Functions

Fundamentals of Programming

Lecture 4

©2020 PSB Academy

Contents

- Function
- Declaration of a function
- Multiple functions
- Function prototypes
- Library functions

Function

- A function contains code to perform a task that is repeatable
- To declare a function:
  - Return variable data type
  - Function identifier
  - Input parameters
  - Body of the function consisting of compound statements
- **Function header** is the first line of the function

```
return_type function_identifier(input_parameters)
```

Declaration of a function

```
return_type function_identifier(input_parameters)
{
  variable declarations
  statements
  return expression
}
```

Function identifier

- Name of the function
- Same rules apply as to names for variables, arrays, etc.
- Common naming conventions for identifiers
  - PascalCase
  - camelCase

Input parameters

- Input parameters (arguments) are used to provided symbolic variables that can be manipulated within the function
- It is not a variable, but treated like a variable
- Function reads input parameters using parameter variables

Function variables

- Local variables are declared in the function
- Global variables are declared outside of any function anywhere in the program

```
#include <stdio.h>
int x; // global variable
int main()
{
  int y; // local variable
  printf("Hello\n");
  return 0;
}
```

Function return type

- A function must declare a `return_type`
- Function that returns a value must use the keyword `return` followed by an expression
- Any valid data type:
  - void
  - int, float, double, char
- Void function – Function does not return any values
- Value-returning function – Function returns a value of data type

Preprocessor directives (1)

- Directives are special instructions to the compiler before compiling the source code
- Directives may involve:
  - Text processing
  - Setting compiler options
  - Ignore certain portions of the source code based on a flag value

Preprocessor directives (2)

- Directives always begin with #
- #include
  - To insert library or user header files in the program
- #define
  - To create macros (symbolic constants) in place of values
- #pragma
  - To configure compiler specific options or settings

Directive #include

- Directive #include is used to insert standard library header files or user-defined header files into a program
- Header files contain function headers, which can make programming effort easier
- This allows for the program to call on useful functions to solve problems
- Standard header files use angular brackets < … >
  - #include <header_file.h>
- User-defined header files use double quotes " … "
  - #include "user_header_file.h"

Directive example 1

```
#include <stdio.h>
#include <stdlib.h>
#include "myfunctions.h"
#include "C:\functions\newtype.h"
#define SIZE 20
#define BYTE 8
#define OFF 0
#define ON 1
```

Directive example 2

- The macro constants OFF and ON are assigned with values 0 and 1 respectively
- Using an IF statement, the value in the macro may also represent True and False conditions
- ON = 1 = True
- OFF = 0 = False

```
#include <stdio.h>
#define OFF 0
#define ON 1
int main()
{
  if (ON) {
    printf("Machine is ON\n");
  } else {
    printf("Machine is OFF\n");
  }
  return 0;
}
```

Directive example 3

- The macro MAX is represented by 5
- Before the source code is compiled, the compiler will replace any occurrence of MAX with 5
- In main() the FOR loop uses the macro MAX to control the loop in its second expression (n < MAX)

```
#include <stdio.h>
#define MAX 5
int main()
{
  int n;
  for (n = 0; n < MAX; n++) {
    printf("%d ", n);
  }
  printf("\n");
  return 0;
}
```

Directive example 4

- Macro constant N is assigned 3
- N is used to specify the size of array x[]
- N is also used in calc() to control the FOR loop

```
#include <stdio.h>
#define N 3
int x[N] = {1, 2, 3};

int calc(void)
{
  int n, total = 0;
  for (n = 0; n < N; n++) {
    total += x[n];
  }
  return total;
}
int main()
{
  printf("Total = %d\n", calc());
  return 0;
}
```

Multiple functions

Program without external functions

- All work done in main() function is possible
- It becomes hard to manage when more work is required
- Repeated tasks will require the same multiple statements
- Tasks are still run in sequence

```
#include <stdio.h>
int main()
{
   <statement_1>
   <statement_2>
   <statement_3>
   ...
   <statement_N>
   return 0;
}
```

Function call

- Function are called from main() function or any external functions
- Functions are called from main() because
  - Program starts at main()
  - Better management of functions
- To call a function, state the name with parentheses
- Two types of function calls:
  - Call by value
  - Call by reference

Function example 1

- The hello() function will print the message “Hello” when called
- Return type is `void` because no value is returned
- Since no input parameters are used, the `void` in parentheses is used
- In main(), state the function name followed by parentheses

```
#include <stdio.h>
void hello(void)
{
  printf("Hello\n");
}

int main()
{
  hello();
  return 0;
}
```

Function call with input parameters

- Functions with input parameters need to be called with values placed in a pair of parentheses

```
void func1(int a) // expects an integer
void func2(float a) // expects an float
void func3(int a[]) // expects an integer array
void func4(int *p) // expects an integer pointer
```

Function with multiple inputs

- Function with multiple input parameters are separated by commas
- Function func() must be called with two input arguments
- Input parameters with different data types can be used

```
void func5(int a, int b)
void func6(int a, float b)
void func7(int a[], int size)
void func8(float x, float y, int *p, int *q)
```

©2020 PSB Academy

Function example 2 (Call by value)

- The display() has an input parameter (int a)
- When display() is called, the function will print the variable a to the terminal
- In the main() function, the value 3 is to be displayed, so call display(3)

```
#include <stdio.h>
void display(int a)
{
  printf("%d\n", a);
}
int main()
{
  display(3);
  return 0;
}
```

Function example 3 (Call by reference)

- The display() has an int pointer (int *a) as input parameter
- When display() is called, the function will print the value from pointer *a to the terminal
- In the main() function, the value 3 is assigned to variable i
- Function display(&i) is called with the address of variable i as argument

```
#include <stdio.h>
void display(int *a)
{
  printf("%d\n", *a);
}
int main()
{
  int i = 3;
  display(&i);
  return 0;
}
```

Function example 4 (Call by value)

- Function add() reads a and b
- Save result from a+b in variable x
- Variable x is output to the terminal
- In main(), add(4, 5) means a = 4 and b = 5 during parameter passing

```
#include <stdio.h>
void add(int a, int b)
{
  int x;
  x = a + b;
  printf("Result = %d\n", x);
}

int main()
{
  add(4, 5);
  return 0;
}
```

©2020 PSB Academy

Function example 5 (Call by reference)

- Function add( ) reads integer pointers a and b
- In main(), local pointers p and q are initialised with address values of variables i and j respectively
- Calling function add(p, q) will pass the address of variables i and j to the function

```
#include <stdio.h>
void add(int *a, int *b)
{
  int x;
  x = *a + *b;
  printf("Result = %d\n", x);
}
int main()
{
  int i=4, j=5;
  int *p, *q;
  p = &i;
  q = &j;
  add(p, q);
  return 0;
}
```

Function with return data

- For functions that return data, the caller must prepare to store the returned data in a variable (local or global)
- The variable used must match the return data type
- Usually only one value is returned by the external function

Function example 6 (Call by value)

- The add() is a value-returning function with two input parameters, a and b
- In main() variable x is used to store the returned data from add(4,5)
- Result is output to the terminal using PRINTF()

```
#include <stdio.h>
int add(int a, int b)
{
  return (a + b);
}
int main()
{
  int x;
  x = add(4,5);
  printf("Result = %d\n", x);
  return 0;
}
```

Function example 7 (Call by value)

- The display() is a void function made to print values from input parameter a
- The add() returns the sum two input parameters, a and b
- In main(), the nested function call allows add() to pass the return data as argument to display() for printing

```
display(add(4,5));
```

```
#include <stdio.h>
void display(int a)
{
  printf("Result = %d\n", a);
}
int add(int a, int b)
{
  return (a + b);
}
int main()
{
  display(add(4,5));
  return 0;
}
```

Function example 8 (Call by reference)

- Function add() has two input parameters, int pointers a and b
- In main() the add() function is called with two arguments, the addresses &i and &j
- Variable x is used to store the returned data from add(&i , &j)
- Result is output to the terminal using PRINTF()

```
#include <stdio.h>
int add(int *a, int *b)
{
  return (*a + *b);
}
int main()
{
  int x, i=4, j=5;
  x = add(&i, &j);
  printf("Result = %d\n", x);
  return 0;
}
```

Function example 9 (Call by reference)

- Function sum() accepts an int array with array size
  - Parameter int x[] will expect the address of the first element as input, which is why it is considered as **call by reference**
- Function sum() is called in the argument section of PRINTF() in main()
- Array size of a[] and calling sum() with N as argument is set by constant N
  - If N value is changed, the rest of the program does not have to be modified

```
#include <stdio.h>
#define N 3
int sum(int x[], int size)
{
  int n, total = 0;
  for (n=0; n<size; n++) {
    total += x[n];
  }
  return total;
}
int main()
{
  int a[N] = {1,2,3};
  printf("%d\n", sum(a, N));
  return 0;
}
```

Function prototypes

Placement of main()

- In a C program, main() can be placed anywhere
- However, the compiler needs to know what functions are referenced during compilation in main()
- So main() is usually located at the bottom of the program

Function prototypes

- Function prototype is the first line of a function, placed at the top of the program as a statement
- Its purpose is to act as reference for the compiler to know that such a function exists, and may be called in main() or other functions
- This allows the placement of main() at the top of the program

Function prototype example 1

```
#include <stdio.h>
void func1(int a);
void func2(int a);

int main()
{
  func1(3);
  func2(5);
  return 0;
}
```

```
void func1(int a)
{
  printf("%d\n", a);
}

void func2(int a)
{
  printf("%d\n", a);
}
```

User-defined header file

- Function prototypes can be moved into a header file
- Using #include allows the function prototypes to be references in other C program files (scripts)
- Also possible to include global variables that work with these function prototypes in the same header file
- Example:

```
#include <stdio.h>
#include "proj.h"
```

Function prototype example 2

```
// main.c
#include <stdio.h>
#include "proj.h"

int main()
{
  func1(3);
  func2(5);
  return 0;
}
```

```
// proj.c
#include <stdio.h>
void func1(int a)
{
  printf("%d\n", a);
}

void func2(int a)
{
  printf("%d\n", a);
}
```

```
// proj.h
void func1(int a);
void func2(int a);
```

Extern variable

- Global variables declared in a program file (script) is limited to that file
- Using keyword `extern` the global variable context will apply to all program files in the project
- This means that other C files will have access to the global variables

Extern example

```
// main.c
#include <stdio.h>
#include "proj.h"
int main()
{
   x = 2;
   printf("%d\n", x);
   func1(3);
   func2(5);
   return 0;
}
```

```
// proj.c
#include <stdio.h>
int x;
void func1(int a)
{
   printf("%d\n", a);
}
void func2(int a)
{
   printf("%d\n", a);
}
```

```
// proj.h
void func1(int a);
void func2(int a);

extern int x;
```

Library functions
ctype.h

Ctype.h library

- The ctype.h library has functions for character class tests
- Focus on a single ASCII character
- The function will check the char variable
  - Returns a non-zero (true)
  - Returns a zero (false)

Ctype.h functions

- isalnum() = check if argument is a digit or an alphabetical letter
- isalpha() = check if argument is an alphabetical letter
- isdigit() = check if argument is a digit
- isxdigit() = check if argument is a hexadecimal digit
- isspace() = check if argument is a whitespace
- ispunct() = check if argument is a punctuation character
- iscntrl() = check if argument is a control character
- toupper() = convert lowercase letters to uppercase
- tolower() = convert uppercase letters to lowercase

Isdigit() example

```
#include <stdio.h>
#include <ctype.h>
int main()
{
  char x = '7';
  if (isdigit(x)) {
    printf("x is a digit\n");
  } else {
    printf("x is not a digit\n");
  }
  return 0;
}
```

Toupper() example

```
#include <stdio.h>
#include <ctype.h>
int main()
{
  char x = 'g';
  printf("Old = %c\n", x);
  printf("New = %c\n", toupper(x));
  return 0;
}
```

Library functions
math.h

Math.h library

- The math.h library has mathematics-related functions
- Calculations involving trigonometry, log, power, exponential, fractions, etc.
- Remember, trigonometric functions use radians values
- Most of the input parameters for the math functions are double

©2020 PSB Academy

Math.h functions (1)

- sin(x) = sine of x
- cos(x) = cosine of x
- tan(x) = tangent of x
- asin(x) = inverse sine of x (where x = −1, …, 1)
- acos(x) = inverse cosine of x (where x = −1, …, 1)
- atan(x) = inverse tangent of x (where x = −1, …, 1)

©2020 PSB Academy

Math.h functions (2)

- exp(x) = exponential x
- pow(x,y) = x^y
- log(x) = natural logarithm of x (where x > 0)
- log10(x) = base 10 logarithm of x (where x > 0)
- sqrt(x) = square root of x
- fabs(x) = absolute value of |x|

©2020 PSB Academy

Sin() example

```
#include <stdio.h>
#include <math.h>
#define pi 3.141592
int main()
{
  float x = (30*pi)/180;
  printf("sin(%f) = %f\n", x, sin(x));
  return 0;
}
```

Log() example

```
#include <stdio.h>
#include <math.h>
int main()
{
  float x = 5.0;
  printf("log(%f) = %f\n", x, log(x));
  return 0;
}
```

Sqrt() example

```
#include <stdio.h>
#include <math.h>
int main()
{
  float x = 5.0;
  printf("sqrt(%f) = %f\n", x, sqrt(x));
  return 0;
}
```

Library functions
stdlib.h

Stdlib.h library

- The stdlib.h library has utility functions for:
  - Number conversion
  - Generating pseudo-random integers
  - Memory storage allocation
  - Sorting arrays
  - Absolute value
  - Determine quotient and remainder of a fraction

Stdlib.h functions

- atof(s) = converts string to double type
- atoi(s) = converts string to integer type
- rand() = generates pseudo-random number between 0 and 32767
- srand() = initialise seed value for pseudo-random number
- abs(n) = absolute value of integer n
- div(num,denom) = finds quotient and remainder of num/denom
- malloc(), calloc() = dynamic arrays
- bsearch() = binary search of an array
- qsort() = sort an array

©2020 PSB Academy

Atoi() example

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  char x[20] = "12";
  printf("Input = %s\n", x);
  printf("Output = %d\n", atoi(x));
  return 0;
}
```

Srand() and Rand() example

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  srand(123);
  printf("%d\n", rand());
  srand(456);
  printf("%d\n", rand());
  return 0;
}
```

Summary

- The function creates a more organised structure for code
- Frequently used arithmetic or logic operations should be organised as a function
- Functions can be made to be reusable and portable across projects or computer platforms

Further readings

- C: How to Program, 8th Edition, Paul Deitel & Harvey Deitel
  - Functions: pp. 189–205
- The C Programming Language, 2nd Edition, Kernighan & Ritchie
  - Functions: pp. 62–82