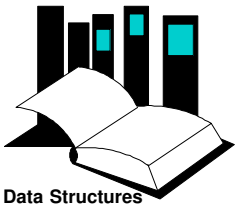
Preconditions and Postconditions



Data Structures and Other Objects Using C++

- An important topic: **preconditions** and **postconditions**.
- They are a method of specifying what a function accomplishes.

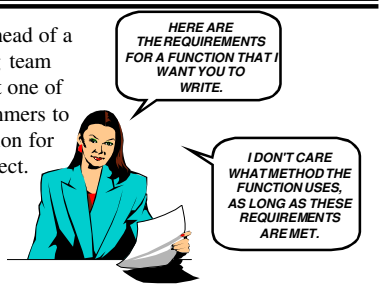
Preconditions and Postconditions

Frequently a programmer must communicate precisely **what** a function accomplishes, without any indication of **how** the function does its work.

Can you think of a situation where this would occur ?

Example

- You are the head of a programming team and you want one of your programmers to write a function for part of a project.



What are Preconditions and Postconditions?

- One way to specify such requirements is with a pair of statements about the function.
- The **precondition** statement indicates what must be true before the function is called.
- The **postcondition** statement indicates what will be true when the function finishes its work.

Example


```
void write_sqrt( double x)

//  Precondition:  x >= 0.
//  Postcondition:  The square root of x has
//  been written to the standard output.

...
```

Example


```
void write_sqrt( double x)

//  Precondition:  x >= 0.
//  Postcondition:  The square root of x has
//  been written to the standard output.
```

- The precondition and postcondition appear as comments in your program.
- They are usually placed after the function's parameter list.

Example

```
void write_sqrt( double x)

// Precondition:  x >= 0.
// Postcondition:  The square root of x has
// been written to the standard output.
```

- In this example, the precondition requires that

  x >= 0

  be true whenever the function is called.

Example

Which of these function calls meet the precondition ?

```
write_sqrt( -10 );
write_sqrt( 0 );
write_sqrt( 5.6 );
```


Example

Which of these function calls meet the precondition ?

```
write_sqrt( -10 );
write_sqrt( 0 );
write_sqrt( 5.6 );
```

The second and third calls are fine, since the argument is greater than or equal to zero.

Example

Which of these function calls meet the precondition ?

```
write_sqrt( -10 );
write_sqrt( 0 );
write_sqrt( 5.6 );
```

But the first call violates the precondition, since the argument is less than zero.

Example

```
void write_sqrt( double x)

// Precondition:  x >= 0.
// Postcondition:  The square root of x has
// been written to the standard output.
```

- The postcondition always indicates what work the function has accomplished. In this case, when the function returns the square root of **x** has been written.

Another Example

```
bool is_vowel( char letter )
// Precondition:  letter is an uppercase or
// lowercase letter (in the range 'A' ... 'Z' or 'a' ... 'z') .
// Postcondition:  The value returned by the
// function is true if Letter is a vowel;
// otherwise the value returned by the function is
// false.

  ...
```

Another Example

What values will be returned by these function calls ?

```
is_vowel( 'A' );
is_vowel(' Z' );
is_vowel( '?' );
```


Another Example

What values will be returned by these function calls ?

```
is_vowel( 'A' );
is_vowel(' Z' );
is_vowel( '?' );
```

true

false

Nobody knows, because the precondition has been violated.

Another Example

What values will be returned by these function calls ?

```
is_vowel( '?' );
```

Violating the precondition might even crash the computer.

Always make sure the precondition is valid . . .

- The programmer who calls the function is responsible for **ensuring that the precondition is valid** when the function is called.

. . . so the postcondition becomes true at the function's end.

- The programmer who writes the function counts on the precondition being valid, and **ensures that the postcondition becomes true** at the function's end.

A Quiz

Suppose that you call a function, and you neglect to make sure that the precondition is valid. Who is responsible if this inadvertently causes a 40-day flood or other disaster?

① You

② The programmer who wrote that torrential function

③ Noah

A Quiz

Suppose that you call a function, and you neglect to make sure that the precondition is valid. Who is responsible if this inadvertently causes a 40-day flood or other disaster?

① You

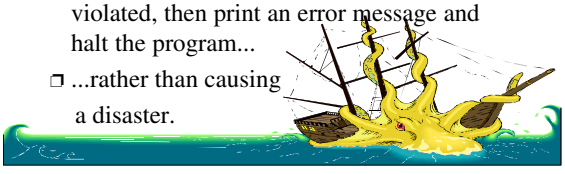
The programmer who calls a function is responsible for ensuring that the precondition is valid.

On the other hand, careful programmers also follow these rules:

- When you write a function, you should make every effort to detect when a precondition has been violated.
- If you detect that a precondition has been violated, then print an error message and halt the program.

On the other hand, careful programmers also follow these rules:

- When you write a function, you should make every effort to detect when a precondition has been violated.
- If you detect that a precondition has been violated, then print an error message and halt the program...
- ...rather than causing a disaster.



Example


```
void write_sqrt( double x)
//   Precondition:  x >= 0.
//   Postcondition:  The square root of x has
//   been written to the standard output.
{
   assert(x >= 0);

   . . .
```

- The assert function (described in Section 1.1) is useful for detecting violations of a precondition.

Advantages of Using Preconditions and Postconditions

- Succinctly describes the behavior of a function...
- ... without cluttering up your thinking with details of how the function works.
- At a later point, you may reimplement the function in a new way ...
- ... but programs (which only depend on the precondition/postcondition) will still work with no changes.

Summary

Precondition

- The programmer who calls a function ensures that the precondition is valid.
- The programmer who writes a function can bank on the precondition being true when the function begins execution.

Postcondition

- The programmer who writes a function ensures that the postcondition is true when the function finishes executing.

Presentation copyright 1997, Addison Wesley Longman
For use with *Data Structures and Other Objects Using C++*
by Michael Main and Walter Savitch.

Some artwork in the presentation is used with permission from Presentation Task Force (copyright New Vision Technologies Inc.) and Corel Gallery Clipart Catalog (copyright Corel Corporation, 3G Graphics Inc., Archive Arts, Cartesia Software, Image Club Graphics Inc., One Mile Up Inc., TechPool Studios, Totem Graphics Inc.).

Students and instructors who use *Data Structures and Other Objects Using C++* are welcome to use this presentation however they see fit, so long as this copyright notice remains intact.

THE END