Functions

Functions are like building blocks
They allow complicated programs to be divided into manageable pieces
Some of the advantages of functions are:
- A programmer can focus on just that part of the program and construct it, debug it, and perfect it
- Different people can work on different functions simultaneously
- If a function is needed in more than one place in a program, or in different programs, you can write it once and use it many times
Predefined Functions

Some of the predefined mathematical functions are:
- \( \sqrt{x} \)
- \( \text{pow}(x, y) \)
- \( \text{floor}(x) \)

Predefined functions are organized into separate libraries.
I/O functions are contained in the header file `iostream`.
Math functions are contained in the header file `cmath`.

To use these functions you need to:
- include the correct header file
- know the name of the function
- know the number of parameters, if any
- know the data type of each parameter
- know the data type of the value computed by the function, called the type of the function.

The Power Function ( pow )

- \( \text{pow}(x, y) \) calculates \( x^y \), for example:
  \( \text{pow}(2, 3) = 8.0 \)
- the function \( \text{pow} \) returns a value of the type \( \text{double} \)
- \( x \) and \( y \) are called the parameters (or arguments) of the function \( \text{pow} \)
- Function \( \text{pow} \) has two parameters

<table>
<thead>
<tr>
<th>Function</th>
<th>Standard Header File</th>
<th>Purpose</th>
<th>Parameter(s) Type</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(x)</td>
<td>&lt;cstdlib&gt;</td>
<td>Returns the absolute value of its argument: ( \text{abs}(-7) = 7 )</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>&lt;cmath&gt;</td>
<td>Returns the smallest whole number that is not less than ( x ): ( \text{ceil}(56.34) = 57.0 )</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>cos(x)</td>
<td>&lt;cmath&gt;</td>
<td>Returns the cosine of angle ( x ): ( \text{cos}(0.0) = 1.0 )</td>
<td>double (radians)</td>
<td>double</td>
</tr>
<tr>
<td>exp(x)</td>
<td>&lt;cmath&gt;</td>
<td>Returns ( e^x ), where ( e = 2.71828 ): ( \text{exp}(1.0) = 2.71828 )</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>fabs(x)</td>
<td>&lt;cmath&gt;</td>
<td>Returns the absolute value of its argument: ( \text{fabs}(-5.67) = 5.67 )</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>floor(x)</td>
<td>&lt;cmath&gt;</td>
<td>Returns the largest whole number that is not greater than ( x ): ( \text{floor}(45.67) = 45.00 )</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>pow(x, y)</td>
<td>&lt;cmath&gt;</td>
<td>Returns ( x^y ); if ( x ) is negative, ( y ) must be a whole number: ( \text{pow}(0.16, 0.5) = 0.4 )</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>tolower(x)</td>
<td>&lt;cctype&gt;</td>
<td>Returns the lowercase value of ( x ) if ( x ) is uppercase; otherwise, returns ( x )</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>toupper(x)</td>
<td>&lt;cctype&gt;</td>
<td>Returns the uppercase value of ( x ) if ( x ) is lowercase; otherwise, returns ( x )</td>
<td>int</td>
<td>int</td>
</tr>
</tbody>
</table>
The sqrt and floor Functions

- The square root function sqrt(x)
  - calculates the non-negative square root of x for \( x \geq 0.0 \)
  - \( \sqrt{2.25} \) is 1.5
  - It is of the type double and has only one parameter
- The floor function floor(x)
  - calculates the largest whole number not greater than x
  - \( \text{floor}(48.79) \) is 48.0
  - It is of the type double and has only one parameter

User-Defined Functions

- Void functions - functions that do not have a data type
- Value-returning functions - functions that have a data type

Since the value returned by a value-returning function is unique, we:
- save the value for further calculation
- use the value in some calculation
- print the value

A value-returning function is either used in an assignment statement or in an output statement such as cout
User-Defined Functions

- **Formal Parameter** - A variable declared in the function heading
- **Actual Parameter** - A variable or expression listed in a call to a function

Syntax

- The syntax of the formal parameter list is:
  
  `dataType identifier, dataType identifier, ...

- The syntax for a function call is:
  
  `functionName(actual parameter list)

- The syntax for the actual parameter list is:
  
  `expression or variable, expression or variable, ...

Value-returning Functions

- The syntax is:

  `functionType functionName(formal parameter list)
  {
    statements
    return ...;
  }

  `functionType - type of the value returned by the function
  `functionType - also called the data type of the value-returning function

- The formal parameter list can be empty
- If the formal parameter list is empty
  
  `the parentheses are still needed
  `the function heading of the value-returning function takes either of the following forms:
    
    `functionType functionName()
    `functionType functionName(void)
  
  `in a function call the actual parameter is empty
- A call to a value-returning function with an empty formal parameter list is:

  `functionName()`
Value-Returning Functions

To call a value-returning function:
- Use its name, with the actual parameters (if any) in parentheses
- There is a one-to-one correspondence between actual and formal parameters
- A value-returning function is called in an expression
- The expression may be part of an assignment statement or an output statement
- A function call in a program results in the execution of the body of the called function

Flow of Execution

When the program is executed (run), execution always begins at the first statement in the function main no matter where it is placed in the program
- Other functions are executed only when they are called
- Function prototypes appear before any function definition, so the compiler translates these first
- The compiler can then correctly translate a function call

Flow of Execution

A function call statement results in the transfer of control to the first statement in the body of the called function
- After the last statement of the called function is executed, the control is passed back to the point immediately following the function call
- A value-returning function returns a value
- After executing the function when the control goes back to the caller, the value that the function returns replaces the function call statement

User-Defined Functions

An example

Define a function
- take an integer n as the input
- return n^3
User-Defined Functions

- **Another example**

- Define a function
- input 2 integers
- return the larger

```
function_larger.cpp
```

```
function_larger_2.cpp
```

Example 3

- Define a function
- input 3 integers
- return the largest

```
function_largest.cpp
```

In the body of a function, if the return statement includes more than 1 values (expression), only the last value will be returned

```
return x, y;
```

```
function_2_values.cpp
```

Example 4

- Define a function
- input n
- return n!
- Write a program, output the values of
  - 1! = 1
  - 2! = 2
  - 3! = 6
  - 4! = 24
  - 5! = 120

```
function_factorial.cpp
```
User-Defined Functions

Example 5

bool function function_isPrime.cpp
input n
output whether n is a prime

Example 6

void function without parameters
function_void_without_parameters.cpp
define a function, print 5 stars
Write a program, print
*****
*****
*****
function_void_without_parameters.cpp

Example 7

void function with parameters
function_stars.cpp
define a function, input integer n
print a line with n stars
Write a program, print
*
**
***
****
*****

Example 8

define a function printStars, input integer n
print a line with n stars

define a function printBlanks, input integer n
print a line with n blanks
Write a program, print
*
**
***
****
*****
User-Defined Functions

Example 8 another solution

- define a function printCharacters, input integer n and a character
  - print a line with n specified characters
- Write a program, print
  
  *
  ***
  *****

Passing by value(s)

- After copying data, a formal parameter has no connection with the actual parameter
- Any changes to the formal parameters cannot affect the actual parameters in the calling function (program)

Passing by reference

- An example
  ```cpp
  int f(int x);
  int main()
  {
      int i = 10;
      cout << "i=" << i << endl;
      cout << "f(i)=" << f(i) << endl;
      cout << "i=" << i << endl;
      pause();
      return 0;
  }
  ```

```cpp
int f(int x)
{
    x++;
    return x;
}
```
User-Defined Functions

- Passing by reference
  
  example 2
  
  ref_para_2.cpp

---

User-Defined Functions

- Passing by reference
  
  An example

- Write a function `compTwo`, which takes 2 integers as the parameters and returns the larger value and the smaller value

  reference_para_2_comparison.cpp

---

User-Defined Functions

- Passing by reference
  
  An example

- Write a function `compThree`, which takes 3 integers as the parameters and returns the largest value, the middle value and the smallest value

  reference_para_3_comparison.cpp
Scope of An Identifier

- **Local Identifier**: identifiers declared within a function (block)
- **Global Identifier**: identifiers declared outside of every function (block)

Dev-C++.lnk

Side effect of global variables:
- Any function that uses global variables is not independent
- If more than one function uses the same global variable and something goes wrong, it is difficult to discover what went wrong and where.
- We strongly recommend you not to use global variable

Automatic variable: a variable for which is allocated at block entry and deallocated at block exit is called an automatic variable

Static variable: a variable for which memory remains allocated as long as the program executes is called static variable

Global variables are by default static variables

“static int x;” can declare a static variable

Dev-C++.lnk

```cpp
int factorial(int n)
{
    if(f == 1)
        for(int i = 1; i <= n; i++)
            f = f * i;
    else
        f = f * n;
    return f;
}
```

Dev-C++.lnk

```cpp
int factorial(int n)
{
    static int f = 1;
    if(f == 1)
        for(int i = 1; i <= n; i++)
            f = f * i;
    else
        f = f * n;
    return f;
}
```