do while loops and troubleshooting code

Comp Sci 1570 Introduction to C++

Outline

1. do while
   - Diagram
   - Syntax
   - Examples

2. Troubleshoot errors
   - Types of errors
     - Compile-time errors
     - Run-time errors
     - Logic errors
   - Debugging

do while

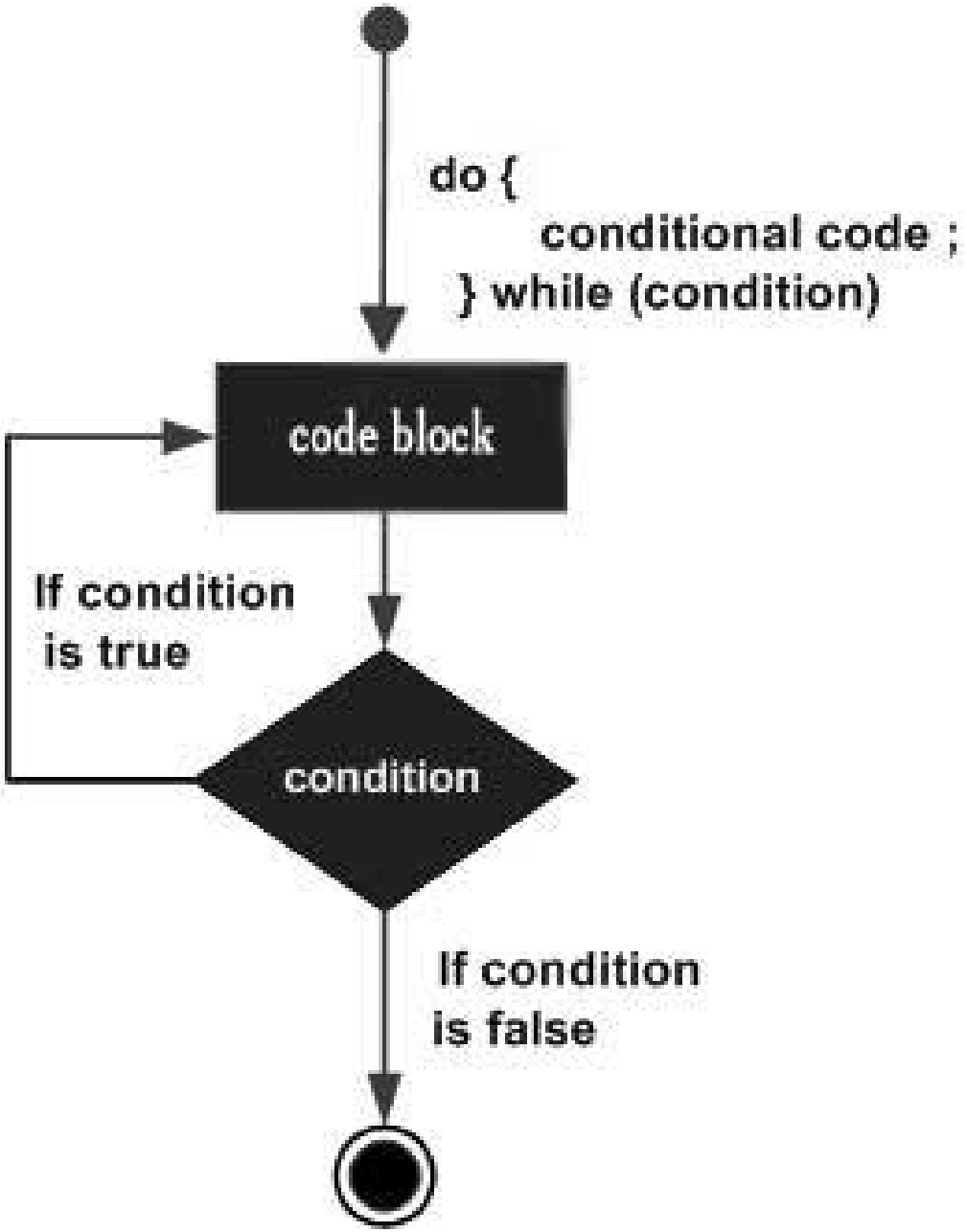
- If the while loop condition is initially false, the body of the loop will not execute at all.
- We may want a loop to execute at least once, such as when displaying a menu.
- A do-while loop is similar to a while loop, except that a do-while loop is guaranteed to execute at least one time.
- The do-while loop is usually preferred over a while-loop when the statement needs to be executed at least once, such as when the condition that is checked to end the loop is determined within the loop statement itself.

Outline

1 do while
Diagram
Syntax
Examples

2 Troubleshoot errors
Types of errors
Compile-time errors
Run-time errors
Logic errors
Debugging

Diagram



Outline

1 do while
Diagram
Syntax
Examples

2 Troubleshoot errors
Types of errors
Compile-time errors
Run-time errors
Logic errors
Debugging

Syntax

- do statement evaluates the body of the loop first and at the end, the condition is checked using while statement.
- General format of do-while loop is

```
do
{
   statement(s);
}
while(condition);
```

Outline

1 do while
Diagram
Syntax
Examples

2 Troubleshoot errors
Types of errors
Compile-time errors
Run-time errors
Logic errors
Debugging

Examples

Check out code

Outline

1 do while
Diagram
Syntax
Examples

2 Troubleshoot errors
Types of errors
Compile-time errors
Run-time errors
Logic errors
Debugging

Outline

1 do while
Diagram
Syntax
Examples

2 Troubleshoot errors
Types of errors
Compile-time errors
Run-time errors
Logic errors
Debugging

Compile-time errors

- A compile-time error results from the programmer's misuse of the language.
- A syntax error is a common compile-time error.
- The compiler can only translate a program if the program is syntactically correct; otherwise, the compilation fails and you will not be able to run your program.
- Syntax refers to the structure of your program and the rules about that structure.
- E.g., missing ;
- Try some

Run-time errors

- Run-time errors do not appear until you run the program.
- The compiler ensures that the structural rules of the C++ language are not violated.
- It can detect, for example, the malformed assignment statement and the use of a variable before its declaration.
- A program may not run to completion but instead terminate with an error.
- We commonly say the program "crashed."
- Examples include:
  - Mismatch of data type.
  - Array index out of range.
  - A number divided by zero.
  - An incompatible value input
- Try some

Logic errors and semantics

- If there is a logical error in your program, it may compile and run successfully, and the computer might not generate any error messages, but it will produce incorrect output.
- The problem is that the program you wrote is not the program you wanted to write.
- The meaning of the program (its semantics) is wrong.
- Errors that escape compiler detection (run-time errors and logic errors) are commonly called bugs.
- Since the compiler is unable to detect these problems, such bugs are the major source of frustration for developers.
- The frustration often arises because in complex programs the bugs sometimes only reveal themselves in certain situations that are difficult to reproduce exactly during testing.
- These come in infinite variety

Outline

1 do while
Diagram
Syntax
Examples

2 Troubleshoot errors
Types of errors
Compile-time errors
Run-time errors
Logic errors
Debugging

Debugging as diagnosis

- One of the most important skills you should acquire as a programmer is debugging.
- Debugging is like detective work, in that you are confronted with clues and you have to infer the processes and events that lead to the results you see.
- Debugging is also like an experimental science, in that once you have an idea what is going wrong, you modify your program and try again. If your hypothesis was correct, then you can predict the result of the modification, and you take a step closer to a working program. If your hypothesis was wrong, you have to come up with a new one.

Programming process

- Programming itself is can be seen as a process of gradually debugging a program until it does what you want.
- You should always start with a working program that does something, and make small modifications, debugging them as you go, so that you always have a working program.

Don't randomly change your code

- Debugging is accomplished by **gathering data** until you understand the cause of the problem.
- Debugging is accomplished by **comparing the data** that you have to what you know the data from a working system should look like.
- Do not change your code haphazardly trying to track down a bug. This is like a scientist who changes more than one variable at a time. It makes the observed behavior much more difficult to interpret, and you tend to introduce new bugs. Don't just change code and "hope" you'll fix the problem!
- Instead, make the bug reproducible, then use methodical "Hypothesis Testing":

Hypothesis testing

While(bug)
{
Ask, what is the simplest input that produces the bug?

Identify assumptions that you made about program operation that could be **false**.

Ask yourself "How does the outcome of this test/change guide me toward finding the problem?"

Use pen & paper to stay organized!
}

General principles

- Clues to what is wrong in your code exist in the values of your variables and the flow of control.
- If your code was working a minute ago, but now it doesn't what was the last thing you changed?
- Test your code as you, go rather than all at once.
- If you find some wrong code which does not seem to be related to the bug you were tracking, fix the wrong code anyway; the wrong code can be related to or obscured the bug in a way you had not imagined.
- Debugging depends on an objective and reasoned approach.
- It depends on overall perspective and understanding of the workings of your code. Be systematic.