If/else & switch

Unit 3

Sections 4.1-12, 4.14-15

CS 1428
Fall 2017

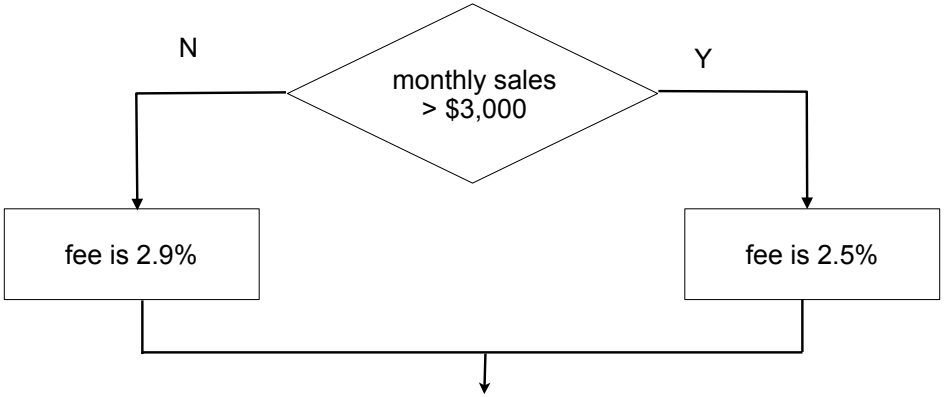
Jill Seaman

Straight-line code

- So far all of our programs have followed this basic format:
  - Input some values
  - Do some computations
  - Output the results
- The statements are executed in a sequence, first to last.

Decisions

- Sometimes we want to be able to decide which of two statements to execute:



Relational Expressions

- Making decisions require being able to ask “Yes” or “No” questions.
- Relational expressions allow us to do this.
- Relational expressions evaluate to **true** or **false**.
- Also called:
  - logical expressions
  - conditional expressions
  - boolean expressions

Relational Expressions

- Boolean literals:

```
true
false
```

`true` evaluates to true | `false` evaluates to false

- Boolean variables

```
bool isPositive = true;
bool found = false;
```

`isPositive` evaluates to true
`found` evaluates to false

4.1 Relational Operators

- Binary operators used to compare expressions:
  - `<` Less than
  - `<=` Less than or equal to
  - `>` Greater than
  - `>=` Greater than or equal to
  - `==` Equals (note: do not use =) !!
  - `!=` Not Equals

Relational Expressions

- Examples:

```
int x=6;
int y=10;

a. x == 5          evaluates to ___false___
b. 7 <= x + 2      evaluates to _____
c. y – 3 > x       evaluates to _____
d. x != y          evaluates to _____
d. true            evaluates to ___true____
```

- Can assign relational expressions to variables:

```
bool isPositive;
int x;
cin >> x;
isPositive = x > 0;
```

if the user types: 25
`isPositive` stores the value _____

Relational Operator Precedence

- Relational operators are LOWER than arithmetic operators:

```
int x, y;

...  x < y -10 ...       // minus happens first
...  x * 5 >= y + 10 ... // mult, then plus, then >=
```

- Relational operators are HIGHER than assignment:

```
int x, y;
...
bool t1 = x > 7;            // > then =
bool t2 = x * 5 >= y + 10;  // *, +, >=, =
```

4.4 if-else statement

- if-else statement is used to make decisions

```
if (expression)
   statement1
else
   statement2
```

statement1 and statement2 are called branches

- `expression` is evaluated
  - If it is true, then `statement1` is executed. (`statement2` is skipped).
  - If it is false, then `statement2` is executed (`statement1` is skipped).

if-else example

```
double rate;
double monthlySales;

cout << "Enter monthly sales last month: " ;
cin >> monthlySales;

if (monthlySales > 3000)
   rate = .025;
else
   rate = .029;

double price;
cout << "Enter selling price of item: " ;
cin >> price;
double commission = (price + 3.99) * rate;
cout << "Commission: $" << commission << endl;
```

```
Enter monthly sales last month: 3025
Enter selling price of item: 100
Commission: $2.59975
```


if-else structure

Notice:

```
if (monthlySales > 3000)
   rate = .025;
else
   rate = .029;
```

- relational expression is in parentheses
- NO semi-colon after expression, nor the else
- Good style: indent the statements!!
- The semi-colons belong to the statements, not to the if-else

4.3 The block statement

- a block (or a compound statement) is a set of statements inside braces:

```
{  int x;
   cout << “Enter a value for x: “ << endl;
   cin >> x;
   cout << “Thank you.” << endl;
}
```

- This groups several statements into a single statement.
- This allows us to use multiple statements when by rule only one is allowed.

if-else with blocks

- We can use blocks to put more than one statement in the branches of the if-else:

```
int number;
cout << "Enter a number" << endl;
cin >> number;

if (number % 2 == 0)
{
   number = number / 2;
   cout << "Even ";
}
else
{
   number = number + 1;
   cout << "Odd ";
}
cout << number << endl;
```


4.2 if statement

- The else part of the if-else stmt is optional:

```
if (expression)
   statement
```

- `expression` is evaluated
  - If it is true, then `statement` is executed.
  - If it is false, then `statement` is skipped

if statement example

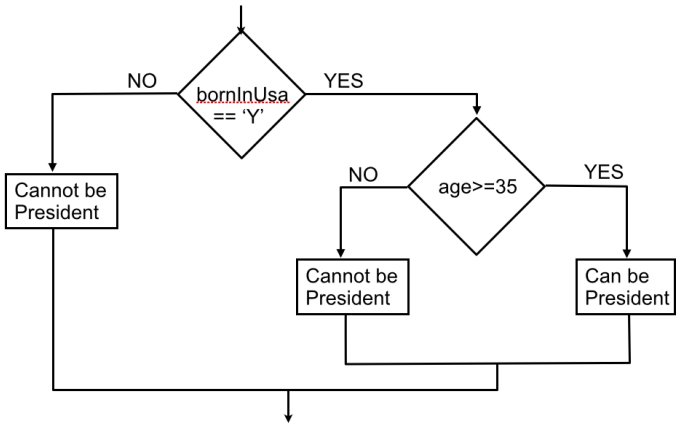
- Example: input validation

```
cout << "Enter a positive number: ";
cin >> x;
if (x <= 0)
{
   cout << "That number is not positive. "
        << "Please enter a positive number: ";
   cin >> x;
}

//do something with x here
```


4.5 Nested if statements

- if-else is a statement. It can occur as a branch of another if-else statement.



Nested if statements

- if-else is a statement. It can occur as a branch of another if-else statement.

```
char bornInUSA;
int age;
cout << "Were you born in the USA (Y/N)?: " ;
cin >> bornInUSA;
cout << "Please enter your age: ";
cin >> age;

if (bornInUSA == 'Y')
   if (age >= 35)
      cout << "You qualify to run for President\n";
   else
      cout << "You are too young to run for President\n";
else
   cout << "You must have been born in the US in order "
        << "to run for President" << endl;
```


Nested if statements

- if-else is a statement. It can occur as a branch of another if-else statement.

```
char bornInUSA;
int age;
cout << "Were you born in the USA (Y/N)?: " ;
cin >> bornInUSA;
cout << "Please enter your age: ";
cin >> age;

if (bornInUSA == 'Y')
   if (age >= 35)
      cout << "You qualify to run for President\n";
   else
      cout << "You are too young to run for President\n";
else
   cout << "You must have been born in the US in order "
        << "to run for President" << endl;
```


Common nested if pattern

- Determine letter grade from test score:

If we are in this else branch, what do we know about the value of testScore?

```
if (testScore < 60)
   grade = 'F';
else {
     if (testScore < 70)
        grade = 'D';
     else {
          if (testScore < 80)
             grade = 'C';
          else {
               if (testScore < 90)
                  grade = 'B';
               else
                  grade = 'A';
               }
          }
     }
```

- Note the braces are actually optional here!

4.6 The if-else if Statement

- Not really a different statement, just a different way of indenting the nested if statement from the previous slide:

```
if (testScore < 60)
   grade = 'F';
else if (testScore < 70)
   grade = 'D';
else if (testScore < 80)
   grade = 'C';
else if (testScore < 90)
   grade = 'B';
else
   grade = 'A';
```

- removed braces, put "if (…)" on previous line
- eliminated nested indentation.

4.8 Logical Operators

- Used to create relational expressions from other relational expressions:
  - **&&** AND (binary operator)

    a && b is true only when both a and b are true

  - **||** OR (binary operator)

    a || b is true whenever either a or b is true

  - **!** NOT (unary operator)

    !a is true when a is false

Logical Operators

- Examples

```
int x=6;
int y=10;

a. x == 5 && y <= 3
b. x > 0 && x < 10
c. x == 10 || y == 10
d. x == 10 || x == 11
e. !(x > 0)
f. !(x > 6 || y == 10)

bool flag;
flag = (x > 0 && x < 25);
g. !flag
h. flag || x < 100
```

false && false is false
true && true is true
false || true is true
___ || ___ is ____
!true is _____
!(false || true) is ____

Logical Operator Precedence

- ! is higher than most operators, so use parentheses:

```
int x;

...  !(x < 0 && x > -10) ... // <, >, &&, !
```

- && is higher than ||

```
int x, y;
bool flag;

... flag || x * 5 >= y + 10 && x == 5

   // which op is first? second? etc?
```

- && and || are lower than arithmetic+relational operators: parens not usually needed

4.9 Checking Numeric Ranges

- We want to know if x is in the range from 1 to 10 (inclusive)

```
a. if (1 <= x <= 10)          //as in math class
      cout << "YES" << endl;

   //WRONG: ((1<=x) <=10)  (assume x is -5)
   //    => ( false <= 10)
   //    => ( 0 < = 10 ) is true

b. if (1 <= x && x <= 10)
      cout << "YES" << endl;

   -check: x=0? (1<=0 && 0<=10) => false && true

   -check: x=5? (1<=5 && 5<=10) => true && true

   -check: x=100? (1<=100 && 100<=10) => ??
```

4.11 Validating User Input

- Input validation: inspecting input data to determine whether it is acceptable
- Invalid input is an error that should be treated as an exceptional case.
  - The program can ask the user to re-enter the data
  - The program can exit with an error message

```
cout << "Enter a positive number: ";
cin >> x;
if (x > 0) {
  //do something with x here

} else {
  cout << "You entered a negative number or 0." << endl;
  cout << "The program is ending." << endl;
}
```


4.12 Comparing Characters and Strings

- Characters are compared using their ASCII values

```
'A'<'B'
```

  - This is true.
    ASCII value of 'A' (65) is less than the ASCII value of 'B'(66)

```
'1'<'2'
```

  - This is true.
    ASCII value of '1' (49) is less than the ASCI value of '2' (50)

- Lowercase letters have higher ASCII codes than uppercase letters, so 'a' > 'Z'

Comparing string objects

- Like characters, strings are compared using their ASCII values

```
string name1 = "Mary";
string name2 = "Mark";

name1 > name2    // true
name1 <= name2   // false
name1 != name2   // true


name1 < "Mary Jane" // true
```

The characters in each string must match exactly in order to be equal

Otherwise, use first non-equal character as basis of the comparison ('y'>'k')

If a string is a prefix of the other, then it is less than the other

4.14 The `switch` statement

- Like a nested if/else, used to select one of multiple alternative code sections.
- tests **one** integer/char expression against **multiple** constant integer/char values:

```
switch (expression) {
   case const1: statements
   ...
   case const2: statements
   default: statements
}
```

`switch` statement behavior

```
switch (expression) {
   case const1: statements
   ...
   case const2: statements
   default: statements
}
```

- expression is evaluated to an int/char value
- execution starts at the case labeled with that int/char value
- execution starts at default if the int/char value matches none of the case labels

`switch` statement syntax

```
switch (expression) {
   case const1: statements
   ...
   case const2: statements
   default: statements
}
```

- expression must have int/char type
- const1, const2 must be constants!
  a literal or named constant
- statements is one or more statements
  (braces not needed and not recommended!)
- default: is optional

`switch` statement example

- Example:

```
int quarter;
...
switch (quarter) {
   case 1: cout << "First";
           break;
   case 2: cout << "Second";
           break;
   case 3: cout << "Third";
           break;
   case 4: cout << "Fourth";
           break;
   default: cout << "Invalid choice";
}
```


The `break` Statement

- The break statement causes an immediate exit from the switch statement.
- Without a break statement, execution continues on to the next set of statements (the next case).
- Sometimes this is useful: the textbook has some nice examples.

Multiple labels

- if ch is ‘a’, it falls through to output “Option A” (then it breaks)

```
char ch;
...
switch (ch) {
   case ‘a’:
   case ‘A’: cout << “Option A”;
             break;
   case ‘b’:
   case ‘B’: cout << “Option B”;
             break;
   case ‘c’:
   case ‘C’: cout << “Option C”;
             break;
   default: cout << “Invalid choice”;
}
```


4.10 Menus

- Menu-driven program: program controlled by user selecting from a list of actions
- Menu: list of choices on the screen
- Display list of numbered/lettered choices
- Prompt user to make a selection
- Test the selection in nested if/else or switch
  - Match found: execute corresponding code
  - Else: error message (invalid selection).

Sample menu code

```
int choice;
double charges;
int months = 12;
```

```
// Display the menu and get a choice.
cout << "Health Club Membership Menu\n\n";
cout << "1. Standard Adult Membership\n";
cout << "2. Child Membership\n";
cout << "3. Senior Citizen Membership\n";
cout << "Enter your choice: ";
cin >> choice;

// Respond to the user's menu selection.
switch (choice) {
   case 1:
     charges = months * 40.0;
     cout << "The total charges are $" << charges << endl;
     break;
   case 2:
     charges = months * 20.0;
     cout << "The total charges are $" << charges << endl;
     break;
   case 3:
     charges = months * 30.0;
     cout << "The total charges are $" << charges << endl;
     break;
   default:
     cout << “ERROR: The valid choices are 1 through 3.“ << endl;
}
```


4.15 More about blocks and scope

- The scope of a variable is the part of the program where the variable may be used.
- The scope of a variable is the innermost block in which it is defined, from the point of definition to the end of that block.
- Note: the body of the main function is just one big block.

Scope of variables in blocks

```
int main()
{
   double income;  //scope of income is red + blue
   cout << "What is your annual income? ";
   cin >> income;

   if (income >= 35000) {
      int years;   //scope of years is blue;
      cout << "How many years at current job? ";
      cin >> years;
      if (years > 5)
         cout << "You qualify.\n";
      else
         cout << "You do not qualify.\n";
   }
   else
      cout << "You do not qualify.\n";
   cout << "Thanks for applying.\n";
   return 0;
}
```

Cannot access years down here

Variables with the same name

- In an inner block, a variable is allowed to have the same name as a variable in the outer block.
- When in the inner block, the outer variable is not available (it is hidden).
- Not good style: difficult to trace code and find bugs

Variables with the same name

```
int main()
{
   int number;
   cout << "Enter a number greater than 0: ";
   cin >> number;
   if (number > 0) {
      int number;  // another variable named number
      cout << "Now enter another number ";
      cin >> number;
      cout << "The second number you entered was ";
      cout << number << endl;
   }
   cout << "Your first number was " << number << endl;
}
```

```
Enter a number greater than 0: 88
Now enter another number 2
The second number you entered was 2
Your first number was 88
```