Top-Down Design with Functions

OSW 3541

Department of Computer Science & Engineering
ITER, Siksha 'O' Anusandhan Deemed To Be University
Jagamohan Nagar, Jagamara, Bhubaneswar, Odisha - 751030

Book(s)

Text Book(s)

Jeri R Hanly, & Elliot B. Koffman

Problem Solving & Program Design in C

Seventh Edition

Pearson Education

Kay A. Robbins, & Steve Robbins

Unix™ Systems Programming

Communications, concurrency, and Treads

Pearson Education

Reference Book(s)

Brain W. Kernighan, & Rob Pike

The Unix Programming Environment

PHI

Talk Flow

Functions in C

A function is a block of code which only runs when it is called and performed a task. We can categorize C functions into two categorizes, i.e., Library functions and User defined functions. Some examples of these two types of functions are given in the following table.

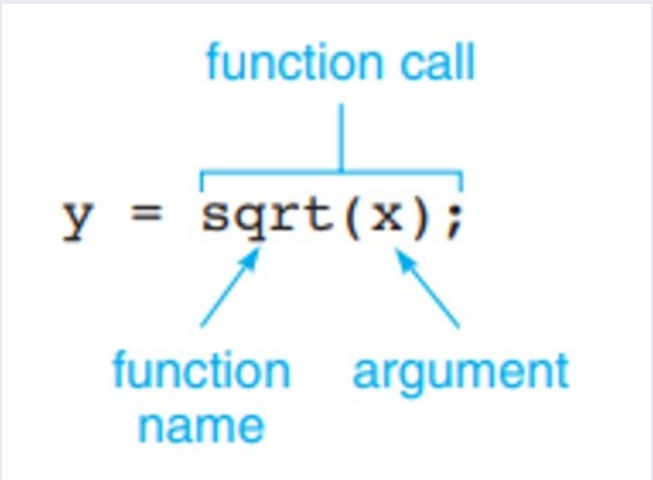
Table 1: Functions

Library functions	User defined functions
printf()	*sum*()
scanf()	*check_digit*()
sqrt()	*check_odd_even*()
abs()	*check_prime*()

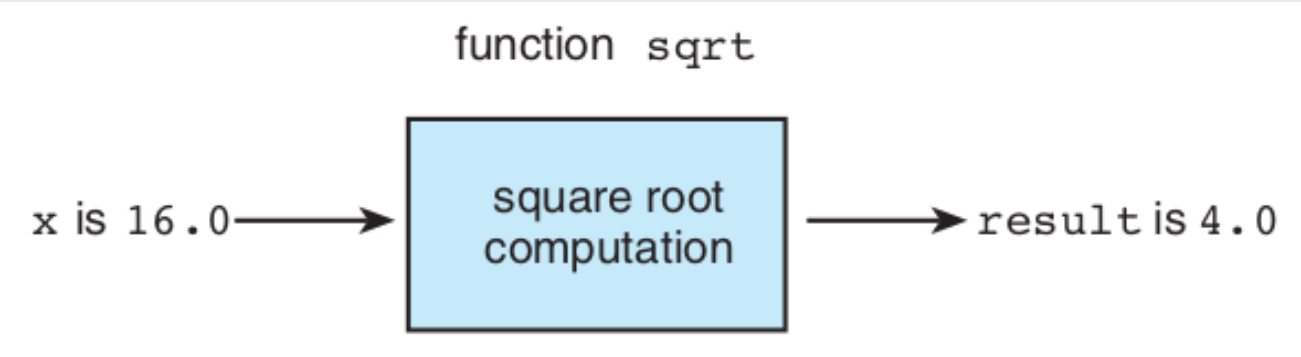
Functions in C

Library Functions (predefined functions and code reuse)

Library functions are built-in functions that are grouped together and placed in a common location called library.



Function sqrt as a "Black Box"



Library Functions

TABLE 3.1 Some Mathematical Library Functions

Function	Standard Header File	Purpose: Example	Argument(s)	Result
`abs(x)`	`<stdlib.h>`	Returns the absolute value of its integer argument: if `x is -5`, `abs(x)` is 5	`int`	`int`
`ceil(x)`	`<math.h>`	Returns the smallest integral value that is not less than `x`: if `x` is `45.23`, `ceil(x)` is `46.0`	`double`	`double`
`cos(x)`	`<math.h>`	Returns the cosine of angle `x`: if `x` is `0.0`, `cos(x)` is `1.0`	`double` (radians)	`double`
`exp(x)`	`<math.h>`	Returns e^x where e = `2.71828...`: if `x` is `1.0`, `exp(x)` is `2.71828`	`double`	`double`
`fabs(x)`	`<math.h>`	Returns the absolute value of its type `double` argument: if `x` is `-8.432`, `fabs(x)` is `8.432`	`double`	`double`
`floor(x)`	`<math.h>`	Returns the largest integral value that is not greater than `x`: if `x` is `45.23`, `floor(x)` is `45.0`	`double`	`double`
`log(x)`	`<math.h>`	Returns the natural logarithm of `x` for `x > 0.0`: if `x` is `2.71828`, `log(x)` is `1.0`	`double`	`double`
`log10(x)`	`<math.h>`	Returns the base-10 logarithm of `x` for `x > 0.0`: if `x` is `100.0`, `log10(x)` is `2.0`	`double`	`double`
`pow(x, y)`	`<math.h>`	Returns x^y. If `x` is negative, `y` must be integral: if `x` is `0.16` and `y` is `0.5`, `pow(x,y)` is `0.4`	`double, double`	`double`
`sin(x)`	`<math.h>`	Returns the sine of angle `x`: if `x` is `1.5708`, `sin(x)` is `1.0`	`double` (radians)	`double`
`sqrt(x)`	`<math.h>`	Returns the nonnegative square root of `x` ($\sqrt{x}$) for `x` $\geq$ `0.0`: if `x` is `2.25`, `sqrt(x)` is `1.5`	`double`	`double`
`tan(x)`	`<math.h>`	Returns the tangent of angle `x`: if `x` is `0.0`, `tan(x)` is `0.0`	`double` (radians)	`double`

Library Functions

We can use the C functions pow(power) and sqrt to compute the root of a quadratic equation in x of the form $ax^2 + bx + c = 0$

The two roots are defined as

$$root_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad root_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

```
/* Compute two roots, root_1 and root_2, for disc > 0.0 */
disc = pow(b,2) - 4 * a * c;
root_1 = (-b + sqrt(disc)) / (2 * a);
root_2 = (-b - sqrt(disc)) / (2 * a);
```

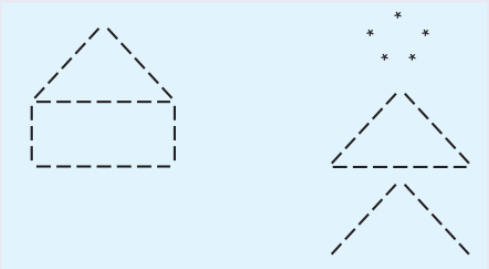
Top-Down Design and Structure Charts

Top-down design is a problem-solving method in which you first break a problem up into its major subproblems and then solve the subproblems to derive the solution to the original problem.
Structure chart is a documentation tool that shows the relationships among the subproblems of a problem.

Drawing Simple Diagrams

House and Stick Figure
Four basic components:

- a circle
- a base line
- parallel lines
- intersecting lines



Drawing Simple Diagrams (contd.)

DESIGN

To create the stick figure, you can divide the problem into three subproblems.

INITIAL ALGORITHM

1. Draw a circle.
2. Draw a triangle.
3. Draw intersecting lines.

ALGORITHM REFINEMENTS

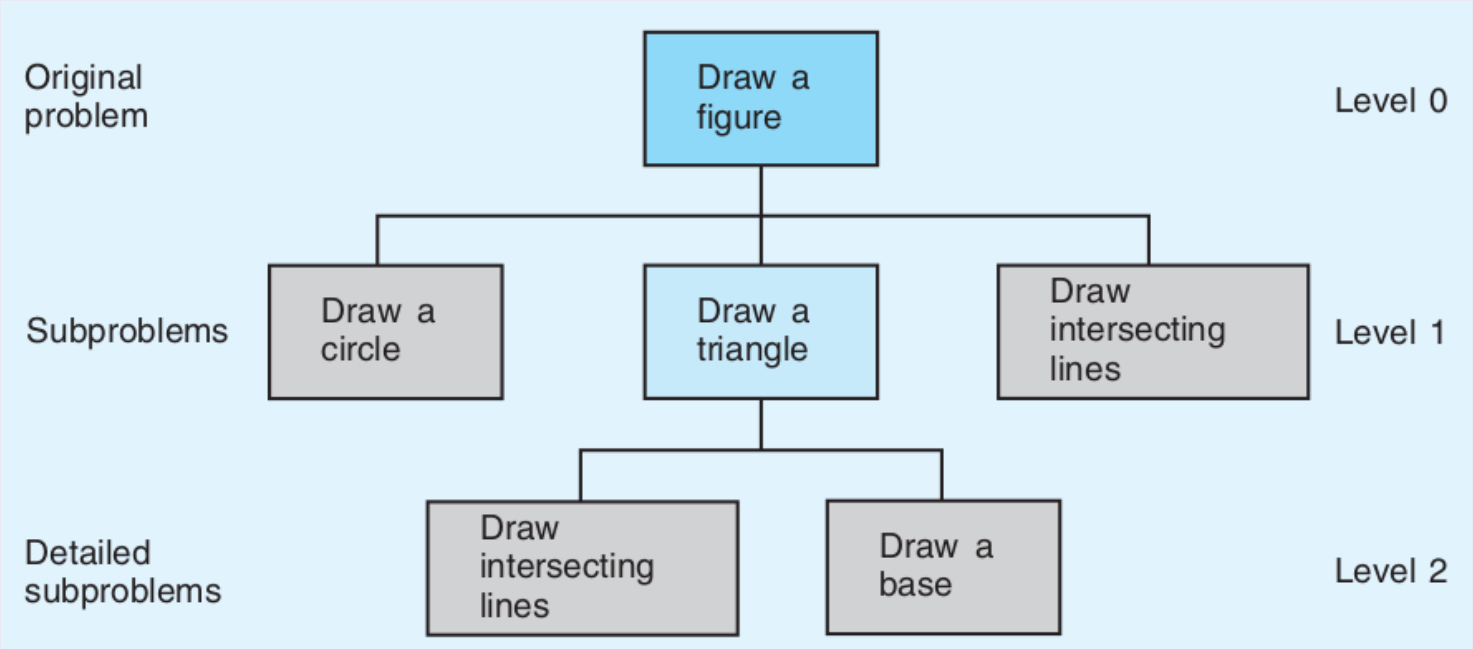
Because a triangle is not a basic component, you must refine step 2, generating the following subproblems:

Step 2 Refinement

2.1 Draw intersecting lines.
2.2 Draw a base.

Top-Down Design and Structure Charts (contd.)

Structure Chart for Drawing a Stick Figure



User defined functions

Function Prototypes

return_type function_name (*dummyargumentlist*);

Example

```
void sum(void);    /*function with no argument and returning no value */
int sum(void);    /*function with no argument and returning a value */
void sum(int, int);    /*function with multiple arguments and returning no value
*/
int sum(int, int);    /*function with multiple arguments and returning a value */
```

More examples

```
int sum(int x, int y);
int sum(int *x, int *y);
int * sum(int *x, int *y);
```

User defined functions (contd.)

Function Definition

return_type function_name (*formalargumentlist*)
{
/*local variable declarations */
/* processing statements */
return (value);
}

Example

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

User defined functions (contd.)

Function Calling

variable = *function_name* (*actual_argument_values*);

Example

result = sum(10, 20);

or

printf("*Result* = %*d*", sum(10, 20));

User defined functions (contd.)

A complete program using user defined function

```
#include < stdio.h >
int sum (int a, int b);    /* function prototype */
int main()
{
int x, y, result;
printf("Enter 2 integers: ");
scanf("%d%d", &x, &y);
result = sum(x,y);    /*calling function sum() */
printf("Result = %d", result);
return 0;
}
```

User defined functions (contd.)

A complete program using user defined function (contd.)

```
/* definition of sum() */
int sum(int p, int q)
{
int s;
s=p+q;
return s;
}
```

User defined functions (contd.)

Argument List Correspondence

- The **n**umber of actual arguments used in a call to a function must be the same as the number of formal parameters listed in the function prototype.
- The **o**rder of arguments in the lists determines correspondence. The first actual argument corresponds to the first formal parameter, the second actual argument corresponds to the second formal parameter, and so on.
- Each actual argument must be of a data **t**ype that can be assigned to the corresponding formal parameter with no unexpected loss of information.

Quick-check Exercises

1. Developing a program from its documentation means that every statement in the program has a comment. True or false?
2. The principle of code reuse states that every function in your program must be used more than once. True or false?

3. Write this equation as a C statement using functions `exp`, `log`, and `pow`:

$$y = (e^{n \ln b})^2$$

4. What is the purpose of a function argument?
5. Each function is executed in the order in which it is defined in the source file. True or false?
6. How is a function in a C program executed?
7. What is a formal parameter?
8. Explain how a structure chart differs from an algorithm.
9. What does the following function do?

```
void
nonsense(void)
{
        printf("*****\n");
        printf("*   *\n");
        printf("*****\n");
}
```

Functions in C (contd.)

Quick-check Exercises (contd.)

10. What does the following main function do?

```
int
main(void)
{
        nonsense();
        nonsense();
        nonsense();

        return (0);
}
```

11. If an actual argument of `-35.7` is passed to a type `int` formal parameter, what will happen? If an actual argument of `17` is passed to a type `double` formal parameter, what will happen?

Programming Projects

Problem Solving & Program Design in C

Seventh Edition

Pearson Education

Chapter 2