Unit 7.1

Function

Introduction

- A function is a block of code that performs a specific task.
- A function is a set of statements that take inputs, do some specific computation and produces output.
- The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.
- Function helps in dividing complex problem into small components makes program easy to understand and use.

Types of function

- Depending on whether a function is defined by the user or already included in C compilers, there are two types of functions in C programming
- There are two types of function in C programming:
  - Standard library functions
  - User defined functions

Standard library functions

- The standard library functions are built-in functions in C programming to handle tasks such as mathematical computations, I/O processing, string handling etc.
- These functions are defined in the header file. When you include the header file, these functions are available for use. For example:
- The printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in "stdio.h" header file.
- There are other numerous library functions defined under "stdio.h", such as scanf(), fprintf(), getchar() etc. Once you include "stdio.h" in your program, all these functions are available for use.

User-defined function

- Functions created by the user are called user-defined functions.
- User defined function has basically following characteristics
  - A function is named with unique name
  - A function performs a specific task
  - A function is independent
  - A function may receive values from the calling program (caller)
  - A function may return a value to the calling program

Example

```
#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
}
```

Function Components

- **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.
- **Syntax of function prototype**
- returnType functionName(type1 argument1, type2 argument2,...);

- For example

  ```
  int addNumbers(int a, int b);
  ```

- It is the function prototype which provides following information to the compiler:
- name of the function is addNumbers()
- return type of the function is int
- 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.

Function definition

- Function definition contains the block of code to perform a specific task
- **Syntax of function definition**

```
returnType functionName(type1 arg1, type2 arg2, ...)
{
 //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.

Calling a function

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

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

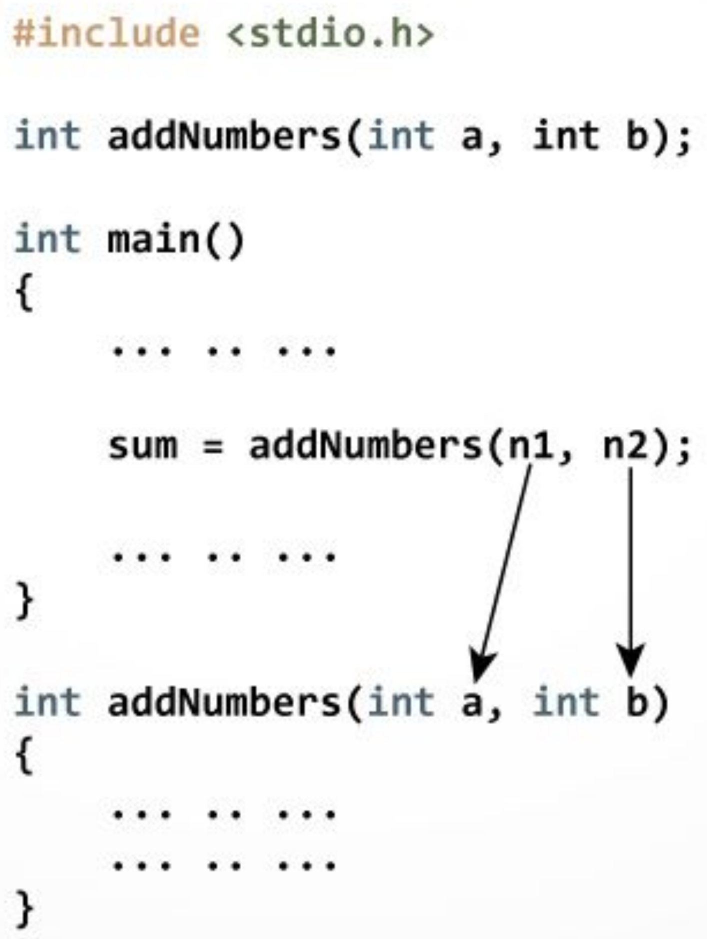
- For example

```
void main()
{
 addNumbers(n1,n2);
}
```

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 function call.
- The parameters a and b accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.
- The type of arguments passed to a function and the formal parameters must match, otherwise the compiler throws 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.
- A function can also be called without passing an argument.

How to pass arguments to a function?

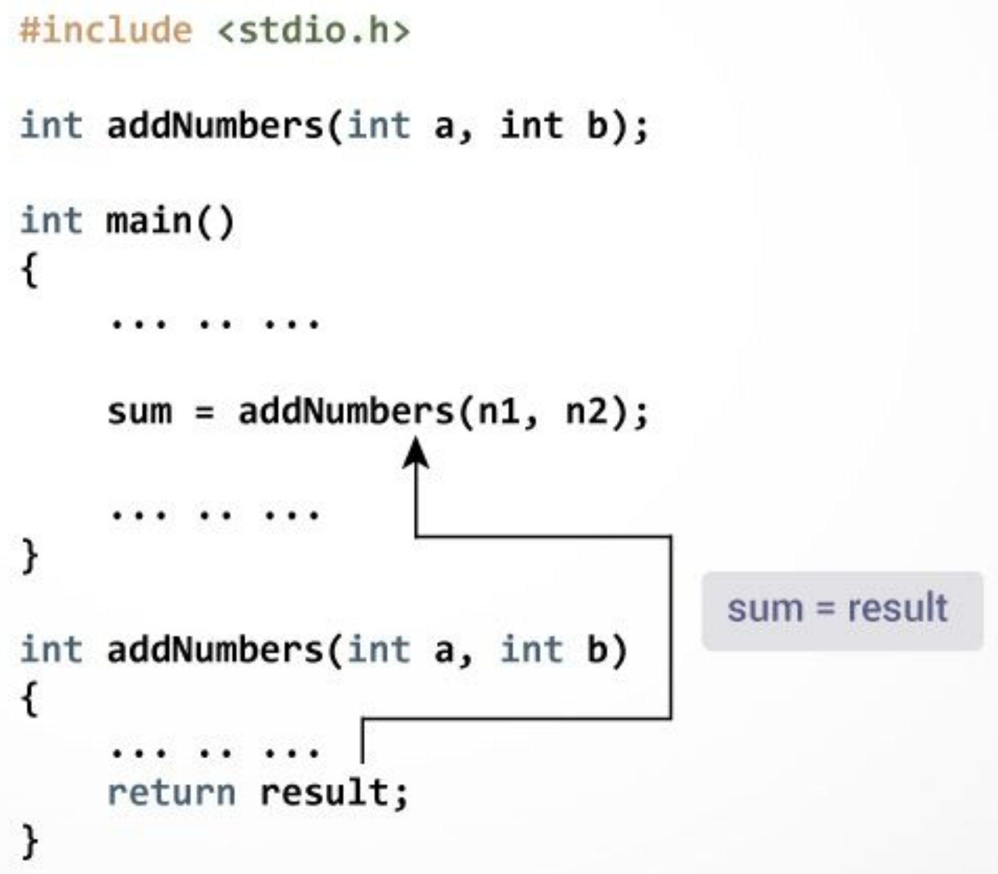


```c
#include <stdio.h>

int addNumbers(int a, int b);

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

    sum = addNumbers(n1, n2);

    ... .. ...
}

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

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 return statement.
- In the above example, the value of variable result is returned to the variable sum in the main() function.

Return statement of a Function



```
#include <stdio.h>

int addNumbers(int a, int b);

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

    sum = addNumbers(n1, n2);

    ... .. ...
}

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

Syntax of return statement

- return (expression);
- For example,
  - return a;
  - return (a+b);
- The type of value returned from the function and the return type specified in function prototype and function definition must match.

Types of User-defined Functions in C Programming

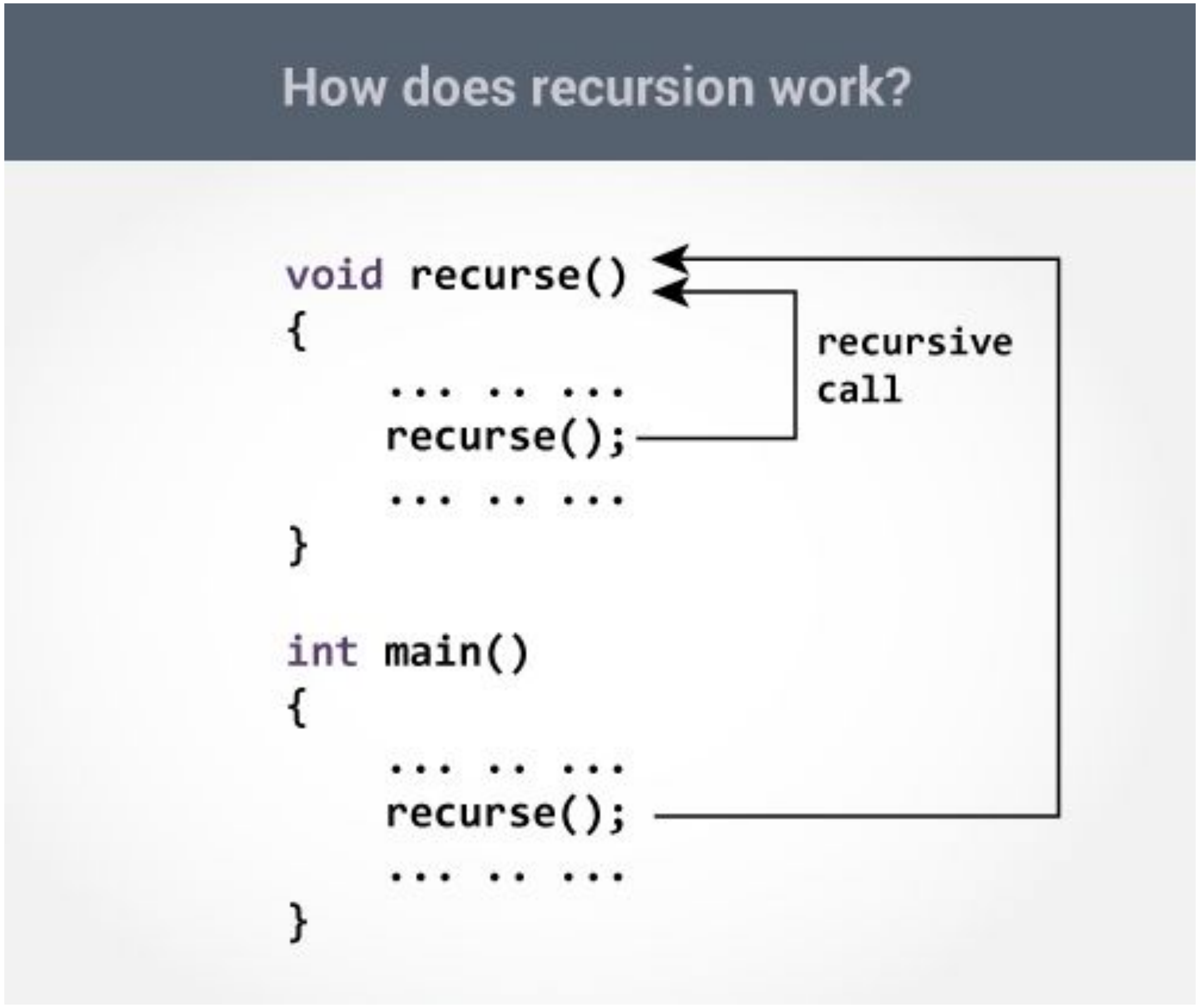
- No arguments passed and no return value
- No arguments passed but a return value
- Argument passed but no return value
- Argument passed and a return value

C Recursion

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

How recursion works?

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

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



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

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

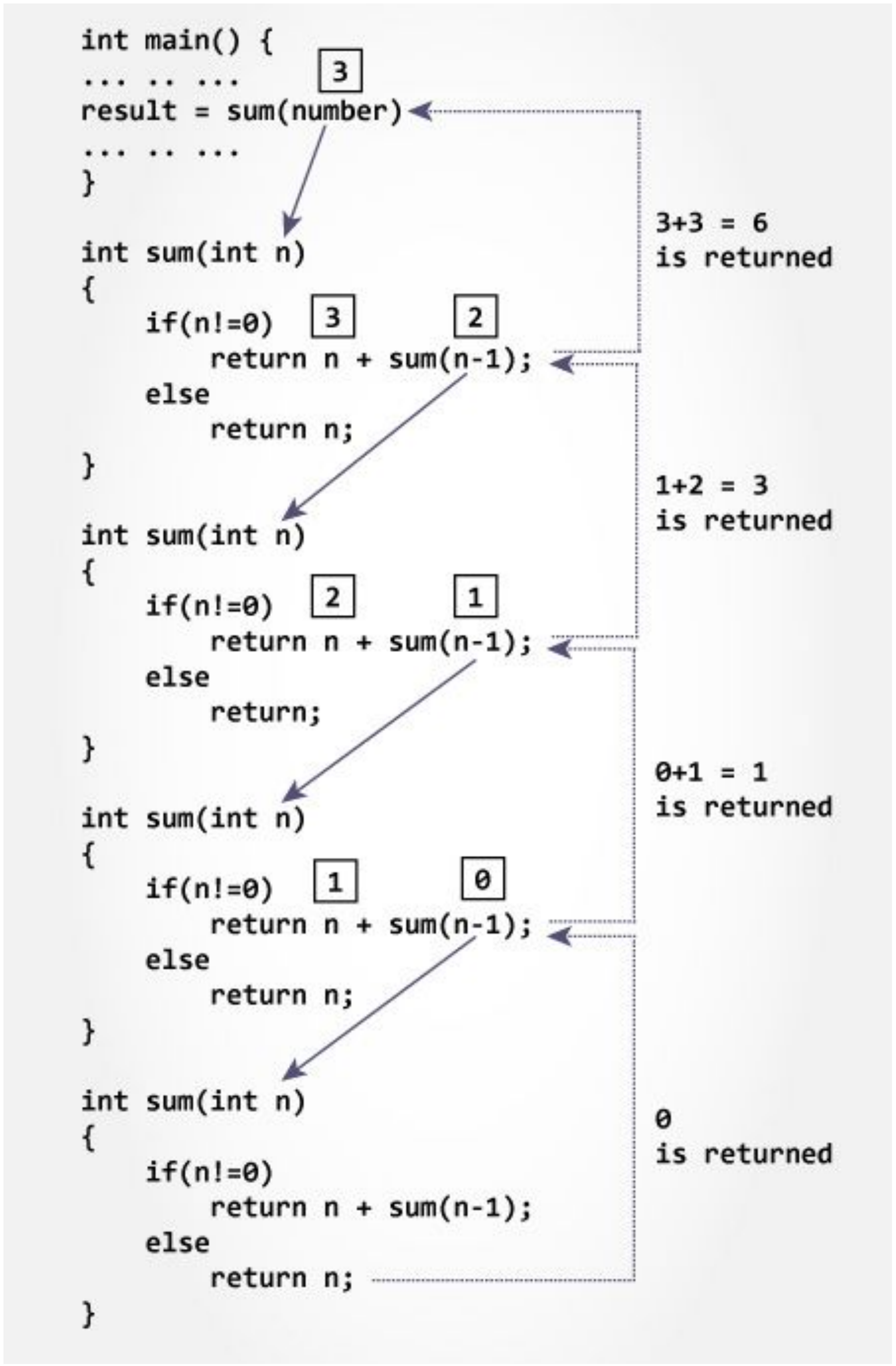
- The recursion continues until some condition is met to prevent it.
- To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call and other doesn't.

Sum of Natural Numbers Using Recursion

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

- Initially, the sum() is called from the main() function with number passed as an argument.
- Suppose, the value of num is 3 initially. During next function call, 2 is passed to the sum() function. This process continues until num is equal to 0.
- When num is equal to 0, the if condition fails and the else part is executed returning the sum of integers 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;
}

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;
}
```

Advantages and Disadvantages of Recursion

- Recursion makes program elegant and more readable. However, if performance is vital then, use loops instead as recursion is usually much slower.
- Note that, every recursion can be modeled into a loop.
- **Recursion Vs Iteration?** Need performance, use loops, however, code might look ugly and hard to read sometimes. Need more elegant and readable code, use recursion, however, you are sacrificing some performance.

How to pass arrays to a function

- **Passing One-dimensional Array to a Function**
- Passing a single element of an array to a function is similar to passing variable to a function.

```
#include <stdio.h>
void display(int age)
{
    printf("%d", age);
}
void main()
{
    int a[] = {2, 3, 4};
    display(a[2]); //Passing array element a[2]
}
```

Passing an entire array to a function

```
#include <stdio.h>
float average(int []);
void main()
{
      float avg;
      int age[] = {23, 55, 22, 5, 40, 18};
      avg = average(age); // Only name of an array is passed as an
argument
      printf("Average age = %.2f", avg);


}
```

```
float average(int age[])
{
    int i,sum=0;
    float avg;
    for (i = 0; i < 6; ++i) {
        sum += age[i];
    }
    avg = (float)sum / 6;
    return avg;
}
```

Passing Multi-dimensional Arrays to Function

- To pass multidimensional arrays to a function, only the name of the array is passed (similar to one dimensional array).

```
#include <stdio.h>
void displayNumbers(int num[2][2]);
void main()
{
   int num[2][2], i, j;
   printf("Enter 4 numbers:\n");
   for (i = 0; i < 2; ++i)
      for (j = 0; j < 2; ++j)
         scanf("%d", &num[i][j]);
    displayNumbers(num); // passing multi-dimensional array to a
function
}
```

```
void displayNumbers(int num[2][2])
{
    int i, j;
    printf("Displaying:\n");
    for (i = 0; i < 2; ++i)
        for (j = 0; j < 2; ++j)
            printf("%d\n", num[i][j]);
}
```