SFU CMPT-212 2008-1 Topic: Control Statements

Ján Maňuch

E-mail: jmanuch@sfu.ca

Wednesday 23rd January, 2008

Control Statements

— control the flow of the program: which code parts of code are executed and how many times

- *conditional* (selection) statements:
  - `if...else`
  - `switch`
- *repetition* statements (loops)
  - `for`
  - `while`
  - `do...while`

Conditional Statements

- *if*

```
1 if ( condition )
2   statement
```

Example:

```
1   if ( grade >= 60 )
2     cout <<"Passed";
```

- *if . . . else*

```
1 if ( condition )
2   statement_true
3 else
4   statement_false
```

Example:

```
1   if ( grade >= 60 )
2     cout <<"Passed";
3   else
4     cout <<"Failed";
```

Compound Statements — Blocks

A *compound statement* or *block* is

- a sequence of statements enclosed in curly brackets (`{...}`).

Example:

```
1   if ( grade >= 60 ) {
2     passing++;
3     cout <<"Pased";
4   }
```

Note: variables declared inside of a block, exist only inside of the block

Example:

```
1   int x = 20;
2   {
3     cout << x << endl;  // prints 20
4     int x = 100;
5     cout << x << endl;  // prints 100
6   }
7   cout << x << endl;    // prints 20
```

Nested if ... else statements

Question: what's wrong with the following code?

```
1  if ( x>5 )
2    if ( y>5 )
3      cout << "x and y are > 5";
4  else
5    cout << "x is <= 5";
```

Nested if ... else statements

Answer: `else` is binding with the closest `if`, hence the correct formatting should be:

```
1 if ( x>5 )
2   if ( y>5 )
3     cout << "x and y are > 5";
4   else
5     cout << "x is <= 5";
```

Question: How to fix the code so that it does what intended?

Nested if ... else statements

Answer:

```
1  if ( x>5 ) {
2    if ( y>5 )
3      cout << "x and y are > 5";
4  } else
5    cout << "x is <= 5";
```

- *the conditional operator* `?:`

```
1  condition ? expression_if_true : expression_if_false
```

Example:

```
1    c = a>b ? a : b;
```

is equivalent to

```
1    if (a>b)
2      c = a;
3    else
4      c = b;
```

Note: `?:` operator with three operands forms an *expression*

More complicated example:

```
1    m = a>b ? a>c ? a : c : b>c ? b : c;
```

Conditions

- any expressions which evaluates to a `bool` value (`true` or `false`); Remember: C++ can automatically convert any non-zero integer value to `true` and zero integer value to `false`
- relational expressions: `<`, `<=`, `==`, `>`, `>=`, `!=`
- logical expressions: `||` (logical OR), `&&` (logical AND), `!` (logical NOT)

Comments:

- do not mix the assignment operator `=` and the is-equal-to operator `==`
- the expression on the right of the `||` operator is not evaluated if the expression on the left is true
- the expression on the right of the `&&` operator is not evaluated if the expression on the left is false

Comparing strings

Example:

```
1   char a[20];
2   cin >> a;
3
4   if (a == "password")
5     cout <<"Access granted."<<endl;
```

Doesn't work! What's wrong?

Example: compstr.cpp

Note: `strcmp(str1,str2)` returns

- < 0 if `str1` precedes `str2` in alphabetical order
- $= 0$ if strings `str1` and `str2` are identical
- > 0 if `str2` precedes `str1` in alphabetical order

switch

```
1  switch ( integer_expression ) {
2    case constant_1:
3     statements_1;
4     break;
5    case constant_2_1:
6    case constant_2_2:
7     statements_2;
8     break;
9  ...
10   default:
11    statements_default;
12 }
```

Note. *Not including* `break` *at the end of a* `case` *will cause the execution of all instructions until the next* `break` *or the end of the* `switch` *statement.*

Example: switch.cpp

Sample questions

- Assume that in the following statement `a`, `b`, `c` and `m` are integers:

```
1  m = a<b ? b<c ? b : a<c ? c : a : a<c ? a : b<c ? c : b;
```

  What is the value of `m` (depending on values of `a`, `b` and `c`) after executing of the statement? Write a code performing the same task without using the `?:` operator.

- What is the output of the following code?

```
1  int a, b=5;
2  cin >> a;
3  if ( (a==2) && (b=a) )
4    cout <<"ok ";
5  cout <<a<<" "<<b;
```

- Rewrite the following code without using `switch` statement:

```
1  switch (val)
2  {
3    case 2:
4      cout << "case 2\n";
5    case 3:
6      cout << "case 3\n";
7      break;
8    case 4:
9    case 5:
10     cout << "case 4\n";
11     break;
12   default:
13     cout << "switch default\n";
14 }
```

References

— where to find functions

- MSDN of MS Visual C++
- C++ language reference:

  http://www.cs.iastate.edu/~leavens/larchc++manual/lcpp_toc.html
- standard libraries: `http://www.cppreference.com/`
- standard libraries: `http://www.cplusplus.com/ref/` (not complete)
- some other libraries (e.g., for working with directories, signals, etc) can be found here:

  http://www.informit.com/guides/guide.asp?g=cplusplus&rl=1

Repetition statements (loops)

C++ accept three types of *loops*:

- *for* loop

```
for ( initialization; condition; update_statement )
  statement
```

- *while* loop

```
while ( loop_condition )
  statement
```

- *do . . . while* loop

```
do
  statement
while ( loop_condition );
```

For loop

```
1 for ( initialization ; condition ; update_statement )
2   statement
```

Steps:

1. *initialization* (typically set up a loop variable)
2. *test* the loop condition; if false exit, otherwise continue with next step
3. *execute* the statement of the loop
4. *update* (typically increment (decrement or otherwise modify) the loop variable)
5. continue with step 2.

Example: forloop.cpp [Prata]

Comments:

- the loop variable(s) can be initialized directly in the loop statement

  Example:

```
1 for (int i = 0; i < 5; i++)
2   cout << i <<" C++ knows loops.\n";
```

- according to C++ standard, variable `i` exists only inside of the loop

  Example 1: two consecutive loops (can be compiled on standard compliant compiler) (forlooptwice.cpp)

```
1 for (int i = 0; i < 5; i++)
2   cout << i;
3 for (int i = 0; i < 5; i++)
4   cout << i;
```

  Example 2: incorrect code (however, only this version can be compiled with MS Visual C++ 6.0)

```
1 for (int i = 0; i < 5; i++)
2   cout << i;
3 for (i = 0; i < 5; i++)
4   cout << i;
```

For loop and the comma operator

```
1 for ( initialization ; condition ; update_statement )
2   statement
```

Note: each `initialization`, `condition` and `update_statement` has to be one single statement or expression (you cannot use a block).

But you can use the *comma* operator "`,`" to use multiply statements:

```
i++, j--
```

- the expressions are evaluated from left to right
- the value of a comma expression is the value of the *last* expression

Example: `i=1, j=i+2, k=j+3` — will set `i` to 1, `j` to 3 and `k` to 6, and the value of the expression will be 6

Example: comma.cpp, forstr2.cpp [Prata]

While loops

- ```
  while (condition)
    statement
  ```

  is equivalent to

  ```
  for ( ; condition; )
    statement
  ```

- ```
  do
    statement
  while (condition);
  ```

  is an *exit-condition* loop. The statement gets executed at least once (disregard whether condition is true or not).

  Steps:

  1. *execute* statement
  2. *test* the condition; if false exit, otherwise continue with step 1.

Break, continue

Note. *Alternately, loops can be terminated using the* `break` *keyword, which will end the nearest enclosing* `switch` *or loop statement.*

Example:

```
1  int number,sum=0;
2  for (;;) {
3    cin >>number;
4    if (number<0)
5      break;
6    sum=sum+number;
7  }
```

Note. *The remaining part of the body of the loop can be skipped by using the* `continue` *keyword. The control proceeds with the next iteration (with testing the condition in* `while` *loops and with the update statement in a* `for` *loop).*

Example: continue.cpp

Sample questions

- Simulate the `for` loop with a `while` loop.
- What are the values of `i`, `j` and `k` after executing the following code?

```
1 int i,j,k;
2 i=(i=1, j=2*i+1, k=j-i);
```

- What is the difference between the `while` and `do...while` loops?
- Write a code which prints the following output:

```
  *
 ***
*****
```

  where the number of rows is `k`. In the example `k=3`. Use only two loops!

- What does `continue` statement do?