

cisc1110 fall 2010 lecture V.3

- *more control structures*
- switch statement
- break
- continue

switch control structure

- a switch statement is useful if you are making a choice between a number of options all concerning the value of a single, simple-typed variable
- it can replace multiple if-else-if-else... statements and tends to look neater
- but it can only replace multiple if-else-if-else... statements if the variable being compared in each statement is the same variable and it is of a simple data type (e.g., int, char, bool, etc.)

for example:

```
char c;
...
if ( c=='F' ) {
    y = y + 1;
}
else if ( c=='B' ) {
    y = y - 1;
}
else if ( c=='Q' ) {
    q = true;
}
else {
    cout << "oops!\n";
}
```

which can be replaced with:

```
char c;
...
switch ( c ) {
    case 'F':
        y = y + 1;
        break;
    case 'B':
        y = y - 1;
        break;
    case 'Q':
        q = true;
        break;
    default:
        cout << "oops!\n";
        break;
} // end of switch
```

- note the new keywords:

- switch which begins the statement and indicates the name of the variable you want to compare
- case which indicates the value that you want to compare the variable to
- break which ends the clause that gets executed for each matching "case", i.e., when the value of the variable matches that specified in the enclosing case
- default, which specifies the "default" case, when the value of the variable does not match any of those in the specified cases

- note that if you don't use a break command, then the program control will keep going at the end of one case and go into the code for the next case
There are times when you want this behavior, but most of the time you don't.
Here's an example of a case when you would want this behavior:

```
switch ( c ) {
    case 'Q':
    case 'q':
        q = true;
        break;
} // end of switch
```

break statement

- the break statement can also be used inside a loop (either for or while loops)
- the break statement causes the loop to stop executing and shifts program control to the end (outside) of the loop
- for example:

```
for ( int i=1; i<11; i++ ) {  
    cout << "I am " << i << endl;  
    if ( i % 3 == 0 ) {  
        break;  
    }  
    cout << "and I am NOT divisible by 3!\n";  
}  
cout << "goodbye" << endl;
```

and the output will be:

```
I am 1  
and I am NOT divisible by 3!  
I am 2  
and I am NOT divisible by 3!  
I am 3  
goodbye.
```

- the program will stop the loop as soon as the condition (`i % 3 == 0`) is true. the next line executed is outside of the loop, where "goodbye" is displayed.

continue statement

- the continue statement can also be used inside a loop (also for either for or while loops)
- the continue statement causes the loop to stop executing and shifts program control back to the beginning (inside) the loop
- for example:

```
for ( int i=1; i<11; i++ ) {  
    cout << "I am " << i << endl;  
    if ( i % 3 == 0 ) {  
        continue;  
    }  
    cout << "and I am NOT divisible by 3!\n";  
}  
cout << "goodbye" << endl;
```

and the output will be:

```
I am 1  
and I am NOT divisible by 3!  
I am 2  
and I am NOT divisible by 3!  
I am 3  
I am 4  
and I am NOT divisible by 3!  
I am 5  
and I am NOT divisible by 3!  
I am 6  
I am 7  
and I am NOT divisible by 3!  
I am 8  
and I am NOT divisible by 3!  
I am 9  
I am 10  
and I am NOT divisible by 3!  
goodbye.
```

- the program will jump to the next iteration in the loop as soon as the condition (`i % 3 == 0`) is true.