Debugging

Michael Nowak

Texas A&M University

Acknowledgement: Lecture slides based on those created by Bjarne Stroustrup for use with his textbook

Overview

Overview

Introduction

- When you have written a program, it will have errors
  - It ll do something, but not what you expected
  - How do you find out what it actually does?
  - How do you correct it?
  - This process is called `debugging`

Overview

What not to do

- `while (program doesn t appear to work)`

What not to do

- `while (program doesn t appear to work)`
  - `randomly look at the program for something`

What not to do

- `while (program doesn t appear to work)`
  - `randomly look at the program for something`
  - `change it to ``look better''`

What not to do

- `while (program doesn t appear to work)`
  - `randomly look at the program for something`
  - `change it to ``look better''`
- Key question: how would I know if the program actually worked correctly?

Overview

Overview

Always write readable code

- Make the program easy to read so that you have a chance of spotting the bugs

Always write readable code

- Make the program easy to read so that you have a chance of spotting the bugs
  - Comment
    - Explain design ideas

Always write readable code

- Make the program easy to read so that you have a chance of spotting the bugs
  - Comment
    - Explain design ideas
  - Use meaningful `names`

Always write readable code

- Make the program easy to read so that you have a chance of spotting the bugs
  - Comment
    - Explain design ideas
  - Use meaningful `names`
  - Indent
    - Use consistent layout

Always write readable code

- Make the program easy to read so that you have a chance of spotting the bugs
  - Comment
    - Explain design ideas
  - Use meaningful `names`
  - Indent
    - Use consistent layout
  - Break code into small `functions`

Always write readable code

- Make the program easy to read so that you have a chance of spotting the bugs
  - Comment
    - Explain design ideas
  - Use meaningful `names`
  - Indent
    - Use consistent layout
  - Break code into small `functions`
  - Avoid complicated code sequences

Always write readable code

- Make the program easy to read so that you have a chance of spotting the bugs
  - Comment
    - Explain design ideas
  - Use meaningful `names`
  - Indent
    - Use consistent layout
  - Break code into small `functions`
  - Avoid complicated code sequences
  - Use library facilities

Overview

Get your program to compile

- Is every string literal terminated?
  - `std::cout << "Hello, << name << std::endl;`

Get your program to compile

- Is every string literal terminated?
  - `std::cout << "Hello, << name << std::endl;`
- Is every character literal terminated?
  - `std::cout << "Hello, " << name << '\n;`

Get your program to compile

- Is every string literal terminated?
  - `std::cout << "Hello, << name << std::endl;`
- Is every character literal terminated?
  - `std::cout << "Hello, " << name << '\n;`
- Is every block terminated?
  - ```
    if (a > 0) {
        /* do something */
    else {
        /* do something else */
    }
    ```
    ```

Get your program to compile

- Is every string literal terminated?
  - `std::cout << "Hello, << name << std::endl;`
- Is every character literal terminated?
  - `std::cout << "Hello, " << name << '\n;`
- Is every block terminated?
  - ```
    if (a > 0) {
        /* do something */
    else {
        /* do something else */
    }
    ```
- Is every set of parentheses matched?
  - ```
    if (a
        /* do something */
    ```

Get your program to compile

- Is every string literal terminated?
  - `std::cout << "Hello, << name << std::endl;`
- Is every character literal terminated?
  - `std::cout << "Hello, " << name << '\n;`
- Is every block terminated?
  - ```
    if (a > 0) {
        /* do something */
    else {
        /* do something else */
    }
    ```
- Is every set of parentheses matched?
  - ```
    if (a
        /* do something */
    ```
- The compiler generally reports these kinds of errors “late”
  - It doesn t know you didn t mean to close “it” later

Get your program to compile

- Is every `name` declared?
  - Did you include the needed headers?

Get your program to compile

- Is every `name` declared?
  - Did you include the needed headers?
- Is every name declared before it s used?
  - Did you spell all of the names correctly?
    - ```
      int count;
      /* do something */
      ++Count;
      ```
    - ```
      char ch;
      /* do something */
      Cin >> c;
      ```

Get your program to compile

- Is every `name` declared?
  - Did you include the needed headers?
- Is every name declared before it s used?
  - Did you spell all of the names correctly?
    - ```
      int count;
      /* do something */
      ++Count;
      ```
    - ```
      char ch;
      /* do something */
      Cin >> c;
      ```
- Did you terminate each expression statement with a semi-colon?
  - `x = sqrt(y) + 2`

Overview

Verify that your program works

- Carefully follow the program through the specified sequence of steps

Verify that your program works

- Carefully follow the program through the specified sequence of steps
  - Pretend you re the computer executing the program

Verify that your program works

- Carefully follow the program through the specified sequence of steps
  - Pretend you re the computer executing the program
  - Does the output match your expectations?

Verify that your program works

- Carefully follow the program through the specified sequence of steps
  - Pretend you re the computer executing the program
  - Does the output match your expectations?
  - If there isn t enough output to help, add a few debug output statements
    - `std::cerr << ``x == '' << x << ``, y == '' << y << std::endl;`

Verify that your program works

- Carefully follow the program through the specified sequence of steps
  - Pretend you re the computer executing the program
  - Does the output match your expectations?
  - If there isn t enough output to help, add a few debug output statements
    - `std::cerr << ``x == '' << x << ``, y == '' << y << std::endl;`
- See what the program specifies, not what you think it should say!

Verify that your program works

- When you write the program, insert some checks that variables have “reasonable values”

Verify that your program works

- When you write the program, insert some checks that variables have "reasonable values"
  - ```
    if (num_of_elements < 0)
        throw runtime_error(``impossible: negative
        number of elements'');
    ```
    ```

Verify that your program works

- When you write the program, insert some checks that variables have "reasonable values"
  - ```
    if (num_of_elements < 0)
        throw runtime_error(``impossible: negative
        number of elements'');
    ```
  - ```
    if (largest_reasonable < number_of_elements)
        throw runtime_error(``unexpectedly large
        number of elements'');
    ```

Verify that your program works

- When you write the program, insert some checks that variables have "reasonable values"
  - ```
    if (num_of_elements < 0)
        throw runtime_error(``impossible: negative
        number of elements'');
    ```
  - ```
    if (largest_reasonable < number_of_elements)
        throw runtime_error(``unexpectedly large
        number of elements'');
    ```
  - ```
    if (x < y)
        throw runtime_error(``impossible: x < y'');
    ```
    ```

Verify that your program works

- When you write the program, insert some checks that variables have "reasonable values"
  - ```
    if (num_of_elements < 0)
        throw runtime_error(``impossible: negative
        number of elements'');
    ```
  - ```
    if (largest_reasonable < number_of_elements)
        throw runtime_error(``unexpectedly large
        number of elements'');
    ```
  - ```
    if (x < y)
        throw runtime_error(``impossible: x < y'');
    ```
- Design these checks so that some can be left in the program even after you believe it to be correct
    ```

Verify that your program works

- When you write the program, insert some checks that variables have "reasonable values"
  - ```
    if (num_of_elements < 0)
        throw runtime_error(``impossible: negative
        number of elements'');
    ```
  - ```
    if (largest_reasonable < number_of_elements)
        throw runtime_error(``unexpectedly large
        number of elements'');
    ```
  - ```
    if (x < y)
        throw runtime_error(``impossible: x < y'');
    ```
- Design these checks so that some can be left in the program even after you believe it to be correct
- It s almost always better for a program to stop than to give wrong results
    ```

Verify that your program works

- Pay special attention to “end cases” (beginnings and ends)

Verify that your program works

- Pay special attention to “end cases” (beginnings and ends)
  - Did you initialize every variable?
    - Was that value reasonable/sensible?

Verify that your program works

- Pay special attention to “end cases” (beginnings and ends)
  - Did you initialize every variable?
    - Was that value reasonable/sensible?
  - Did the function get the right arguments?
    - Did the function return the right value?

Verify that your program works

- Pay special attention to “end cases” (beginnings and ends)
  - Did you initialize every variable?
    - Was that value reasonable/sensible?
  - Did the function get the right arguments?
    - Did the function return the right value?
  - Did you handle the first element correctly?
    - What about the last element?

Verify that your program works

- Pay special attention to “end cases” (beginnings and ends)
  - Did you initialize every variable?
    - Was that value reasonable/sensible?
  - Did the function get the right arguments?
    - Did the function return the right value?
  - Did you handle the first element correctly?
    - What about the last element?
  - Did you handle the empty case correctly?
    - No input provided?
    - No elements in the container?

Verify that your program works

- Pay special attention to "end cases" (beginnings and ends)
  - Did you initialize every variable?
    - Was that value reasonable/sensible?
  - Did the function get the right arguments?
    - Did the function return the right value?
  - Did you handle the first element correctly?
    - What about the last element?
  - Did you handle the empty case correctly?
    - No input provided?
    - No elements in the container?
  - Did you open your files correctly?

Verify that your program works

- Pay special attention to "end cases" (beginnings and ends)
  - Did you initialize every variable?
    - Was that value reasonable/sensible?
  - Did the function get the right arguments?
    - Did the function return the right value?
  - Did you handle the first element correctly?
    - What about the last element?
  - Did you handle the empty case correctly?
    - No input provided?
    - No elements in the container?
  - Did you open your files correctly?
  - Did you actually read that input?
    - Did you actually write that output?

Verify that your program works

- “If you can t see the bug, you re looking in the wrong place”

Verify that your program works

- “If you can t see the bug, you re looking in the wrong place”
  - It s easy to be convinced that you know what the problem is and stubbornly keep looking in the wrong place

Verify that your program works

- “If you can t see the bug, you re looking in the wrong place”
  - It s easy to be convinced that you know what the problem is and stubbornly keep looking in the wrong place
  - Don t just guess, be guided by output!

Verify that your program works

- “If you can t see the bug, you re looking in the wrong place”
  - It s easy to be convinced that you know what the problem is and stubbornly keep looking in the wrong place
  - Don t just guess, be guided by output!
    - Work forward through the code from a place you knowis right

Verify that your program works

- “If you can t see the bug, you re looking in the wrong place”
  - It s easy to be convinced that you know what the problem is and stubbornly keep looking in the wrong place
  - Don t just guess, be guided by output!
    - Work forward through the code from a place you knowis right
    - Work backwards from some bad output

Verify that your program works

- “If you can t see the bug, you re looking in the wrong place”
  - It s easy to be convinced that you know what the problem is and stubbornly keep looking in the wrong place
  - Don t just guess, be guided by output!
    - Work forward through the code from a place you knowis right
    - Work backwards from some bad output
  - Once you ve found the “the bug” carefully consider if fixing it solves the *whole* problem

Overview

Aside on error handling

- Error handling is fundamentally more difficult and messy than "ordinary code"
  - There is bascially just one way things can work right
  - There are many ways that things can go wrong

Aside on error handling

- Error handling is fundamentally more difficult and messy than "ordinary code"
  - There is bascially just one way things can work right
  - There are many ways that things can go wrong
- The more people use a program, the better the error handling must be

Aside on error handling

- Error handling is fundamentally more difficult and messy than "ordinary code"
  - There is bascially just one way things can work right
  - There are many ways that things can go wrong
- The more people use a program, the better the error handling must be
  - If you break your own code, that s your own problem

Aside on error handling

- Error handling is fundamentally more difficult and messy than "ordinary code"
  - There is bascially just one way things can work right
  - There are many ways that things can go wrong
- The more people use a program, the better the error handling must be
  - If you break your own code, that s your own problem
  - If your code is used by your friends, uncaught errors can cause you to lose friends

Aside on error handling

- Error handling is fundamentally more difficult and messy than "ordinary code"
  - There is bascially just one way things can work right
  - There are many ways that things can go wrong
- The more people use a program, the better the error handling must be
  - If you break your own code, that s your own problem
  - If your code is used by your friends, uncaught errors can cause you to lose friends
  - If your code is used by strangers, uncaught errors can cause serious grief

Overview

References

- Lippman, B., Lajoie, Josee, & Moo, B. E. (2016). *C++ primer* (5th ed.). Addison-Wesley.
- Stroustrup, B. (2014). *Programming: principles and practice using C++* (2nd ed.). Addison-Wesley.