C Branches

Kurt Schmidt
Skipjack Solutions
March 13, 2021
Objectives

Intended audience: Student who has working knowledge of Python
Target compiler: I’ll try to center the discussion on C99 using gcc 7.4

Objectives:

- To gain some experience with a statically-typed language
- To gain some experience with a compiled language
- To get practice in using pointers
  - To get practice with dynamically-allocated memory
  - Play around with linked lists
Intro
- We need to be able to execute conditionally
- We need to be able to make decisions
- To be *Turing complete* a language must allow for branches
  - (There are a couple other criteria)
- C provides *if*, *if-else*, a conditional expression (*?:*), and a *switch* (selection) statement
- Also, remember that the logical operators `&&` and `||` use short-circuit logic, so, provide for conditional execution, too
• Also, remember that the logical operators `&&` and `||` in C use short-circuit logic (also called "lazy evaluation"), so, provide for conditional execution, too

• The use of such short-circuit operators has been criticised:

  The conditional connectives – `cand` and `cor` for short — are ... less innocent than they might seem at first sight. For instance, `cor` does not distribute over `cand`; compare:

  \[
  (A \text{ cand } B) \text{ cor } C \equiv (A \text{ cor } C') \text{ cand } (B \text{ cor } C');
  \]

  in the case \( \neg A \wedge C \), the second expression requires \( B \) to be defined, the first one does not. Because the conditional connectives thus complicate the formal reasoning about programs, they are better avoided.

  – Edsger W. Dijkstra
• Code examples might have an accompanying link
  • Follow link to step through example at pythontutor.com
  • Does a nice job of graphically showing variables in memory, the heap, and the stack
If
if Syntax

if ( test )
    body

• Remember, C doesn’t care about newlines, nor indenting
  • Though your grade does
• Recall, also, 0 is false, all else is true
• The body is:
  • A single C statement, OR
  • A sequence of statements, inside curly braces

```c
if ( test )
{
    stmt1 ;
    stmt2 ;
    ... ;
    stmtm ;
}
```
Curly Braces & Style

Columns & Indenting

```c
if( test ) {
    ... ;
}

if( test ) {
    ... ;
}
```

- Opening curly brace, {, could be in either place
- Closing brace, }, should line up with the i in if
- Indent body (statements between braces) one level
Always Use Braces?

Don’t Trust Indenting!

• Many experience coders write a C statement like this
  • It’s fine
• But, consider debugging
• You want to add a print statement

```c
if( x>17 )
  i += 2 ;
printf( "x=%f\n", x ) ;
```

• For what values of \( x \) is the print statement evaluated?
  • \( All \) values \( x \)
  • Not part of the if-body
• Common, and maddening, bug
Another Common Error

Consider this if-stmt:
• Is always true, regardless of $x$
• Afterwards, $x$ is, invariably, 42

```c
if( x=42 )
    printf( "Found the answer!  x = %d\n, x ) ;
```

Maybe get in the habit of putting the literal (if there is one) first
• Won’t compile
• Trying to assign into a literal

```c
if( 42=x )
    printf( "Found the answer!  x = %d\n, x ) ;
```
If-Else
if( test )
  body
else
  body

• ! one body (consequent or alternate) will be evaluated
• Same rules for the body
Example if-else

```c
if( bSailing )
    puts( "Big smile\n" ) ;
else
{
    chocolate += 2 ;
    getMusic() ;
    makeKnots() ;
}
```
The Dangling else

- To resolve ambiguity, `else` is matched with the closest `if`
  - Don’t be fooled by indenting
  - Use curly braces to group statements
    - Also use them for clarity
- These are equivalent

```c
if( bRaining )
{
  if( bOutside )
    donOilies();
  else
    takeSunglasses();
}
```

- Note, I have my sunglasses if it’s raining, and if I’m inside
Condition Expression
test ? true-val : false-val

- An expression with two potential values
- Only ternary operator in C
- In C++, can *not* be overloaded (probably for the better)
- Low precedence
  - Just above assignment
- Can be nested
  - Please do not do this
Using ?: 

- This operator can be handy:
  ```c
  max = (a >= b) ? a : b;
  ...
  printf( "There are %d light%s\n", n, (n>1) ? "s" : "" );
  ```

- When used with simple, short expressions
- Note, the parentheses around the test are unneeded here

- We can quickly get messy:
  ```c
  lc = (!right) ? left : (right->data > left->data) ? right : left;
  ```

- Choose legibility over brevity
  ```c
  lc = left;
  if( right != NULL && right->data > left->data )
     lc = right;
  ```
The Switch Statement
The `switch` Statement

```c
switch var {
    case case1 : stmts1 ; break ;
    case case2 : stmts2 ; break ;
    ...
    default : stmts
}
```

- Provides for a *limited* expression of stacked if-else statements
- The variable being tested *must* be an integer type (`char`, `short`, etc.)
- Tests only for equality
- The `case` statement does *not* understand logical operators as you want
Example Using `switch`

```c
char opt;
...
switch opt {
    case 'c':
        bChocolate = true; break;
    case 'o':
        bStdout = false;
        fn = strdup(optarg);
        break;
    case 'H': /* fall through */
    case 'h': /* fall through */
    case '?':
        usage(); break;
    default:
        fprintf(stderr, "Bad option: %c. Exiting.\n", opt);
        usage();
        exit(1);
}
```
switch

Observations and Warnings

• Most likely matches should go high
• default should be last (always true)
• Once a label matches, the code immediately following is executed
  • Continues until a break is encountered
• Note, the ‘H’, ‘h’, and ‘?’ are stacked
  • All have the same result (same code is executed)
• As always, choose clarity over cleverness