switch statement

• syntax
• example
• vs. else-if
switch statement

- Captures a particular common form of nested-if pattern

- Name “switch” suggests selecting an option based on a central setting
switch syntax

switch ( <case-selector-expr> )
{
    case <label1> : <statement(s)>1>
    case <label2> : <statement(s)>2>
    ...
    case <labelN> : <statement(s)>N>
    default : <statements(s)>M>
}

integral

unique

constant

optional
Nested If Example

int vehicleClass;  double toll;  string type;
cout << “Enter vehicle class: “;
cin >> vehicleClass;
if ( vehicleClass == 1 ) {
    type = “Passenger car”;
    toll = 0.50;
} else if ( vehicleClass == 2 ) {
    type = “Bus”;  
    toll = 1.50;
} else if ( vehicleClass == 3 ) {
    type = “Truck”;  
    toll = 2.00;
} else {
    type = “Unknown vehicle class!”;
}
if (vehicleClass == 1) {
    type = "Passenger car";
    toll = 0.50;
} else if (vehicleClass == 2) {
    type = "Bus";
    toll = 1.50;
} else if (vehicleClass == 3) {
    type = "Truck";
    toll = 2.00;
} else {
    type = "Unknown vehicle class!";
}

switch (vehicleClass)
{
    case 1:  type = "Passenger car";
            toll = 0.50;
            break;
    case 2:  type = "Bus";
            toll = 1.50;
            break;
    case 3:  type = "Truck";
            toll = 2.00;
            break;
    default:
        type = "Unknown vehicle class!";
}

- Used to exit a switch statement
- (Also exits other control structures, such as loops, …)

```java
switch ( vehicleClass ) {
  case 1:
    type = "Passenger car";
    toll = 0.50;
    break;
  case 2:
    type = "Bus";
    toll = 1.50;
    break;
  // ...}
```

With no “break”, the switch statement continues to the code in the next case!
vehicleClass = 1;
switch ( vehicleClass )
{
    case 1 : type = "Passenger car";
            toll = 0.50;
    case 2 : type = "Bus";
            toll = 1.50;
    case 3 : type = "Truck";
            toll = 2.00;
            break;
    default : type = "Unknown vehicle class!";
}
    cout << type << endl;

What prints?

Truck
vehicleClass = 1;
switch ( vehicleClass )
{
    case 1 : cout << “Passenger car” << endl; 
toll = 0.50;
    case 2 : cout << “Bus” << endl;
            toll = 1.50;
    case 3 : cout << “Truck” << endl;
            toll = 2.00;
            break;
    default : cout << “Unknown vehicle class!” << endl;
}

What prints?

Passenger car
Bus
Truck
Switch vs. if-then-else

• Efficiency
  • switch typically runs faster
  • Not a big deal

• Generality
  • switch syntax limits applicability
    • (many things that if can do that switch can’t)
    • can’t switch on doubles, ranges, strings, …

• Readability
  • switch table format very clear
if or switch?

• Program is given a heart rate

• Needs to print out a different message depending on a range:
  – i.e. “Too High” if rate is > 100
  – “Normal” if rate is between 60 and 100
  – etc.

• Should we use if or switch?
Why not use switch here?

Case labels must be single constants. Consider one range:
   60 through 100
How would we represent it?

case 100:
case 99:
case 98:
   . . . (the other 37 cases!)
case 60: cout << "Normal range";
Choosing If

if ( ( 0 <= heartRate ) && ( heartRate <= 200 ) )
    if ( heartRate > 100 )
        cout << "Too high";
    else if ( heartRate >= 60 )
        cout << "Normal range";
    else if ( heartRate > 0 )
        cout << "Too low";
    else
        cout << "Too late";

why not "else if ((heartRate >= 60) && (heartRate <= 100))"?
What is the output of the following code fragment? (beta is of type int.)

beta = 5;
switch (beta) {
    case 1 : cout << 'R';
            break;
    case 2 :
    case 4 : cout << 'O';
            break;
    case 5 : cout << 'L';
}
cout << 'X';

A)X  
B)LX  
C)RX  
D)OX  
E)ROLX

Ans:B
What is the output of the following code fragment?
(beta is of type int.)

beta = 2;
switch (beta) {
    case 1 : cout << 'R';
        break;
    case 2 :
    case 4 : cout << 'O';
        break;
    case 5 : cout << 'L';
}
cout << 'X';

A)X  B)LX  C)RX  D)OX  E)ROLX
Ans:D
What is the output of the following code fragment? (beta is of type int.)

beta = 2;
switch (beta) {
    case 1: cout << 'R'; break;
    case 2: cout << 'Q';
    case 4: cout << 'O'; break;
    case 5: cout << 'L';
}
cout << 'X';

A)X
B)QX
C)QOX
D)QO
E)RQOLX
Ans:C
What is the output of the following code fragment? (beta is of type int.)

beta = 5;
do {
    switch (beta) {
        case 1 : cout << 'R';
            break;
        case 2 :
        case 4 : cout << 'O';
            break;
        case 5 : cout << 'L';
    }
    beta--;
} while (beta > 1);
cout << 'X';

A)X
B)ROOLX
C)LOOX
D)LOORX
E)ROOX
Ans:C
```cpp
int diameter;
... 
cin >> diameter;
... 
if (diameter == 24) {
    numQuarters = numQuarters + 1;
} else if (diameter == 21) {
    numNickels = numNickels + 1;
} else if (diameter == 19) {
    numPennies = numPennies + 1;
} else if (diameter == 18) {
    numDimes = numDimes + 1;
} else {
    cout << "Not a correct diameter  
    for a US coin. " << endl;
}
```
Coin Sorter

```cpp
int diameter;
...
cin >> diameter;
... if (diameter == 24) {
    numQuarters = numQuarters + 1;
} else if (diameter == 21) {
    numNickels = numNickels + 1;
} else if (diameter == 19) {
    numPennies = numPennies + 1;
} else if (diameter == 18) {
    numDimes = numDimes + 1;
} else {
    cout << "Not a correct diameter for a US coin. " << endl;
}
```

```cpp
int diameter;
...
cin >> diameter;
... switch (diameter) {
    case 24: numQuarters += 1; break;
    case 21: numNickels += 1; break;
    case 19: numPennies += 1; break;
    case 18: numDimes += 1; break;
    default: cout << "Not a correct diameter for a US coin. " << endl;
}
```
for loop

• *for* iteration construct
• Syntax, semantics, usage
• vs. *while*
for loop

• Typical usage: count-controlled loops
  – provides convenient specification of counter, increment and test

• Another general iteration construct
for loop : SYNTAX

Initialize → Test → Update

for ( <expr1>; <expr2>; <expr3> )

<statement>

Loop body
Initialize

Test

Body

Update

EXIT

FALSE

TRUE
"Table of Squares" example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>98</td>
<td>9604</td>
</tr>
<tr>
<td>99</td>
<td>9801</td>
</tr>
<tr>
<td>100</td>
<td>10000</td>
</tr>
</tbody>
</table>
i = 1;
while ( i <= 100 )
{
    square = i * i;
    cout << i << " " << square << endl;
    i++;
}

for

i = 1;
while ( i <= 100 )
{
    square = i * i;
    cout << i << "    " << square << endl;
    i++;
}

// -----------------------------------------

for ( i = 1; i <= 100; i++ )
{
    square = i * i;
    cout << i << "    " << square << endl;
}
for ( i = 1; i <= 100; i++ )
{
    square = i * i;
    cout << i << "  " << square << endl;
    i++;  
}

What if we ... increment i within Loop Body?
What if we ... increment i within Loop Body?

```cpp
for ( i = 1; i <= 100; i++ )
{
    square = i * i;
    cout << i << "   " << square << endl;
    i++;  
}
```

// ----------------------------------------

```cpp
i = 1;
while (i <= 100)
{
    square = i * i;
    cout << i << "   " << square << endl;
    i++;  
    i++;  
}
```
Common Usage & Variations

- standard interpretations
  - But any exprs allowed

```plaintext
for ( <exp1>; <exp2>; <exp3> )
<statement>
```
Initialization Statement

for ((<expr1>; <exp2>; <exp3>))
  <statement>

- Can start counter at any integer
- Use a *comma expression* to initialize more than one variable, as in
  for (count = 1, sum = 0; <expr2>; <expr3>)
- Can declare counter as in
  for (int count = 1; <expr2>; <expr3>)
Update Expression

for (<exp1>; <exp2>; <exp3>)
<statement>

• Optional:
  for (<exp1>; <exp2>; )
• Can decrement, as in:
  for (<exp1>; <exp2>; count--)
• Count up (or down) by any values as in:
  for (<exp1>; <exp2>; count = count + 5)
Generality

- All elements optional (i.e., null expressions allowed)

- Loop body
  - Any statement
  - Altering counter inside loop body is allowed, but not recommended
Characters as Counters

for (char ch = 'a'; ch <= 'z'; ch++) {
    cout << ch;
}

prints (assuming the ASCII character set):

abcdef...wxyz
False Conditions

• Loop body may execute zero or more times

```cpp
int n = 3;

for (i = 10; i <= n; i++) {
    cout << i << endl;               // body never
}                                     // executes
```
const int MAX = 9;

int row;

for (row = 1; row <= MAX; row++) {
    cout << row << endl;
}

What is Output?
What is Output?

const int MAX = 9;

int row;

for (row = 1; row <= MAX; row++) {
    cout << row << endl;
}

Pattern: 1 to N loop

1 2 3 4 5 6 7 8 9
Choosing a loop structure

Is the loop count-controlled?

“for” usually best

Is the loop event-controlled?

“while” or “do while”

Ok for the loop body to execute at least once?

if not, choose “while” or “for”

Multiple termination conditions?

choose “while” or “do while”
What are the values of sum and i at the end of this code?

const int TOTAL = 10;
int i = 4, sum = 0;

for ( ; i <= (TOTAL - 1); i++ ) {
    if ( ( TOTAL / ( sum + 1 ) ) == 1 )
        sum += 3;
    else
        sum += 1;
}

A. sum = 3, i = 1
B. sum = 9, i = 9
C. sum = 10, i = 10
D. sum = 12, i = 10
E. sum = 8, i = 10

Ans: E
What is the output of the following code fragment?

```cpp
int limit = 8;
cout << 'H';
for ( int i = 10; i <= limit; i++ )
{
    cout << 'E';
}
cout << "LP";
```

A) HLP  
B) HELP  
C) HEELP  
D) HEEELP  
E) none of the above  
Ans: A
Example

```cpp
int t = 0;
int n;
cin >> n;

for ( int i = 0; i < n; i ++ ) {
  t += i;
}

cout << t << endl;
```
Example

```cpp
int t = 0;
int n;
cin >> n;

for (int i = 0; i < n; i++) {
    t += i;
}

cout << t << endl;
// cout << i << endl; // i is out of scope here
```
Example

```cpp
int t = 0;
int n;
cin >> n;
int i;
for (i = 0; i < n; i++) {
    t += i;
}

cout << t << endl;
cout << i << endl; // i is in scope here
```
Example

```cpp
int t = 1;
int n;
cin >> n;

for ( int i = 0; i < n; i ++ ) {
    t *= i;
}

cout << t << endl;
```
Example

```cpp
int t = 1;
int n;
cin >> n;

for ( int i = 1; i < n; i ++ ) {
    t *= i;
}

cout << t << endl;
```
Example

```cpp
int t = 1;
int n;
cin >> n;

for ( int i = 1; i <= n; i ++ ) {
    t *= i;
}

cout << t << endl;
```
Example

```cpp
int t = 1;
int n;
cin >> n;

for ( int i = 1; i <= n; i ++ ) {
    t *= n;
}

cout << t << endl;
```