Control flow

Anna Simon

September 2, 2015

Control flow: introduction

There are several different type of control flow statements:

- conditional branches - changes the path of execution based on the value of an expression, e.g. `if`, `else`
- loops - repeatedly execute a series of statement, e.g. `for`, `while`
- jumps - causes the program to move to another statement, e.g. `goto`, `break`, `continue`
- halt - tells the program to quit , e.g. `exit()`

if statements

Syntax

```
 if (expression)
     statement

 if (expression)
       statement1
else
       statement2

 if (expression1)
       statement1
else if (expression2)
       statement2
else
       statement3
```

```cpp
int main()
{
    using namespace std;
    cout << "Enter a number: ";
    int nX;
    cin >> nX;

    if (nX > 10)
        cout << nX << "is greater than 10
            " << endl;
    else if (nX < 5)
        cout << nX << "is less than 5" <<
            endl;
    // could add more else if statements
        here
    else
        cout << nX << "is between 5 and 1
            0" << endl;

    return 0;
}
```

if statements

Use blocks, to execute multiple statements within one `if` condition (also useful when executing one statement: makes it more clear and easier to track)

```

int main()
{
    using namespace std;
    cout << "Enter a number: ";
    int nX;
    cin >> nX;

    if (nX > 10)
        {
        cout << "You entered " << nX <<
             endl;
        cout << nX << "is greater than 10
             " << endl;
        }
    else
        {
        cout << "You entered " << nX <<
             endl;
        cout << nX << "is not greater
             than 10" << endl;
        }

    return 0;
}
```

if statements

Use blocks {} to properly nest multiple `if` statements.

Dangling `else`

Which `if` statement is the `else` statement matched up with ?

```
    if (nX > 10)
        // it is bad coding style to nest if statements this way
        if (nX < 20)
            cout << nX << "is between 10 and 20" << endl;

        // who does this else belong to?
        else
            cout << nX << "is greater than 20" << endl;

    return 0;
}
```

switch statements

`switch` statement is equivalent to a chain of `if else` chains testing for equality of a single variable.

```cpp
char cDir;

if (cDir=='l')
        cout << "Go left" << endl;
else if (cDir=='r')
        cout << "Go right" << endl;
else if (cDir=='s')
        cout << "Go straight" << endl;
else if (cDir=='b')
        cout << "Go backwards" << endl;
else
        cout << "This is not a valid instruction" << endl;
```

```cpp
char cDir;

switch(cDir)
{
  case 'l':
      cout << "Go left" << endl;
      break;
  case 'r':
      cout << "Go right" << endl;
      break;
  case 's':
      cout << "Go straight" << endl;
      break;
  case 'b':
      cout << "Go backwards" << endl;
      break;
  default:
  //executed when non of the cases matched the tested variable
      cout << "This is not a valid instruction" << endl;
}
```

while loop

Syntax

```
while (expression)
    statement
```

while loop executes the statement as long as the condition is fulfilled.

```
int iii = 0;
while (iii < 10)
    {
    cout << iii << " ";
    iii++;
    }
cout << "done!";
```

Loops can be nested.

What is the output of the following code:

```
// Loop between 1 and 5
int iii=1;
while (iii<=5)
{
    // loop between 1 and iii
    int jjj = 1;
    while (jjj <= iii)
        cout << jjj++;

    // print a newline at the end
       of each row
    cout << endl;
    iii++;
}
```

for loop

Syntax

```
for (init-statement; expression1; expression2)
     statement
```

Equivalent while loop:

```
{
    init-statement;
    while (expr1)
    {
        statement;
        expr2;
    }
} // variables declared in init-statement go out of scope here
```

What does this code print out? (Off-by-one error)

```
for (int iii=0; iii < 10; iii++)
    cout << iii << " ";
```

for loop

Null statement

This loop will increment iii 10 times and then a null statement is executed, i.e. it does nothing.

```
for (int iii=0; iii < 10; iii++)
    ;
```

NOTE: a misplaced semicolon might be interpreted as a null statement:

```
if (nValue == 0);
    nValue = 1;
```

here, nValue will never be assigned the value of 1.

Multiple declarations

```
for (int iii=0, jjj=9; iii < 10; iii++, jjj--)
    cout << iii << " " << jjj << endl;
```

is NOT equivalent to nested loops:

```
for(int iii=0;iii<10;iii++)
{
    for(int jjj=9;jjj>=0;jjj--)
    {
        cout << iii << " " << jjj << endl;
    }
}
```

What is the difference in the outputs of the two examples?

goto statement (avoid at all cost!)

- goto statement causes the CPU to jump to a point in the code identified by a statement label
- the statement label has to precede the goto statement in the code
- results in a hard to read program (spaghetti code)
- can be replaced by loops resulting in more clearly written code

Rewrite the code using `if else`

```
#include <iostream>
#include <cmath>

int main()
{
    using namespace std;
tryAgain: // this is a statement
    label
    cout << "Enter a non-negative
        number";
    double dX;
    cin >> dX;

    if (dX < 0.0)
        goto tryAgain; // this is
            the goto statement

    cout << "The sqrt of " << dX <<
        " is " << sqrt(dX) <<
        endl;
}
```

break and continue

break

break causes a loop or a switch statement to terminate.

```
// count how many spaces the user
    has entered
int nSpaceCount = 0;
// loop 80 times
for (int ii=0; ii < 80; ii++)
{
// read a char from user
    char chChar = getchar();
// exit loop if user hits enter
    if (chChar == '\n')
        break;
// increment count if user entered
    a space
    if (chChar == ' ')
        nSpaceCount++;
}

cout << "You typed " << nSpaceCount
    << " spaces" << endl;
```

#include <cstdio> for getchar()

continue

continue jumps back to the top of the loop.

```
for (int iii=0; iii < 20; iii++)
{
    // if the number is divisible
        by 4, skip this iteration
    if ((iii % 4) == 0)
        continue;

    cout << iii << endl;
}
```

Careful with while loops!

```
int iii=0;
while (iii < 10)
{
    if (iii==5)
        continue;
    cout << iii << " ";
    iii++;
}
```

exit()

Requires `<stdlib.h>` header.
Terminates the process normally, performing the regular cleanup for terminating programs.
The variable passed by `exit()` in your program to the program running it (e.g. a bash script, a wrapper) can be used to execute follow up actions, e.g. error messages.

- `exit(0)` - indicates that there was no error
- `exit(1)` - the program could not be executed properly (e.g. an input file was missing)
- each type of error can be indicated by a different integer

```
//open input file
sprintf(fileName,"branchings_sorted.dat");
ifstream inFile(fileName);

if (!inFile){
   cout << "Couldn't open file " << fileName << ". Exiting now." <<
        endl;
   exit(1);
}
```