Department of
Computer Science and Engineering

Title: User Defined Function

Structured Programming Lab
CSE 104

Green University of Bangladesh

1 Learning Objective

- To attain knowledge on Function prototype, Function definition, Arguments passing, Return statement
- To identify the scope and lifetime of local and global variables
- to implement Recursive functions
- To implement programs using User Defined Functions

2 C User-defined functions

A function is a block of code that performs a specific task. C programming language allows coders to define functions to perform special tasks. As functions are defined by users, they are called user-defined functions. For example:

Here is an example to add two integers. To perform this task, we have created an user-defined addNumbers().

Code:

```
#include <stdio.h>
int addNumbers(int a, int b);         // function prototype

int main()
{
    int n1,n2,sum;

    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);

    sum = addNumbers(n1, n2);        // function call
    printf("sum = %d",sum);

    return 0;
}

int addNumbers(int a, int b)         // function definition
{
    int result;
    result = a+b;
    return result;                  // return statement
}
```

2.1 Function prototype

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body. A function prototype gives information to the compiler that the function may later be used in the program.

2.1.1 Syntax of function prototype

`returnType functionName(type1 argument1, type2 argument2, ...);`

In the above example, int addNumbers(int a, int b); is the function prototype which provides the following information to the compiler:

1. name of the function is addNumbers()
2. return type of the function is int
3. two arguments of type int are passed to the function

The function prototype is not needed if the user-defined function is defined before the main() function.

2.2 Calling a function

Control of the program is transferred to the user-defined function by calling it.

2.2.1 Syntax of function call

```
functionName(argument1, argument2, ...);
```

In the above example, the function call is made using addNumbers(n1, n2); statement inside the main() function.

2.3 Function definition

Function definition contains the block of code to perform a specific task. In our example, adding two numbers and returning it.

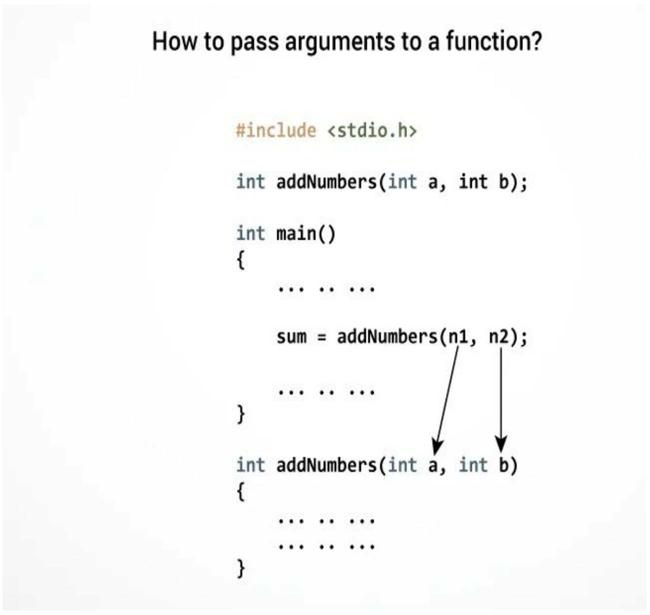
2.3.1 Syntax of function definition

```
returnType functionName(type1 argument1, type2 argument2, ...)
{
    //body of the function
}
```

When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function.

2.4 Passing arguments to a function

In programming, argument refers to the variable passed to the function. In the above example, two variables n1 and n2 are passed during the function call. The parameters a and b accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.



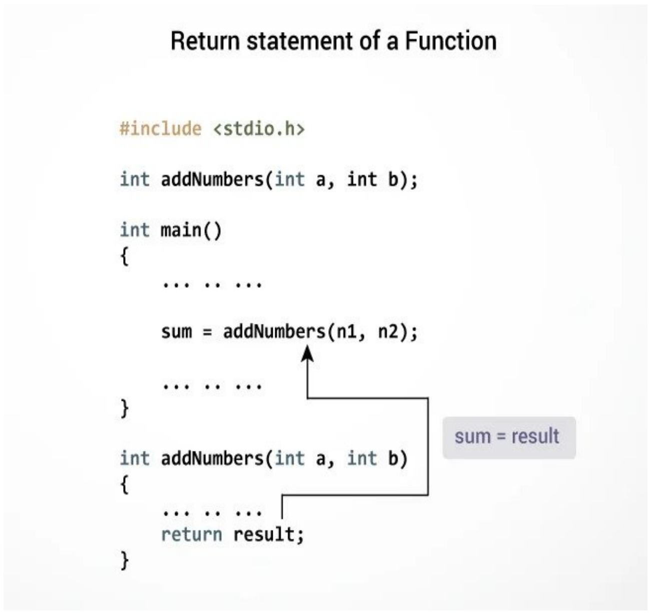
```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    ... .. ...
}
```

Figure 1: Passing Argument to Function

The type of arguments passed to a function and the formal parameters must match, otherwise, the compiler will throw an error. If n1 is of char type, a also should be of char type. If n2 is of float type, variable b also should be of float type.

2.5 Return Statement

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after the return statement.
In the above example, the value of the result variable is returned to the main function. The sum variable in the main() function is assigned this value.



```c
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... .. ...

    sum = addNumbers(n1, n2);

    ... .. ...
}

int addNumbers(int a, int b)
{
    ... .. ...
    return result;
}
```

Figure 2: Return statement of a Function

2.5.1 Syntax of return statement

`return (expression);`

For example,

```c
return a;
return (a+b);
```

The type of value returned from the function and the return type specified in the function prototype and function definition must match.

3 Example of User-defined function

3.1 Example 1: No arguments passed and no return value

Code:

```c
#include <stdio.h>

void checkPrimeNumber();

int main()
{
    checkPrimeNumber();    // argument is not passed
    return 0;
}

// return type is void meaning doesn't return any value
void checkPrimeNumber()
{
    int n, i, flag = 0;

    printf("Enter a positive integer: ");
    scanf("%d",&n);

    for(i=2; i <= n/2; ++i)
    {
        if(n%i == 0)
        {
            flag = 1;
        }
    }
    if (flag == 1)
        printf("%d is not a prime number.", n);
    else
        printf("%d is a prime number.", n);
}
```

The checkPrimeNumber() function takes input from the user, checks whether it is a prime number or not and displays it on the screen.

The empty parentheses in checkPrimeNumber(); statement inside the main() function indicates that no argument is passed to the function.

The return type of the function is void. Hence, no value is returned from the function.

3.2 Example 2: No arguments passed but a return value

Code:

```c
#include <stdio.h>
int getInteger();

int main()
{
    int n, i, flag = 0;

   // no argument is passed
    n = getInteger();

    for(i=2; i<=n/2; ++i)
    {
        if(n%i==0){
            flag = 1;
            break;
        }
    }

    if (flag == 1)
        printf("%d is not a prime number.", n);
    else
        printf("%d is a prime number.", n);

    return 0;
}

// returns integer entered by the user
int getInteger()
{
    int n;

    printf("Enter a positive integer: ");
    scanf("%d",&n);

    return n;
}
```

The empty parentheses in the n = getInteger(); statement indicates that no argument is passed to the function. And, the value returned from the function is assigned to n.

Here, the getInteger() function takes input from the user and returns it. The code to check whether a number is prime or not is inside the main() function.

3.3 Example 3: Argument passed but no return value

Code:

```c
#include <stdio.h>
void checkPrimeAndDisplay(int n);

int main()
{
    int n;

    printf("Enter a positive integer: ");
    scanf("%d",&n);

    // n is passed to the function
    checkPrimeAndDisplay(n);

    return 0;
}

// return type is void meaning doesn't return any value
void checkPrimeAndDisplay(int n)
{
    int i, flag = 0;

    for(i=2; i <= n/2; ++i)
    {
        if(n%i == 0){
            flag = 1;
            break;
        }
    }
    if(flag == 1)
        printf("%d is not a prime number.",n);
    else
        printf("%d is a prime number.", n);
}
```

The integer value entered by the user is passed to the checkPrimeAndDisplay() function.

Here, the checkPrimeAndDisplay() function checks whether the argument passed is a prime number or not and displays the appropriate message.

3.4 Example 4: Argument passed and a return value

Code:

```c
#include <stdio.h>
int checkPrimeNumber(int n);

int main()
{
    int n, flag;

    printf("Enter a positive integer: ");
    scanf("%d",&n);

    // n is passed to the checkPrimeNumber() function
    // the returned value is assigned to the flag variable
    flag = checkPrimeNumber(n);

    if(flag == 1)
        printf("%d is not a prime number",n);
    else
        printf("%d is a prime number",n);

    return 0;
}

// int is returned from the function
int checkPrimeNumber(int n)
{
    int i;

    for(i=2; i <= n/2; ++i)
    {
        if(n%i == 0)
            return 1;
    }

    return 0;
}
```

The input from the user is passed to the checkPrimeNumber() function. The checkPrimeNumber() function checks whether the passed argument is prime or not. If the passed argument is a prime number, the function returns 0. If the passed argument is a non-prime number, the function returns 1. The return value is assigned to the flag variable. Depending on whether flag is 0 or 1, an appropriate message is printed from the main() function.

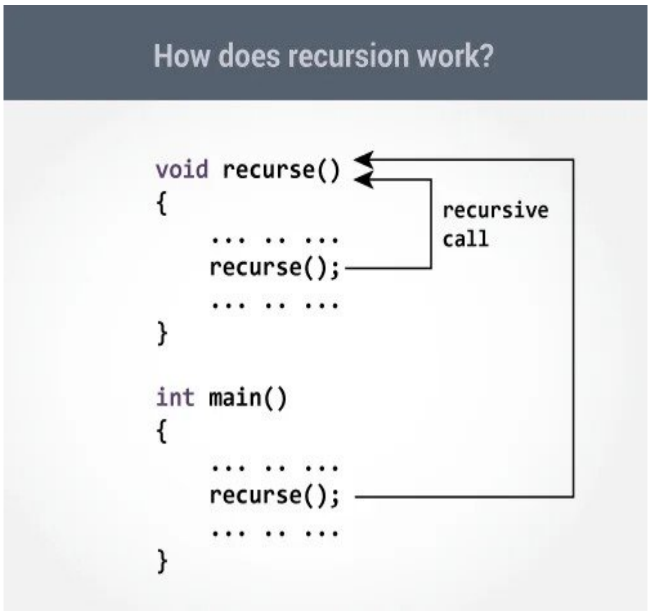
4 C Recursion

A function that calls itself is known as a recursive function. And, this technique is known as recursion.

4.1 Syntax of return statement

```c
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

The recursion continues until some condition is met to prevent it.



```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

Figure 3: Working of Recursion

4.2 Example: Sum of Natural Numbers Using Recursion

Code:

```c
#include <stdio.h>
int sum(int n);

int main() {
    int number, result;

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

    result = sum(number);

    printf("sum = %d", result);
    return 0;
}

int sum(int n) {
    if (n != 0)
        // sum() function calls itself
        return n + sum(n-1);
    else
        return n;
}
```

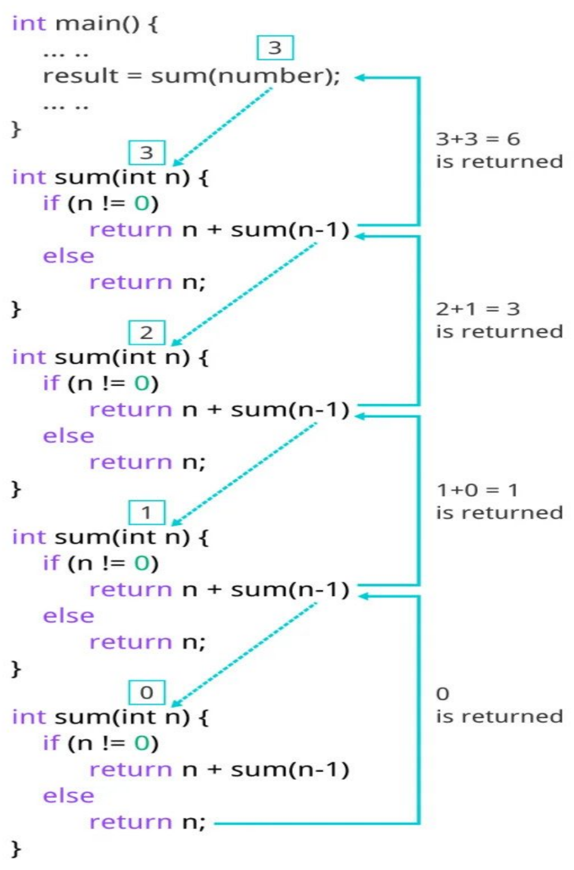
Output:

```
Enter a positive integer:3
sum = 6
```

Initially, the sum() is called from the main() function with number passed as an argument.

Suppose, the value of n inside sum() is 3 initially. During the next function call, 2 is passed to the sum() function. This process continues until n is equal to 0.

When n is equal to 0, the if condition fails and the else part is executed returning the sum of integers ultimately to the main() function.



```
int main() {
   ... ..
   result = sum(number);
   ... ..
}
int sum(int n) {
   if (n != 0)
      return n + sum(n-1)
   else
      return n;
}
int sum(int n) {
   if (n != 0)
      return n + sum(n-1)
   else
      return n;
}
int sum(int n) {
   if (n != 0)
      return n + sum(n-1)
   else
      return n;
}
int sum(int n) {
   if (n != 0)
      return n + sum(n-1)
   else
      return n;
}
```

Figure 4: Sum of Natural Numbers

5 C - Scope Rules

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed. There are three places where variables can be declared in C programming language:

- Inside a function or a block which is called **local** variables.
- Outside of all functions which is called **global** variables.

5.1 Local Variables

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. The following example shows how local variables are used. Here all the variables a, b, and c are local to main() function.

```c
#include <stdio.h>

int main () {

  /* local variable declaration */
  int a, b;
  int c;

  /* actual initialization */
  a = 10;
  b = 20;
  c = a + b;

  printf ("value of a = %d, b = %d and c = %d\n", a, b, c);

  return 0;
}
```

5.2 Global Variables

Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. The following program show how global variables are used in a program.

```c
#include <stdio.h>

/* global variable declaration */
int g;

int main () {

  /* local variable declaration */
  int a, b;

  /* actual initialization */
  a = 10;
  b = 20;
  g = a + b;

  printf ("value of a = %d, b = %d and g = %d\n", a, b, g);

  return 0;
}
```

A program can have same name for local and global variables but the value of local variable inside a function will take preference. Here is an example –

```c
#include <stdio.h>

/* global variable declaration */
int g = 20;

int main () {

  /* local variable declaration */
  int g = 10;

  printf ("value of g = %d\n",  g);

  return 0;
}
```

6 Lab Task (Please implement yourself and show the output to the instructor)

1. Write a C program to input a number from user and check whether given number is even or odd using functions.
2. Write a C Program to print Perfect Numbers between given interval using function.
3. Write a recursive function to generate nth Fibonacci term in C programming.
4. Write a C program to input a number from user and find power of given number using recursion.

7 Lab Exercise (Submit as a report)

1. Write a C Program to convert a decimal number to an equivalent binary number using function. [Decimal to Binary Conversion Method]

2. Write a C program to create menu driven calculator that performs basic arithmetic operations (add, subtract, multiply and divide) using functions.

3. Write a C Program to print Strong Numbers between given interval using function.

4. Write a C program to calculate sum of all digits of a number using recursion.

8 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.