Note for Lecture 6

What is a function ?

A function is a block of code that performs a specific task. It is a self-contained unit of code that can be reused multiple times in a program.

Why functions are important in programming ?

Functions are important in programming for a number of reasons. They can:

Reusability: Functions can be reused multiple times in a program, which can save time and effort.

For example, you can define a function to calculate the factorial of a number, and then call that function whenever you need to calculate the factorial of a number.

Modularity: Functions can be used to break down a program into smaller, more manageable pieces. This can make the program easier to understand and maintain.

For example, you can define a function to read a file, a function to write to a file, and a function to search a file. Then, you can call these functions as needed to perform the tasks of reading, writing, and searching files.

Abstraction: Functions can hide the details of how a task is performed, which can make the code easier to read and understand.

For example, you can define a function to sort a list of numbers. The function's implementation details, such as the sorting algorithm used, are hidden from the user of the function. This makes the code easier to read and understand, because the user of the function only needs to know what the function does, not how it does it.

In addition to these benefits, functions can also be used to improve the readability, maintainability, and portability of code.

What are the types of functions available?

1. Built-in functions
2. User-defined functions

Built-in functions

These are functions that are predefined in the programming language and do not need to be defined by the programmer.

For example,

printf(): This function is used to print formatted text to the console.
scanf(): This function is used to read formatted text from the console.

Math library functions:

sin(x) : This function is used to calculate the sine of a number.
sqrt(x) : This function is used to calculate the square root of a number

These math library function are defined in the `math.h` header file. To use these functions, you need to include the `math.h` header file in your program.

Example of using math library function

```
#include <math.h>

int main() {
    double x = 3.14159;
    double y = sqrt(x);
    printf("sqrt(x) = %f\n", y);
    return 0;

}
```

User-defined functions

These are functions that are defined by the programmer. User-defined functions can be used to perform specific tasks

Format of a function

```
return_type function_name(parameter_list) {
    statements;
    return expression;
}
```

Example 1 –

```
int sum_of_two_numbers(int a, int b) {
     int result;
     result = a + b;
     return result;
}
```

Example 2 – Function without parameter_list

```
int sum_of_two_numbers() {
      int x, y;
      printf("Enter two numbers: ");
      scanf("%d%d", &x, &y);
      int result = x + y;

      return result;

}
```

Example 3 – Function without return type

```
void sum_of_two_numbers(int a, int b) {
      int result;
       result = a + b;
      printf("The sum of %d and %d is %d\n", a, b, result);

}
```

Example 4 – Function without return type and parameter_list

```
void say_goodbye() {
    printf("Goodbye!\n");
}
```

Example of a program with a function

Example 1 : Calling the function **say_goodbye**. The function has no return type and no parameter list

```
#include <stdio.h>

//called function
void say_goodbye() {
    printf("Goodbye!\n");
}

int main() {
    say_goodbye();   // calling the function

    return 0;
}
```

In this when the main program executes, it calls say_goodbye() function. It will execute printf("Goodbye!\n") statement. Then it will come back to main and executes return 0 statement

Arguments

In computer programming, an argument is a value passed to a function. The function uses the argument to perform its task.

For example, the following function adds two numbers:

```
int add_two_numbers(int x, int y) {
      int result;
      result =  x + y;
      return result;
}

int main() {
    int z = add_two_numbers(10, 20);
    printf("The sum is %d\n", z);
    return 0;
}
```

When the function is called in main(), the arguments are passed to the function. For example, the following code calls the function `add_two_numbers()` and passes the values 10 and 20 as arguments:

Arguments can be passed to functions in different ways. In the above example, the arguments are passed by value.

This means

the value 10 is passed to `x` and the value 20 is passed to `y`. The function `add_two_numbers()` then adds the two numbers x and y and stores the answer in the variable result, which is 30. Then the result is returned to the main function. The returned value which is 30 is then stored in variable z inside the main function.

Assert statement

An assert statement in C is a debugging statement that checks the value of an expression. If the expression evaluates to false, the assert statement will cause the program to terminate.

The assert statement is a preprocessor directive, which means that it is processed by the preprocessor before the code is compiled. The syntax for an assert statement is:

```
assert(expression);
```

The assert statement is defined in the `assert.h` header file. To use the assert statement, you must include the `assert.h` header file in your code.

example:

```
#include <stdio.h>
#include <assert.h>

int main() {
    int x = 10;

    assert(x > 0);
    printf("The value of x is greater than 0\n");
```

```
    return 0;
}
```

In this code, the assert statement checks if the value of the variable `x` is greater than 0. If the value of `x` is not greater than 0, the assert statement will cause the program to terminate and print an error message to the console: