Default arguments, documentation

Comp Sci 1570 Introduction to C++

Outline

1. Default arguments

2. Documentation
   - Function description
   - Side effects
   - Preconditions and postconditions
     - Guidelines
   - Examples

Default arguments to functions

- A default parameter (also called an optional parameter or a default argument) is a function parameter that has a default value provided to it.
- If the user does not supply a value for this parameter, the default value will be used.
- If the user does supply a value for the default parameter, the user-supplied value is used instead of the default value.
- All default parameters must be the rightmost parameters.
- If any parameter, every parameter after it must be

Outline

1 Default arguments

2 Documentation

- Function description
- Side effects
- Preconditions and postconditions
  - Guidelines
- Examples

Types of documentation

Documentation is an important part of software engineering. Types of documentation include:

- **Requirements** Statements that identify attributes, capabilities, characteristics, or qualities of a system. This is the foundation for what will be or has been implemented.
- **Architecture/Design** Overview of software. Includes relations to an environment and construction principles to be used in design of software components.
- **Technical** Documentation of code, algorithms, interfaces, and APIs.
- **End user** Manuals for the end-user, system administrators and support staff.
- **Marketing** How to market the product and analysis of the market demand.

Code documentation

- In addition to good variable, function, and class names, it is a good idea to include comments
- A standard way to document functions with a function description, pre-, and post-conditions.
- All three of these comments should be listed just before the prototypes of every function (or, in some cases, groups of functions).

Outline

1 Default arguments

2 Documentation

- Function description
- Side effects
- Preconditions and postconditions
  - Guidelines
- Examples

Function description

- A good function description will describe in adequate terms what the function does for the user.
- What resources are needed (inputs) to produce the results (returned values and side effects).
- It should not need to state how this is done, just what is done.

Outline

1 Default arguments

2 Documentation
- Function description
- Side effects
- Preconditions and postconditions
  - Guidelines
- Examples

What are “side effects”?

- A side effect is anything that is a direct consequent of the execution of a function that is not a direct mapping of inputs to outputs (return values).
- This could include output to the screen, changing of reference parameters, starting a robot, closing a valve, and a whole host of other possible actions that a program may control.

Outline

1 Default arguments

2 Documentation

Function description

Side effects

Preconditions and postconditions

Guidelines

Examples

Preconditions and postconditions

Preconditions

- Preconditions state what must be true of the inputs to the function before a function if executed in order that the postconditions can be guaranteed.

Postconditions

- Postconditions state what will be true after successful execution of a function assuming preconditions are met. This includes return values and any side effects.

Guidelines

- You may think of pre- and postconditions as setting up a contract between you (the programmer) and the user of your function. With this set of comments, you are saying, “follow these rules and I will guarantee these results. Don’t follow the rules, and I can’t promise anything.” You are absolved of all responsibility if the inputs to the function don’t adhere to the preconditions of the function.
- It is perfectly acceptable that there be no preconditions on a function, especially if there are no parameters! Just write “none”.

Guidelines

- Preconditions should never contain obvious, irrelevant, or redundant statements. For example, suppose that a parameter is of type int, you should not state that the argument sent "must be of type int". By itself, that is obvious since it is stated in the parameter list. On the other hand, I think it's acceptable to make a statement such as, "value must be an integer greater than 100...". This adds one word to an informative comment and emphasizes the point that the function is expecting an integer (at no cost).
- I think most often you will find that preconditions will be a statement of the range of values a parameter may take on. There are many ways to do this. You can use good English or some kind of mathematical expression. As long as it is clear to the reader, it's ok.

Guidelines

- When stating postconditions, be sure to include all pertinent information. The possible effects that the function may have on reference parameters are especially important for the user to understand.
- Don't state what is not a result of the function being executed.
- "none" is not a valid postcondition, ever
- All these comments should be placed just before the prototype of the function in question.

Outline

1. Default arguments

2. Documentation
   - Function description
   - Side effects
   - Preconditions and postconditions
     - Guidelines
   - Examples

greetings

```
// Description: The greetings() function will
// greeting the user.
// Pre: None
// Post: Message output to the screen.
void greetings ();
```

cyl_vol

```
// The cyl_vol() function will calculate and
// return the volume of the right circular
// cylinder with base radius rad and height ht
// Pre: Both parameters, rad and ht, must be
//      positive values.
// Post: Volume of cylinder is returned.
float cyl_vol (const float rad, const float ht);
```

swap

```
// The swap() function will swap the values
// of the arguments sent to it.
// Pre: none
// Post: The values of the arguments sent will
//       be swapped in the calling function.
void swap (float & val1, float & val2);
```

Go over more examples here

- `https://sites.google.com/a/mst.edu/price/home/good-code-vs-bad-code`
- `http://web.mst.edu/~cpp/cpp_coding_standard_v1_1.pdf`