Functions in C

Introduction

- A programmer can solve a simple problem in C with a single function. More difficult problems can be decomposed into sub problems, each of which can be either coded directly or further decomposed.
- Decomposing difficult problems until they are directly codable as single C functions is the software engineering method of stepwise refinement. The function construct in C is used to write code for these directly solvable subproblems. These functions are combined into other functions and are ultimately used in main() to solve the original problem.

Introduction…

- The function mechanism is provided in C to perform distinct programming tasks. Some functions, such as strcpy() and rand(), are provided by libraries; others can be written by the programmer.
- A C program is made up of one or more functions, one of which is main(). Program execution always begins with main(). When program control encounters a function name, the function is called, or invoked. This means that program control passes to the function

Need for user-defined functions

- Large programs are divided in functional parts, and each of the functional part is coded separately and independently but later combined into a single unit.
- The independently coded programs are called sub-programs, and in C they are referred to as “functions”. The functions can be called and used whenever needed.

Advantages of using functions

- It facilitates top-down modular programming. The high level logic of the overall problem is solved first while the details of each lower-level functions are addressed later.
- The length of a source program can be reduced by using functions at appropriate places. This factor is particularly critical with microcomputers where memory space is limited.

Advantages of using functions

- It is easy to locate and isolate a faulty function for further investigations.
- A function may be used by many other programs. This means that a C programmer can build on whatever others have already done, instead of starting all over again from scratch.

Components of a function

- **The function name**. This is simply a unique identifier.
- **The function parameters** (also called its signature). This is a set of zero or more typed identifiers used for passing values to and from the function.
- **The function return type.** This specifies the type of value the function returns. A function which returns nothing should have the return type void.
- The body of a function contains the computational steps (statements) that comprise the function

Function definition

- The C code that describes what a function does is called the function definition.
- Syntax:

```
Return type function name (arguments)function header
{
statements
}
```

- Everything before the first brace makes up the header of the function definition, and everything between the braces makes up the body of the function definition.

Parameters in functions

- In C, the empty parameter list is always equivalent to using void. Thus, main() is equivalent to main(void). The function main() implicitly returns the integer value 0 if no explicit return expression statement is executed.

The return statement

- The return statement is a flow of control statement. When a return statement is executed, the current function terminates, and program control is immediately passed back to the place where the function was invoked.
- Syntax:

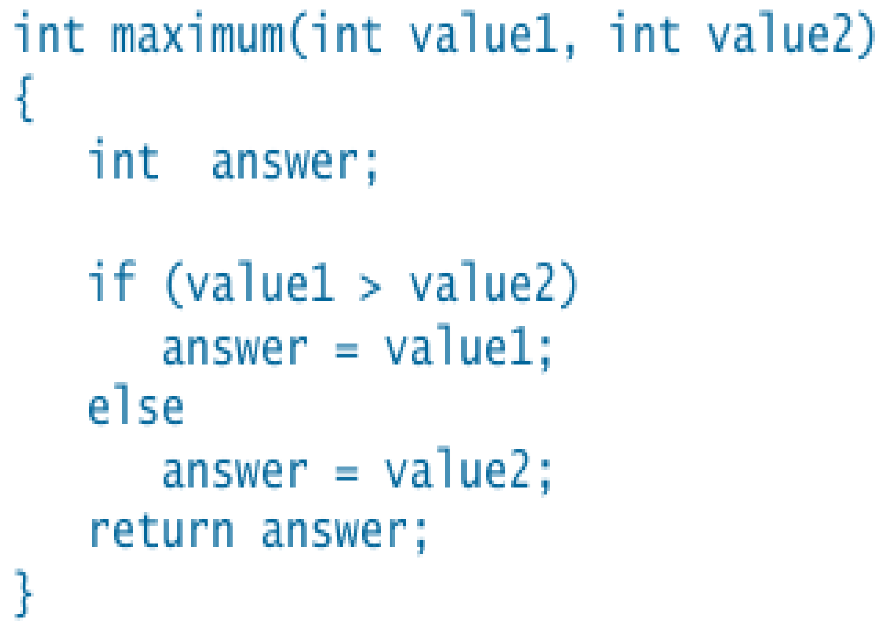
```
return;
return expression;
```

- Example:

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

Example of a function

- A function that compares two values and returns the largest of the two.



```
int maximum(int value1, int value2)
{
   int  answer;

   if (value1 > value2)
      answer = value1;
   else
      answer = value2;
   return answer;
}
```

Calling functions

- Functions are invoked by writing their name and an appropriate list of arguments within parentheses. These arguments match in number and type (or compatible type) the parameters in the parameter list in the function definition.
- The compiler enforces type compatibility. The basic argument-passing mechanism inherited from the C language is call-by-value. That is, each argument is evaluated and its value is used locally in place of the corresponding formal parameter. Thus, if a variable is passed to a function, the stored value of that variable in the calling environment will not be changed.

Function invocation with call-by-value means:

- Each expression in the argument list is evaluated.
- The value of the expression is converted, if necessary, to the type of the formal parameter, and that value is assigned to its corresponding formal parameter at the beginning of the body of the function. This means a local copy is made.
- The body of the function is executed using the local copy of the parameter.

Function invocation with call-by-value means:

- If a return statement is executed, then control is passed back to the calling environment.
- If the return statement includes an expression, then the value of the expression is converted, if necessary, to the type given by the type specifier of the function, and that value is passed back to the calling environment, too.
- If the return statement does not include an expression, then no useful value is returned to the calling environment.

Function invocation with call-by-value means:

- If no return statement is present, then control is passed back to the calling environment when the end of the body of the function is reached. No useful value is returned.
- All arguments are passed call-by-value. A change in the value of the local copy does not affect the passed in arguments value.

Default arguments

- A formal parameter can be given a default argument, usually a constant that occurs frequently when the function is called. Use of a default argument saves writing this default value at each invocation.

Example:

```
int compute_age(int year, month mth, int birth_year = 1989,
    month birth_month = january);
```

Functions as arguments

- Functions in C can be thought of as the addresses of the compiled code residing in memory. Functions are therefore a form of pointer and can be passed as a pointer-value argument into another function.

Scope

- The core language has two principal forms of scope: local scope and file scope. Local scope is scoped to a block. Compound statements that include declarations are blocks.
- Function bodies are examples of blocks. They contain a set of declarations that include their parameters. File scope has names that are external (global).

Scope

- Every variable and function in C has two attributes: type and storage class. The four storage classes are automatic, external, register, and static, with corresponding keywords