* local variable definition */
7.  int a = 100;
8.  int b = 200;
9.  printf("Before swap, value of a : %d\n", a );
10. printf("Before swap, value of b : %d\n", b );
11.
    /* calling a function to swap the values */
12. swap(a, b);
13. printf("After swap, value of a : %d\n", a );
14. printf("After swap, value of b : %d\n", b );
15. return 0;
16. }
```

Define Swap Function

```
1.  void swap(int x, int y)
2.  {
3.  int temp;
4.  temp = x; /* save the value of x */
5.  x = y; /* put y into x */
6.  y = temp; /* put temp into y */
7.  return;
8.  }
```

OUTPUT

Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :100
After swap, value of b :200

Return Statement

- The illustration shows two functions (sum and main).
- The sum with function type int passes two arguments (int a and int b) to perform some task and return some integer value to the calling function main, which is store in a variable.
- Note that the returned value, the variable, and the function type must have the same data type.

```
int sum(int a, int b)
{
    // some task
    return some_iterger_value;
}
int main ()
{
    int variable = sum(10,20);
    return 0;
}
```

***SAME* DATA TYPE**

```c
#include <stdio.h>
/* Function declaration */
int max(int num1, int num2);

int main()
{
    int num1, num2, maximum;
    printf("Enter two numbers: ");
    scanf("%d%d", &num1, &num2);
    /* Call max function with arguments num1 and num2
       Store the maximum returned to variable maximum */
    maximum = max(num1, num2);
    printf("Maximum = %d", maximum);
    return 0;
}

/* Function definition */
int max(int num1, int num2)
{
    int maximum;
    // Find maximum between two numbers
    if(num1 > num2)
        maximum = num1;
    else
        maximum = num2;
    // Return the maximum value to caller
    return maximum;
}
```

OUTPUT

Enter two numbers: 10 20

Maximum = 20

Type of Functions

- Functions in C programming is categorized as either Library functions or User-defined functions.

Library Functions:

- These are built-in functions that have been defined and included in the C complier.
- Common standard library functions are; printf(), scanf(), pow(), strlen(), etc.
- These functions are defined in C header files, which are included in a program as needed.

User Defined Functions:

- User defined functions can be compile as library function and used later in another program. They are placed into four categorizes, namely:

1. Function with no return and no argument:

- This type of function neither returns a value nor accepts any argument. It does not communicate with the caller function, since It works as an independent block.

- The return type of the function must be void if no expression is returned. The syntax is given below:

```
1. void function_name( )
2. {
3.     // body of the function
4. }
```

2. Function with no return but with arguments:

- These type of function does not return a value but accepts arguments as input. For this type of function you must also define function return type as void. The syntax is given below:

```
1. void function_name(type arg1, type arg2, ...)
2. {
3.     // body of the function
4. }
```

3. Function with return but no arguments:

- These type of function return a value but does not accept any arguments. This type of function communicates with the caller function by returning some value back to the caller. The syntax is given below:

```
1. return_type function_name( )
2. {
3.     // body of the function
4.     return some_value
5. }
```

4. Function with return and with arguments:

- This type of function returns a value and may accept arguments. Since the function accepts input and returns a result to the caller, hence this type of functions are most used and best for modular programming:

```
return_type function_name(type arg1, type arg2, ...)
{
    // body of the function
    return some_value
}
```

- Write a C function to generate Fibonacci series of any number. Your function must have no return value and no argument.

```c
#include <stdio.h>

/* Function declaration */
void generateFibo();
int main()
{
    generateFibo();
    return 0;
}
```

```c

/* Function definition */
void generateFibo()
{
    int a, b, c, i, terms;
    /* Input a number from user */
    printf("Enter number of terms: ");
    scanf("%d", &terms);
    a = 0;
    b = 1;
    c = 0;

    printf("Fibonacci terms: \n");
    // Iterate through n terms
    for(i=1; i<=terms; i++)
    {
        printf("%d, ", c);
        a = b;      // Copy n-1 to n-2
        b = c;      // Copy current to n-1
        c = a + b;  // New term
    }
}
```

OUTPUT

Enter number of terms: 12

Fibonacci terms:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89

- Write a C function to print all natural numbers between any input numbers. Your function must have no return value but with argument.

```c
#include <stdio.h>
/* Function declaration */
void printNaturalNumbers(int start, int end);

int main()
{
    int s, e;
    printf("Enter lower range to print natural numbers: ");
    scanf("%d", &s);
    printf("Enter upper limit to print natural numbers: ");
    scanf("%d", &e);
    printNaturalNumbers(s, e);
    return 0;
}

/* Function definition */
void printNaturalNumbers(int start, int end)
{
    printf("Natural numbers from %d to %d are: \n", start, end);
    while(start <= end)
    {
        printf("%d, ", start);
        start++;
    }
}
```

OUTPUT

Enter lower range to print natural numbers: 5
Enter upper limit to print natural numbers: 10
Natural numbers from 5 to 10 are:
5, 6, 7, 8, 9, 10,

- Write a C function that returns a random prime number on each call. Your function has a return but no argument.

```c
#include <stdio.h>
#include <stdlib.h> // Used for rand() function
/* Function declaration */
int randPrime();

int main()
{
    int i;
    printf("Random 5 prime numbers are: \n");
    for(i=1; i<=5; i++)
    {
        printf("%d\n", randPrime());
    }
    return 0;
}

/* Function definition */
int randPrime()
{
    int i, n, isPrime;
    isPrime = 0;
    while(!isPrime)
    {
        n = rand(); // Generates a random number

        /* Prime checking logic */
        isPrime = 1;
        for(i=2; i<=n/2; i++)
        {
            if(n%i==0)
            {
                isPrime = 0;
                break;
            }
        }
        if(isPrime ==1)
        {
            return n;
        }
    }
}
```

OUTPUT

Random 5 prime numbers are:
1350490027
2044897763
35005211
1369133069
135497281

- Write a C function that checks an input number and return a value 0 if even or 1 if odd. Your function must have a return type and an argument.

```c
#include <stdio.h>
/* Function declaration */
int evenOdd(int num);

int main()
{
    int num, isEven;

    printf("Enter a number: ");
    scanf("%d", &num);

    /* Function call */
    isEven = evenOdd(num);
    if(isEven == 0)
        printf("The given number is EVEN (%d).", isEven);
    else
        printf("The given number is ODD (%d).",isEven);
    return 0;
}

/* Function definition */
int evenOdd(int num)
{
    /* Return 0 if num is even */
    if(num % 2 == 0)
        return 0;
    else
        return 1;
}
```

OUTPUT

Enter a number: 2

The given number is EVEN (0).

Function Exercise

1. Write a function that calculates the average of three numbers.
2. Write a function to compute the square of a number.
3. Write a function that returns the largest number in a list(array) of 5 numbers.
4. Write a function that prints all elements in a given array.
5. Write a function that prints the binary representation of a decimal number.
6. Write a function to check if a character is a vowel, consonant, or a special character.