SOUTHERN ADVENTIST UNIVERSITY
School of Computing

Other Conditional and Iterative Statements

Chapter 7

For Next Time

- Read Chapter 7

SOUTHERN ADVENTIST UNIVERSITY
School of Computing

`switch` Statement

- An alternative to some multiway **`if/else`** statements (some restrictions apply)
- Potentially more efficient than a multiway **`if/else`**

```
switch ( integral expression )
{
   body
}
```


`if/else` vs. `switch`

```
if( value == 1 )
    cout << "one";
else if ( value == 2 )
    cout << "two";
else if ( value == 3 )
    cout << "three";
```

```
switch ( value )
{
case 1:
    cout << "one";
    break;
case 2:
    cout << "two";
    break;
case 3:
    cout << "three";
    break;
}
```

SOUTHERN ADVENTIST UNIVERSITY School of Computing

`switch` Body

- Body consists of multiple sections designated by **`case`** labels
- Execution jumps to the first **`case`** label that matches the **`switch`**'s expression
- The **`break`** statement terminates execution within the **`switch`** body

```
switch ( value )
{
case 1:
    cout << "one";
    break;
case 2:
    cout << "two";
    break;
case 3:
    cout << "three";
    break;
}
```

SOUTHERN ADVENTIST UNIVERSITY
School of Computing

`default` Label

- The optional **`default`** label is a "catch all" label that matches the expression only if none of the **`case`** labels match

```
switch ( value )
{
case 1:
    cout << "one";
    break;
case 2:
    cout << "two";
    break;
default:
    cout << "other";
    break;
}
```

SOUTHERN ADVENTIST UNIVERSITY
School of Computing

`switch` Restrictions

- value must be an integral expression
  - **`int`**, **`bool`**, **`char`**, etc.
- **`case`** labels must be integral constants
  - Symbolic or literal
- Not every multiway if/else can be expressed directly with a **`switch`**

```
switch ( value )
{
case 1:
    cout << "one";
    break;
case 2:
    cout << "two";
    break;
case 3:
    cout << "three";
    break;
}
```

SOUTHERN ADVENTIST UNIVERSITY
School of Computing

Direct Translation

```
if( value == 1 )
    cout << "one";
else if ( value == 10 )
    cout << "ten";
else if ( value == -1 )
    cout << "minus one";
else
    cout << "other";
```

```
switch ( value )
{
case 1:
    cout << "one";
    break;
case 10:
    cout << "ten";
    break;
case -1:
    cout << "minus one";
    break;
default:
    cout << "other";
}
```

SOUTHERN ADVENTIST UNIVERSITY
School of Computing

Direct Translation?

```
if ( x == y )
{
    z = 2*x + 1;
    w = t;
}
else if ( x > 2 )
{
    x++;
    y = 2*x + t;
}
else if ( y == 3 )
    x = y;
```

```
switch ( ? )
{

        ?


}
```

SOUTHERN ADVENTIST UNIVERSITY
School of Computing

Omitting the **`break`** statement

```
cin >> key; // get key from user
switch ( key )
{
case 'p':
case 'P':
    cout << "You choose \"P\"" << endl;
    break;
case 'q':
case 'Q':
    done = true;
    break;
}
```

SOUTHERN ADVENTIST UNIVERSITY
School of Computing

Conditional Operator

- Embeds the **`if/else`** concept within an expression

```
if ( z == 0 )
    x = 0;
else
    x = y/z;
```

```
x = (z == 0)? 0 : y/z;
```

SOUTHERN ADVENTIST UNIVERSITY
School of Computing

Conditional Operator

condition **?** ***expression*** $_1$ **:** ***expression*** $_2$

- ***condition*** is a normal Boolean expression as might appear in an if
  - Parentheses around the ***condition*** are not required, but should be used to improve the readability.
- ***expression*** $_1$ the overall value of the conditional expression if the ***condition*** is true.
- ***expression*** $_2$ the overall value of the conditional expression if the ***condition*** is false.

SOUTHERN ADVENTIST UNIVERSITY
School of Computing

Conditional Operator

```
if ( z == 1 )
    cout << "one";
else
    cout << "not one";
```

```
cout << (z == 1)?
        "one" : "not one";
```

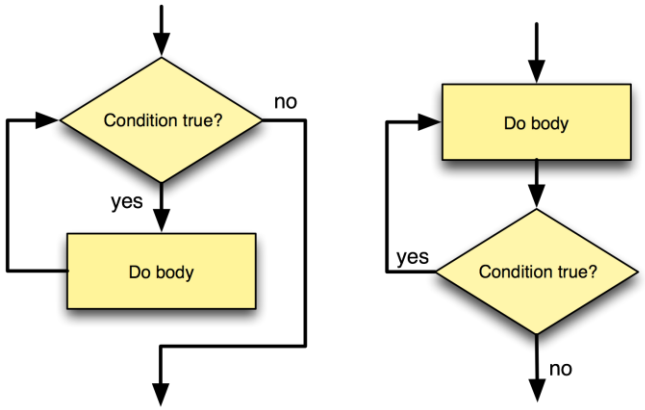
```
// Compute absolute value
if ( z < 0 )
    abs = -z;
else
    abs = z;
```

```
// Compute absolute value
abs = (z < 0)? -z : z;
```


`do/while` Statement

```
do
   body
while( condition );
```

`while` vs. `do/while`



`while` flowchart

`do/while` flowchart

SOUTHERN ADVENTIST UNIVERSITY
School of Computing

`for` Loop

`for` **`(`** *initialization* **`;`** *condition* **`;`** *modification* **`)`**

body

- The reserved word **`for`** identifies a **`for`** statement
- The header, contained in parentheses, contains three parts, each separated by semicolons:
  - The initialization part assigns an initial value to the loop variable; it is performed one time
  - The condition part is a Boolean expression, just like the condition of **`while`** and **`do/while`**; it is checked each time before the body is executed
  - The modification part changes the loop variable; it is performed during each iteration *after the body is executed*
- The body is like the body of any other loop

SOUTHERN ADVENTIST UNIVERSITY
School of Computing

`for` Loop

- The `for` statement should be used for counting
- All the information about the loop control is conveniently packaged in one location
- The loop variable should *not be modified inside the body*

Next . . .

Using functions