Loops and Decisions

Chapter 3
Logical Expressions

Relational and logical operators – result is boolean-valued

- `==` equal to
- `!=` not equal to
- `>` greater than
- `<` less than
- `>=` greater than or equal to
- `<=` less than or equal to
- `&&` logical and
- `||` logical or
- `!` logical not

<table>
<thead>
<tr>
<th>E1</th>
<th>E2</th>
<th>E1 &amp; E2</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>

| E1 | E2 | E1 || E2 |
|----|----|----------|
| f  | f  | f       |
| f  | t  | t       |
| t  | f  | t       |
| t  | t  | t       |

<table>
<thead>
<tr>
<th>E</th>
<th>!E</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
</tr>
</tbody>
</table>
Flow of Control Statements

Three kinds

- Sequential
- Branches (also known as decisions)
  - if
  - if-else
  - switch
- Loops (also known as iterative)
  - for
  - while
  - do-while
Sequential Statements

All statements are executed once, one after another.
**if-Statement**

Executed if true

\[
\text{if (expression)} \\
\text{statement;}
\]

\[
\text{if (line33 < line34)} \\
\text{line37 = line36 - line35;}
\]

\[
\text{if (income >= 1400 || interest > 750)} \\
\{ \\
\text{must_file++;} \\
\text{deductions = 1;} \\
\text{adjusted_income = line37;}
\}
\]
Short Circuit Evaluation

Toward Compact and Efficient Code

- $E_1$ and $E_2$ are boolean-valued expressions
- if $(E_1 \&\& E_2)$
  - if $E_1$ is false, the whole expression is false and $E_2$ is not evaluated
- if $(E_1 \| E_2)$
  - if $E_1$ is true, the whole expression is true and $E_2$ is not evaluated

if $(n \neq 0 \&\& 100 / n > \text{min})$ if $(n \neq 0)$
  if $(100 / n > \text{min})$
**if-else Statement**

Choose One Statement

```c
if (expression)
    statement-1;
else
    statement-2;
if (line18 > line19)
    tax_owed = line18;
else
    refund = line19;
if (lineFlag)
    printf("%8d",lines);
```
True and False

An Existential Dilemma?

- Any expression may be interpreted as a boolean value
  - 0 is false
  - Non-0 is true
- The result of a “boolean” expression is either 1 or 0
- C++ defines type `bool` with possible values: `true` & `false`
  - Syntactic window-dressing for an int
  - `true = 1`
  - `false = 0`

```c
if (n % 2) {
    printf("n is odd\n");
} else {
    printf("n is even\n");
}
```

```c
if (strcmp(s1, s2)) {
    printf("s1 & s2 differ\n");
} else {
    printf("s1 & s2 are equal\n");
}
```
Nested Conditional Statement

Unlimited Nesting Depth

if (expression-1)
  if (expression-2)
    statement-1;
  else
    statement-2;
else
  if (expression-3)
    statement-3;
  else
    statement-4;
The Dangling `else` Problem

`else` attaches to the last `if`

```java
if (expression-1)
    if (expression-2)
        statement-1;
else
    statement-2;
```

```java
if (expression-1)
{
    if (expression-2)
        statement-1;
}
else
    statement-2;
```
if (expression-1)
  statement-1;
else if (expression-2)
  statement-2;
else if (expression-3)
  statement-3;
  ...
else if (expression-m)
  statement-m;
else
  statement-n
Ctype “Library”

Character Identification and Mapping

- **Using ctype macros**
  - Macros return 1 for true and 0 for false
  - `isascii( )` is defined on all integer values; the rest are defined only for integers representing characters, or EOF
  - `#include <ctype.h>`

- `int isalpha(int c)`
- `int isupper(int c)`
- `int islower(int c)`
- `int isdigit(int c)`
- `int isxdigit(int c)`
- `int isalnum(int c)`
- `int isspace(int c)`
- `int ispunct(int c)`
- `int isprint(int c)`
- `int isgraph(int c)`
- `int iscntrl(int c)`
- `int isascii(int c)`
- `int toupper(int c) /* function */`
- `int tolower(int c) /* function */`
Conditional Expressions

Based on the Conditional Operator `?:`

- `(expr 1) ? (expr 2) : (expr 3)`
  - If `expr 1` is true, `expr 2` is the value of the overall expression
  - If `expr 1` is false, `expr 3` is the value of the overall expression
  - Parentheses are not syntactically required
  - Typically used because `?` has a high precedence

- `max = (x > y) ? x : y;`
- `min = (x < y) ? x : y;`
- `index = (index+1 == size) ? 0 : ++index;`
switch Statement

Constant Integral Selection

switch(expression)
{
    case const-1:  statements-1;
                    break;
    case const-2:  statements-2;
                    /* fall-through */
    case const-3:
    case const-4:  statements-3;
                    break;
    case const-5:  break;
    ...
    default:       statements-m;
                    break;
}

CS 1410 Chapter 3
switch Example

Excerpt from \texttt{wc.c}

/* counts characters, words, and lines in a file */

\begin{verbatim}
switch(c)
{   case '\n': lines++;        /* fall through */
    case ' ': case '\t': inword = 0;
    break;
    default : if (! inword)     /* start word */
             { inword = 1;
               words++;  
             }
    break;
}
\end{verbatim}
for Loop

- Test at top
  - May not execute
- Any expression may be omitted
- Expression 1 is the initializer
  - Executed only once
- Expression 2 is the loop test
  - Loops while expression 2 is true
  - Tested after expr 1
  - Tested after expr 3
- Expression 3 is the update

for (expr-1; expr-2; expr-3)
statement;
for-Loop Examples

Simple Counting

```c
int i, j;

for (i = 0; i < 10; i++)
    printf("%d\n", i); /* prints 0 - 9 */

for (i = 0; i < 10; i += 2)
    printf("%d\n", i); /* prints 0, 2, 4, 6, 8 */

for (i = 0, j = 0; i < 10 && j < 5; i += 2, j++) /* comma operator */
    printf("%d\t%d\n", i, j);

for (i = 79; i >= 0 && s[i] == ' '; i--) /* empty statement */
    ;
```
for-Loop Variations

Non-“Standard” Loops

for (;;)
    statements; /* infinite loop */

for (i = 0; s[i] == ‘ ’; i++)
    statements; /* initializes i */

for (; s[i] != ‘\t’; i++)
    statements; /* empty init */

List* L;
for (L = root->next; L != root; L = L->next)
    statements; /* non-integer */

/* loop walks */
/* a linked list */
C++ for-Loops

Loop counter variable scope

- C requires that loop counter variables be defined outside of the loop in the variable definition area
- C++ did away with the variable definition area (i.e., variables may be defined anywhere within a program)
  - Loop counter variables may be defined in a for-loop
    ```cpp
    for (int i = 0; i < 100; i++) cout << i << endl;
    ```
- Loop counter variable scope
  - Old style: from the point of definition to the end of the block
  - New style: only in the for-loop itself
  - Microsoft Visual C++ 6.0 continues to use the old style
  - Microsoft .NET (7.0 and newer) implements the new style
C++ for-Loop Examples

The changing scope definition

- Second if – if (i >= 0)
  - okay in Microsoft and the old style-- i still in scope
  - fails the new style-- i went out of scope with the for

- Okay in all-- i is defined outside of loop and has block scope

- Second for loop
  - okay in the new style-- second i is a new variable
  - fails Microsoft and old style-- i is multiply defined

```cpp
for (int i = 0; i < 10; i++)
  cout << i << endl;
if (i >= 0)
  cout << "i is big\n";
```

```cpp
int i; 
for (i = 0; i < 10; i++)
  cout << i << endl;
if (i >= 0)
  cout << "i is big\n";
```

```cpp
for (int i = 0; i < 10; i++)
  cout << i << endl;
for (int i = 0; i < 10; i++)
  cout << i << endl;
```
While Loop

Controlled repetition

- Test at the top
  - May not be executed
- Loops while expression is true

```c
int n = 100;
while (n > 0)
    printf("%d\n", n--);
```

```c
while ((c = getopt(argc, argv, "cwl")) != EOF) { ... }
while ((c = fgetc(fp)) != EOF) { ... }
```
The `do-while` Loop

- **Test at the bottom**
  - Executed at least once
- **Loops while expression is true**
  - Opposite of Pascal’s `repeat-until`
- **Useful when the test expression is affected by a statement in the loop body**

```c
do
{
    statements;
} while (expression);
```
Compound Statement: while/switch

Excerpt from \texttt{wc.c}

```c
int c, charFlag = 0, wordFlag = 0, lineFlag = 0;

while ((c = getopt(argc, argv, "cwl")) != EOF)
    switch(c)
    {
    case 'c' : charFlag++; break;
    case 'w' : wordFlag++; break;
    case 'l' : lineFlag++; break;
    case '?' : fprintf(stderr, "ERROR: unknown option\n"); break;
    }
```
Loop Interruption

Used to simplify program structure

- **break**
  - Terminates the inner most loop (execution resumes with the statement following the loop)

- **continue**
  - Skips remaining code in inner loop (from the continue statement to the end of the loop)
  - starts next loop iteration
    - for loops resume at the update followed by the test
    - do and do-while loops resume at the test

- Usually in an if-statement
Loop Interruption Examples

Some Common Idioms

```c
for (avi = optind; avi < argc; avi++)
{   if ((fp = fopen(argv[avi], "r")) == NULL)
    { printf("ERROR: unable to open "%s" n", argv[avi]);
        continue;
    }
}

for (;;); /* from qsort partition */
{   while (a[++i] < v);
    while (a[--j] > v);
    if (j <= i) /* loop until i and j cross */
        break;
    temp = a[i];  a[i] = a[j];  a[j] = temp;
} 
```
The null-statement

Not a syntax error

- Uncommon but sometimes useful with for-loops
- Not so useful with other control statements

```java
if (expression) ;
for (ex1, ex2, ex3) ;
for (ex1, ex2, ex3) more clear
```
More library functions

- `exit(status)`
  - Terminates program
  - Returns exit status (0 is okay, non-zero is error)

- `getch()` and `getche()`
  - Returns a character as soon as the key is pressed (do not need to press the enter key)
  - `conio.h` header file
  - Non-ANSI functions (i.e., less portable than usual)
  - Available on Windows
  - Available in the Unix/Linux curses package
Debugging Control Statements

Where is your code going?

if (expression) {
    cout << "true" << endl;
    ....
} else {
    cout << "false" << endl;
    ....
}

for (int = 0; i < exp; i++) {
    cout << "loop: " << i << endl;
    ....
}
### Understanding Loops

Manually work a problem with tables

```java
double a = 0.0;
for (int i = 0; i < f(a); i++)
{
    int x = ....;
    int y = ....;
    a = f(x + i * y);
}
```

<table>
<thead>
<tr>
<th>a</th>
<th>i</th>
<th>x</th>
<th>y</th>
<th>x+i*y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0</td>
<td>5</td>
<td>14</td>
<td>36.78</td>
</tr>
<tr>
<td>36.78</td>
<td>1</td>
<td>19</td>
<td>8</td>
<td>95.03</td>
</tr>
</tbody>
</table>
The goto Statement

Unconditional Jump

- Must be used carefully
  > Can make “spaghetti” code if abused
  > Use in rare, limited situations
    - Interrupting nested loops
    - Creating error handling code
    - Implementing state machines
- May not jump into (i.e., goto) functions, loops, ifs, or switches

```plaintext
for (i = 0; i < 100; i++)
{
  for (j = 0; j < 100; j++)
  {
    ...
    if (error)
    {
      goto done;
    }
  }
}
done: ...;
```