

Special Control Topics in C++

July 6, 2010

switch Statements

switch Statement

```
1 int x(10);
2 char input;
3 cout << "Enter a character:_" << flush;
4 cin >> input;
5 if( input == 'A' ) {
6     x += 10;
7 } else {
8     if( input == 'G' ) {
9         x -= 10;
10    } else {
11        if( input == 'L' ) {
12            x += 100;
13        } else {
14            x = 0;
15        }
16    }
17 }
18 if( x == 0 ) {
19     cout << endl << "You typed_" << input
20         << "_" << endl;
21 }
```

Use switch statements to avoid hard-to-read code like this!

They are usually faster than if-then-else, which matters when you're plowing through a million data points.

switch Details

```
1 switch( expression ) {
2     case constant_A :
3         // if( constantA == expression )
4         // curlys are (usually) optional
5         {
6             statement_A ;
7         }
8         break ;
9
10    case constant_B :
11        // if constant_B == expression
12        statements_B ;
13        break ;
14
15    default :
16        // if( expression != constant_A AND
17        //     expression != or _B )
18        // default is optional , but always
19        // goes at the end (proper style)
20        statements ;
21        break ;
22 }
23 // control jumps here when a break is
24 // encountered or none of the "cases" match.
25 post_break_statement ;
```

- ▶ expression must evaluate to something that can be cast (automatically, by compiler) to an integer type.
Which pretty much means an int, or a char.
- ▶ Each case argument must be a constant (a const declared variable, or a literal).
- ▶ Prefer switch statements to long winded if-then-else integer comparisons.
- ▶ Use break ;s! Falling through a case can be painful.

Falling Through case Blocks

```
1 switch( expression ) {
2     case constant_A :
3         // if( constantA == expression )
4         // curlys are (usually) optional
5         {
6             statement_A;
7         }
8         break;
9
10    case constant_B :
11        // if constant_B == expression
12        statements_B;
13        break;
14
15    default :
16        // if( expression != constant_A AND
17        //     expression != or _B )
18        // default is optional, but always
19        // goes at the end (proper style)
20        statements;
21        break;
22 }
23 // control jumps here when a break is
24 // encountered or none of the "cases" match.
25 post_break_statement;
```

```
1 switch( expression ) {
2     case constant_A :
3         // if( constantA == expression )
4         // curlys are (usually) optional
5         {
6             statement_A;
7         }
8         break;
9
10    case constant_B :
11        // if constant_B == expression
12        statements_B;
13        // HELP! I'M FALLING THROUGH
14
15    default :
16        // if( expression != constant_A )
17        //
18        // These statements are also
19        // executed if constant_B
20        statements;
21        break;
22 }
23 // control jumps here when a break is
24 // encountered or none of the "cases" match.
25 post_break_statement;
```

A Little Bitty Example

```
1 string input_name;
2 int input(-1);
3 cout << "Enter an integer > 0." << endl;
4 cin >> input;
5 if( input > 0 ) {
6     switch( input ) {
7         case 1 : input_name = "one" ; break;
8         case 2 : input_name = "two" ; break;
9         case 3 : input_name = "three" ; break;
10        case 4 : input_name = "four" ; break;
11        case 5 : input_name = "five" ; break;
12        case 6 : input_name = "six" ; break;
13        case 7 : input_name = "seven" ; break;
14        default :
15            input_name = "Sorry , I've only got 3 bits to spare." ;
16            break;
17    }
18    cout << input_name << endl ;
19 }
```

Practice!

```
1 int x(10);
2 char input;
3 cout << "Enter a character:_" << flush;
4 cin >> input;
5 if( input == 'A' ) {
6     x += 10;
7 } else {
8     if( input == 'G' ) {
9         x -= 10;
10    } else {
11        if( input == 'L' ) {
12            x += 100;
13        } else {
14            x = 0;
15        }
16    }
17 }
18 if( x == 0 ) {
19     cout << endl << "You typed_" << input
20         << "_" << endl;
21 }
```

Practice!

```
1 int x(10);
2 char input;
3 cout << "Enter_a_character:_" << flush;
4 cin >> input;
5 if( input == 'A' ) {
6     x += 10;
7 } else {
8     if( input == 'G' ) {
9         x -= 10;
10    } else {
11        if( input == 'L' ) {
12            x += 100;
13        } else {
14            x = 0;
15        }
16    }
17 }
18 if( x == 0 ) {
19     cout << endl << "You_typed_" << input
20         << " " << endl;
21 }
```

```
1 int x(0);
2 char input;
3 cout << "Enter_a_character:_" << flush;
4 cin >> input;
5 switch( input ) {
6     case 'A' :
7         // x=20 if input is A
8         x = 20;
9         break;
10    case 'L' :
11        // x=110 if input is L
12        x = 110;
13        break;
14    case 'G' :
15        // FALL THROUGH
16    default :
17        // Otherwise ,
18        // x=0 and we print this line.
19        cout << endl << "You_typed_"
20            << input << " " << endl;
21    break;
22 }
```

Ternary Operators

Ternary Operator

The Ternary or “Conditional” operator is a convenient way to write one-liner if-then-else statements:

if expr ? then clause : else clause ;

```
1 double x, y;
2 int quadrant;
3
4 cout << "Enter_the_coords_x_y:_ " << flush;
5 cin >> x >> y;
6
7 if( x >= 0 ) {
8     quadrant = y >= 0 ? 1 : 4;
9 } else {
10     if( y >= 0 ) {
11         quadrant = 2;
12     } else {
13         quadrant = 3;
14     }
15 }
16
17 cout << "Quadrant_" << quadrant << endl;
```

<<Interactive Program>>

RUN

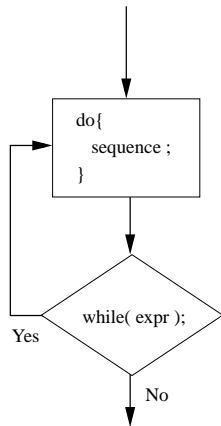
EDIT

ternary_example.cxx

do-while() Loops

The do-while() ; Loops

```
1 double input(0);
2
3 do {
4     cout << "Enter a decimal within "
5         << "(0,1):" << endl;
6     cin >> input;
7
8 } while( input<=0 || input>1 );
9
10 cout << "Thanks ," << input <<
11     " is within (0,1)." << endl;
12 /**
13  * note the spacing for readability in
14  * while() test
15  */
```



while == do-while?

Consider these two loops — do they run the same?

```
1 double sum(0);
2 double input(0);
3
4 cout << "Enter a (real) number: " << flush;
5 cin >> input;
6 while( input > 0 ) {
7     sum += input--;
8 }
9 cout << "Calculated sum: " << sum << endl;
```

```
1 double sum(0);
2 double input(0);
3
4 cout << "Enter a (real) number: " << flush;
5 cin >> input;
6 do {
7     sum += input--;
8 } while( input > 0 );
9 cout << "Calculated sum: " << sum << endl;
```

```
<<Interactive Program>>
```

RUN

EDIT

while_not_dowhile_while.cxx

```
<<Interactive Program>>
```

RUN

EDIT

while_not_dowhile_dowhile.cxx

continue in Loops

The `continue`; Keyword

`continue`; **Jumps** to the *end* of the innermost looping sequence “continues” from there.

continue; Example

Where is the repeating block?

```
1 int i(0);
2 do {
3     cout << i << "_Top_of_do-while_loop."
4         << endl;
5     i += 1;
6     if( i % 2 ) {
7         // avoid the rest of the loop
8         continue;
9     }
10    cout << i << "_Round_and_round_we_go!"
11        << endl;
12 } while( i < 10 );
13
14 cout << "Out_of_do-while_loop." << endl;
```

continue; Example

Where is the repeating block?

```
1 int i(0);
2 do {
3     cout << i << " _Top_of_do-while_loop."
4         << endl;
5     i += 1;
6     if( i % 2 ) {
7         // avoid the rest of the loop
8         continue;
9     }
10    cout << i << " _Round_and_round_we_go!"
11        << endl;
12 } while( i < 10 );
13
14 cout << "Out_of_do-while_loop." << endl;
```

```
0 Top of do-while loop.
1 Top of do-while loop.
2 Round and round we go!
2 Top of do-while loop.
3 Top of do-while loop.
4 Round and round we go!
4 Top of do-while loop.
5 Top of do-while loop.
6 Round and round we go!
6 Top of do-while loop.
7 Top of do-while loop.
8 Round and round we go!
8 Top of do-while loop.
9 Top of do-while loop.
10 Round and round we go!
Out of do-while loop.
```

RUN

EDIT

continue_example.cxx

finis