Unit 3 Lesson 8
Selection Structures

Mr. Dave Clausen
La Cañada High School

Confusing = and = =

- Don’t confuse the meaning of = in Math with its meaning in C++.
- The symbol = means assigning a value to an identifier, and not that two objects or expressions are “equal” in C++.
- The symbol = = means equality in a comparison in C++.
- Side effects are caused if we confuse the two operators.

P209Ex1.cpp

P209Ex1.txt

Compound Boolean Expressions

- Logical Operators
  - And && (two ampersands) Conjunction
  - Or || (two pipe symbols) Disjunction
  - Not ! (one exclamation point) Negation
- Use parentheses for each simple expression and the logical operator between the two parenthetical expressions.
  - i.e. ((grade >= 80) && (grade < 90))

Order of Operations including Booleans

- 1. ( )
- 2. !
- 3. *, /, %
- 4. +, -
- 5. <, <=, >, >=, = =, !=
- 6. &&
- 7. ||

DeMorgan’s Laws

! (A && B) is the same as: !A || !B

Not (true AND true)	Not(true) OR Not(true)
Not (true)	false OR false
false	false

!(A || B) is the same as: !A && !B

Not(true OR true)	Not(true) AND Not(true)
Not (true)	false AND false
false	false

Boolean Expressions

- Selection Statements
  - a control statement that helps the computer make decisions
  - Certain code is executed when the condition is true, other code is executed or ignored when the condition is false.
- Control Structure
  - controls the flow of instructions that are executed.

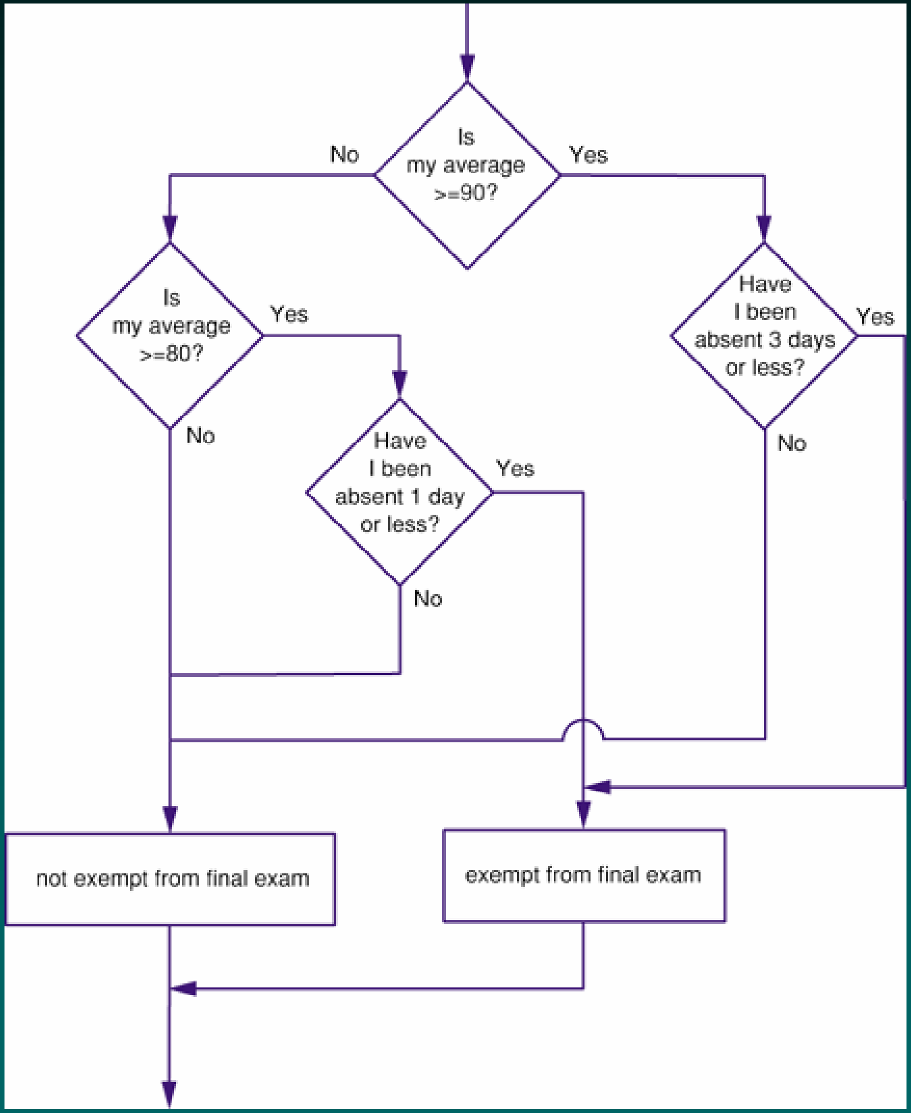
Introduction to Selection Structures

- Programs consist of statements that solve a problem or perform a task.
- Up to this point, you have been creating programs with ***sequence structures***.
- ***Sequence structures*** execute statements one after another without changing the flow of the program.
- Other structures, such as the ones that make decisions, do change the flow of the program.
- The structures that make decisions in C++ programs are called ***selection structures.***
- When a decision is made in a program, a selection controls the flow of the program based on the decisions in your program.

Use Selection Structures to Make Decisions

- ***Selection structures*** are how C++ programs make decisions.
- The ***if*** structure is a one-way selection structure. When a control expression in an *if* statement is evaluated to be true, the statements associated with the structure are executed.
- The ***if/else*** structure is a two-way selection structure. If the control expression in the *if* statement evaluates to true, one block of statements is executed; otherwise (*else*), another block is executed.
- The ***switch*** structure is a multi-way selection structure that executes one of many sets of statements, depending on the value of the control expression. The control expression must evaluate to an integer or character value.

Flowcharts can illustrate program flow when using selection structures


Is my average >=90?
No
Yes
Is my average >=80?
Yes
No
Have I been absent 3 days or less?
Yes
No
Have I been absent 1 day or less?
Yes
No
not exempt from final exam
exempt from final exam

Using if

- Many programming languages include an *if structure.*
- Although the syntax varies among the programming languages, the *if* keyword is usually part of every language.
- If you have used *if* in the other programming languages, you should have little difficulty using *if* in C++.
- The *if* structure is one of the easiest and most useful parts of C++.

If Statements

- Format for if statements:

  if (<Boolean expression>)

      <statement>

  – Parentheses are **required** around the Boolean Expression.
  – The Boolean expression that makes the decision is called the ***control expression***.

```
sum = 0.0;
cin>>number;
if (number > 0.0)
  sum = sum + number;
cout<<“the sum is: “<<sum<<endl;
```

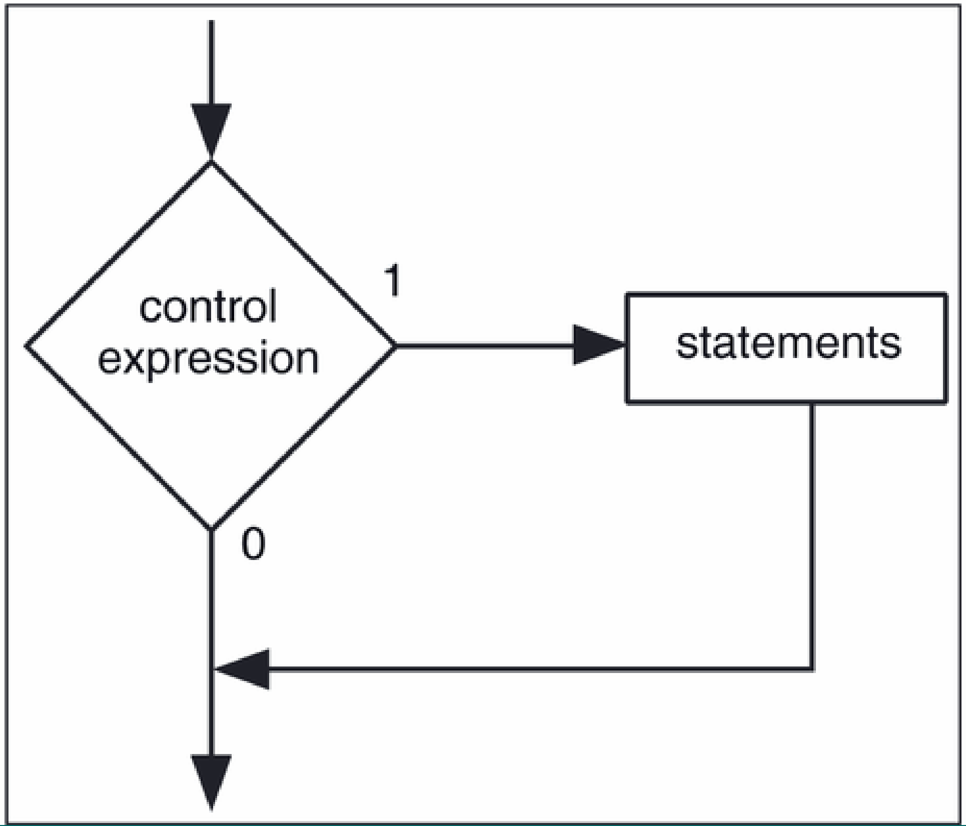
Code List 8-1

```
if ( i = = 3 )
{
   cout << "The value of i is 3"<<endl;
}
```

- The curly brackets are not required if you only want one statement to be executed when the control expression is true.

Figure 8-1

FIGURE 8-1
The if structure is sometimes called a one-way selection structure.



Style for If Statements

- For our class **don’t use the books programming style** of putting the brackets at the beginning and ending of the line, for example:

```
if (i = = 3)
   {cout  <<  “The value of i is 3”<<endl; }
```

- **Instead use the following format:**

```
if (i = = 3)
{
   cout  <<  “The value of i is 3”<<endl;
}
```

Compound Statements

- Use the symbols { and } to group several statements as a single unit.
- Simple statements within the compound statements end with semicolons.
- Compound Statements are sometimes called a statement block.
- Use compound statements in an if statement if you want several actions executed when the Boolean expression is true.

Minmax.cpp

Minmax.txt

Code List 8-2

```
if  ( yes_no  = =  'Y' )
{
    cout  <<  "Enter the title:  ";
    getline(cin, title);
}
```

- If you wish to have more than one line of code executed when the control expression is true, use { and } to treat the lines of code as a compound statement.

Code List 8-3

```
//  city.cpp                    city.txt
#include  <iostream.h>
#include  “oostring.cpp”

int main ()
{
      oostring  city_name;
      unsigned long population;
      cout  <<  “What is the name of you city or town?  “;
      getline(cin, city_name);
      cout  <<  “what is the population of the city or town?  ‘;
      cin  >>  population;
      cin.ignore(80, ‘\n’);
      if  (population  >=  185086)
      {
                cout  <<  “According to estimates population figures,  “;
                cout <<  city_name  << “ is one of the 100 largest u.s. cities.\n”;
      }
return  0;
}
```

Errors with if Statements

- Do NOT place a **;** (semicolon) directly after the Boolean expression in an *if* statement:
- Don’t do this for example:

```
if ( i = = 3); //Don’t do this!
   cout<<“The value of i is 3” << endl;
```

- This will cause syntax and / or logic errors in your program.

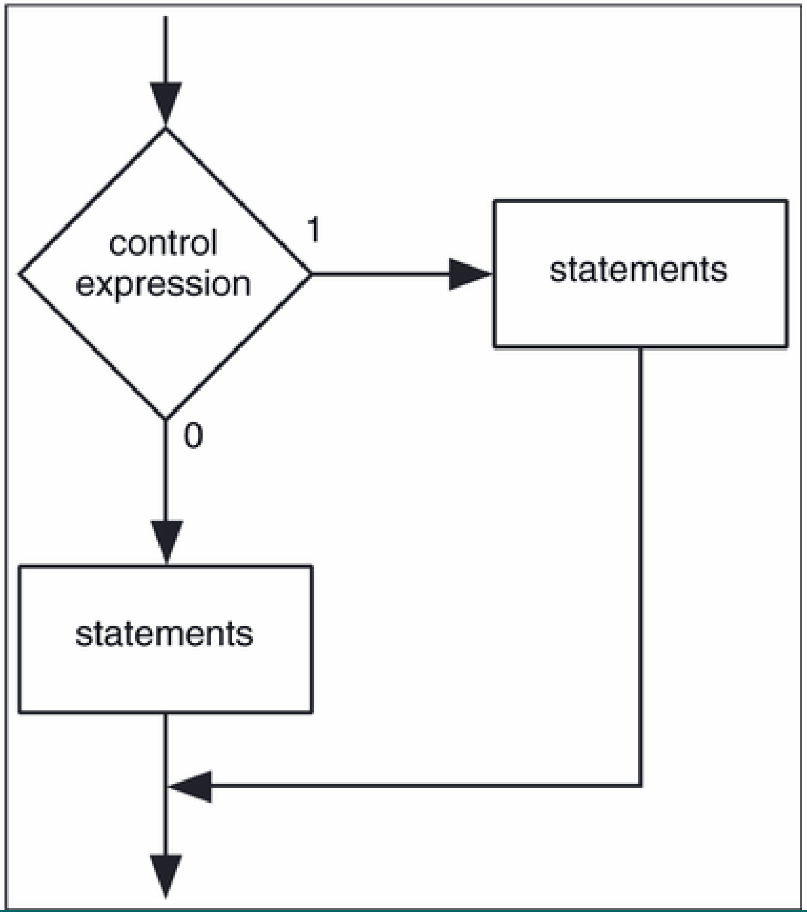
Using if/else

- The if/else structure is sometimes called a two-way selection structure.
- Using if/else, one block of code is executed if the control expression is true.
- The else portion of the structure is executed if the control expression is false.

Two Way Selection Structure

- Figure 8-2 shows a flowchart for a two-way selection structure.

FIGURE 8-2
The if/else structure is a two-way selection structure.



Code List 8-4

- Consider the code fragment in Code List 8-4

```
if ( i < 0)
{
   cout << “The number is negative.\n”;
}
else
{
   cout << “The number is zero or positive.\n”;
}
```

Code List 8-5

```
if (population >= 185086)
{
    cout << “According to estimated population figures, ”;
    cout << city_name << endl;
    cout << “ is one of the 100 largest U.S. cities.\n”;
}
else
{
    cout << “According to estimated population figures, ”;
    cout << city_name << endl;
    cout << “ is not one of the 100 largest U.S. cities.\n”;
}
```

if …else Statements

- Format for if…else statements

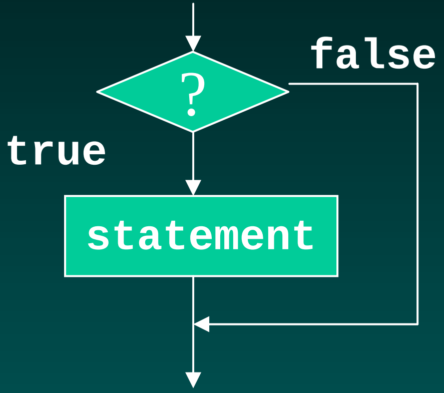
```
if (<Boolean expression>)
   <true statement>      //indent 3 spaces
    //end of if option
else
    <false statement>    //indent 3 spaces
    //end of else option
```

if …else Example

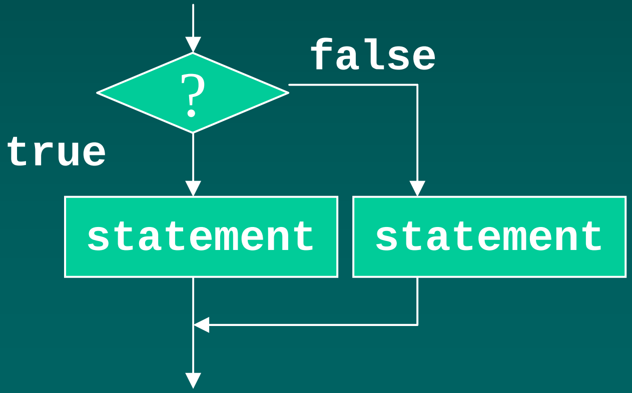
```
cout<<“Please enter a number and press <Enter>. “;
cin>>number;
if (number < 0)
{
   neg_count = neg_count + 1;
   cout<<setw(15) << number<<endl;
}
//end if option
else
{
   non_neg_count = non_neg_count + 1;
   cout << setw(30) << number << endl;
}
//end of else option
```

Behavior of Selection Statements

```
if (<Boolean expression>)
{
   <statement 1>
   .
   <statement n>
}
```



```
if (<Boolean expression>)
   <statement>
else
{
   <statement 1>
   .
   <statement n>
}
```



Errors with if else Statements

- Do NOT place a **;** (semicolon) directly after the command *else* in an *if else* statement:
- Don’t do this for example:

```
if ( i < 0)
   cout << “The number is negative.\n”;
else;  //Don’t do this!
   cout << “The number is zero or positive.\n”;
```

- This will cause syntax errors in your program.

Nested if statements

- Nested if statement
  - an if statement used within another if statement where the “true” statement or action is.

```
if (score >=50)
    if (score>=69.9)
        cout<<blah, blah, blah      //true for score>=69.9 and score>=50
    else
        cout<<blah, blah, blah      //false score>=69.9 true score >=50
else
   cout<<blah, blah, blah           //false for score >=50
```

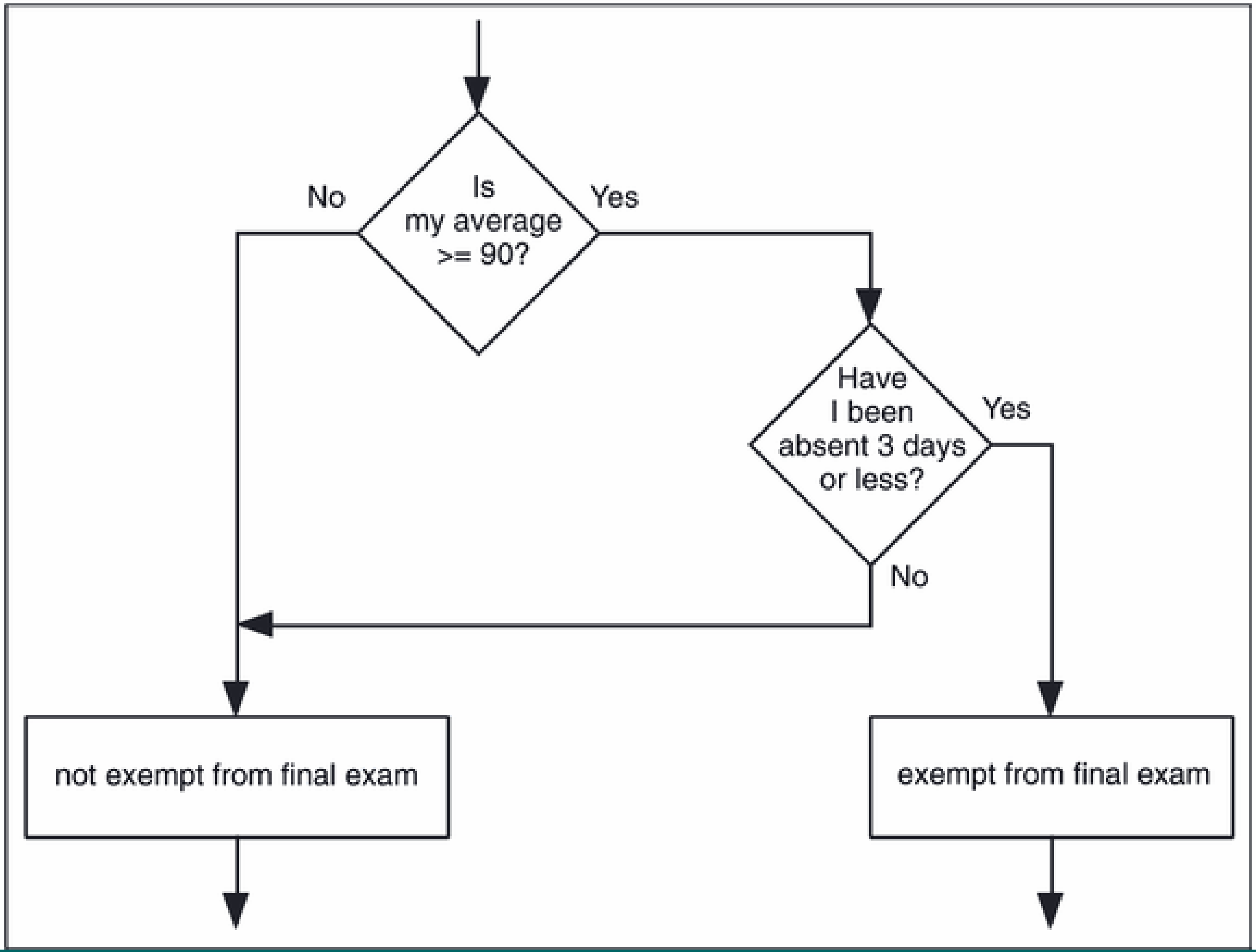
Nested if Structures

- You can place *if* structures within other *if* structures.
- When an *if* or *if/else* structure is placed within another *if* or *if/else* structure, the structures are said to be ***nested***.
- The flowchart in Figure 8-3 decides whether a student is exempt from a final exam based on grade average and days absent.
- Don’t get your hopes up, this doesn’t work at our school!

Figure 8-3

FIGURE 8-3

This flowchart can be programmed using nested if structures.


Is my average >= 90?
No
Yes
Have I been absent 3 days or less?
Yes
No
not exempt from final exam
exempt from final exam

Code List 8-6

- The code in Code List 8-6 is written to initially assume that the student is not exempt from the final exam.

```
exempt_from_final = false;

if (my_average >= 90)
{                                   // If your average is 90 or better
    if (my_days_absent <= 3)        // and you have missed three days or less
    {
          exempt_from_final = true; // you are exempt (not at LCHS).
    }
}
```

Preferred Code List 8-6

- Nested if statements can be confusing, rewriting Code
  List 8-7 using Compound Boolean Expressions is much more clear.
- I prefer that you avoid Nested if statements and use compound Boolean Expressions whenever possible.
- If not, use ***extended*** if statements.

```
if (my_average >= 90) && (my_days_absent <= 3)
{
    exempt_from_final = true;
}
```

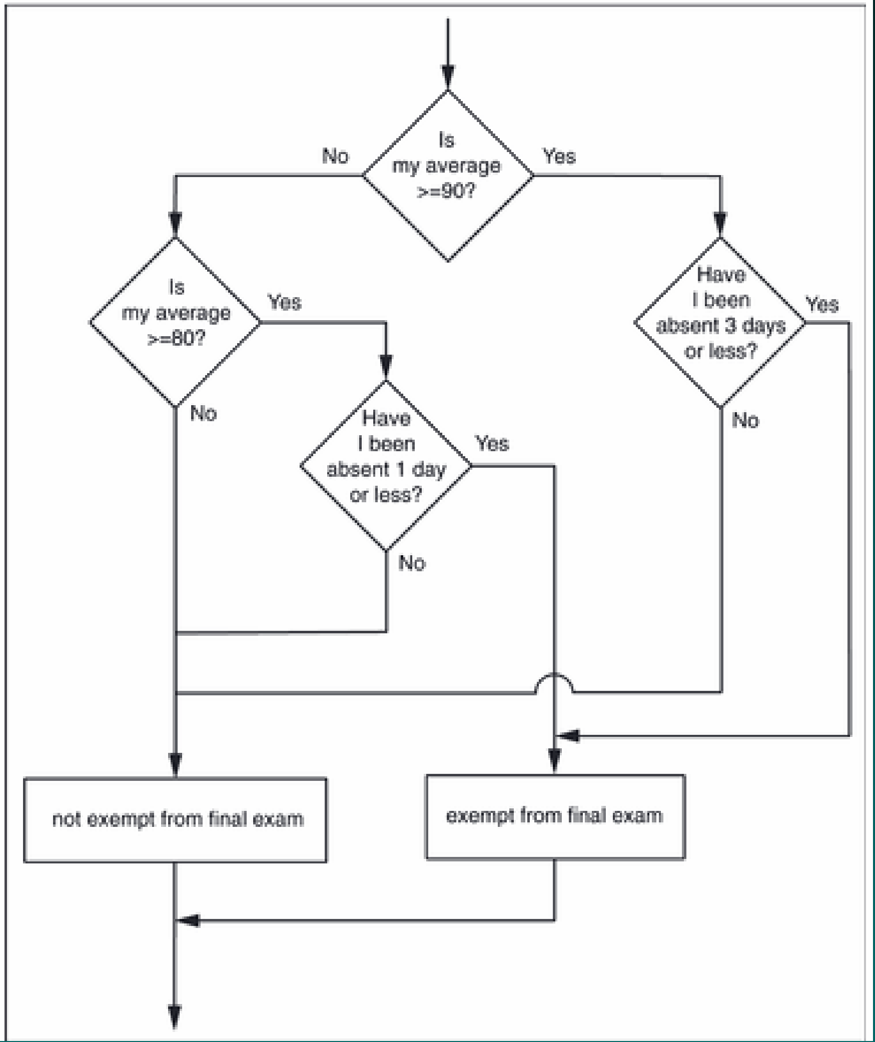
Figure 8-4

- Figure 8-4 shows the flowchart from Figure 8-3 expanded to include another way to be exempted from the final exam.
- The new algorithm allows a student to be exempt from the exam if the student’s grade is greater than or equal to 80 AND they have only been absent from school one day or less.
- This still doesn’t work at our school!

Flowchart 8-4

FIGURE 8-4

This algorithm provides two paths to exemption from the final exam.


Is my average >=90?
No
Yes
Is my average >=80?
Yes
No
Have I been absent 3 days or less?
Yes
No
Have I been absent 1 day or less?
Yes
No
not exempt from final exam
exempt from final exam

Code List 8-7

```
exempt_from_final = false;
if (my_average >= 90)
{                                    // If you average is 90 or better
   if (my_days_absent <= 3)          // and you have missed three days
      exempt_from_final = true;      // or less, you are exempt.
}
else
{
   if (my_average >= 80)
   {                                 // If your average is 80 or better
     if (my_days_absent <= 1)        // and you have missed one day or
        exempt_from_final = true;    // less, you are exempt.
   }
}
```

Preferred Code List 8-7

- Nested if statements can be confusing, rewriting Code List 8-7 using Compound Boolean Expressions is much more clear.
- I prefer that you avoid Nested if statements and use compound Boolean Expressions whenever possible.
- If not, use ***extended*** if statements.

```
exempt_from_final = false;
if (my_average >= 90)  && (my_days_absent <= 3)
{
     exempt_from_final = true;
}
else  if (my_average >= 80)&& (my_days_absent <= 1)
{
            exempt_from_final = true;
}
```

Code List 8-8

```
// deposit.cpp                              deposit.txt
#include <iostream.h>
int main ( )
{
        float amount_to_deposit;

        cout << “How much do you want to deposit to open the account?”;
        cin   >> amount_to_deposit;

        if (amount_to_deposit < 1000.00)
          {
             if (amount_to_deposit < 100.00)
                     cout << “You should consider the EconoCheck account.\n”;
             else
                     cout << “You should consider the FreeCheck account.\n”;
          }
        else
           cout << “You should consider an interest-bearing account.\n”;
        return 0;
}
```

Extended if statements

– Extended if statements are Nested if statements where another if statement is used with the else clause of the original if statement.

```
if  (condition 1)
     action1
else if (condition2)
          action2
     else
          action3
```

Preferred Code List 8-8

```
// deposit2.cpp                deposit2.txt
#include <iostream.h>
int main ( )
{
    float amount_to_deposit;
    cout << “How much do you want to deposit to open the account?”;
    cin  >> amount_to_deposit;
    if (amount_to_deposit < 100.00)
      cout << “You should consider the EconoCheck account.\n”;
    else if (amount_to_deposit < 1000.00)
           cout << “You should consider the FreeCheck account.\n”;
         else
           cout << “You should consider an interest-bearing account.\n”;
    return 0;
}
```

Revised Code List 8-4

- Let’s refine Code List 8-4 to check if the number is zero using ***extended if*** statements.

```
if ( i < 0)
   cout << “The number is negative.\n”;
else if (i > 0)
          cout << “The number is positive.\n”;
     else
          cout<< “The number is zero.\n”;
```

Avoid Sequential Selection

- This is not a good programming practice.
  - Less efficient
  - only one of the conditions can be true
    - this is called mutually exclusive conditions

```
if (condition1)                //avoid this structure
     action1                   //use extended selection statements
if (condition2)
     action2
if (condition3)
     action3
```

Sales.cpp

Sales.txt

The switch Structure

- So far, we have studied one-way (if) and two-way (if/else) selection structures.
- C++ has another method of handling multiple options known as the switch structure.
- The switch structure has many uses but is most often used with menus.
  - A menu is a set of options presented to the user of a program.
  - Code List 8-9 displays a menu of choices and asks the user to enter a number that corresponds to one of the choices.
  - Then a case statement is used to handle each of the options.
- Nested if/else structures could be used in place of the switch structure.

Code List 8-9

```
cout << “How do you want the order shipped?\n”;
cout << “1 - Ground\n”;
cout << “2 - 2-day air\n”;
cout << “3 – Overnight air\n”;
cout << “Enter the number of the shipping method you want:  ”;
cin >> shipping_method;
switch (shipping_method)
{
      case 1:              shipping_cost = 5.00;
                            break;

      case 2:               shipping_cost = 7.50;
                            break;

      case 3:               shipping_cost = 10.00;
                            break;

       default:            shipping_cost =  0.00;
                           break;
}
```

The switch Structure 2

- The switch structure is easier to use and a programmer is less prone to making errors that are related to braces and indentations.
- Remember, however, that an integer or character data type is required in the control expression of a switch structure.
- When using character types in a switch structure, enclose the characters in single quotation marks as in any other character literal.
- Use a **: (colon)** after each ***case*** statement, not a **; (semi-colon)**.
- The code segment in Code List 8-10 is an example of using character literals in a switch structure.

Code List 8-10

```
switch (character_entered)
{

      case ‘A’:      cout << “The character entered was A, as in albatross.\n”;
                     break;

      case ‘B’:      cout << “the character entered was B, as in butterfly.\n”;
                     break;

      default:       cout << “Invalid choice.  Please enter an A or B”  <<endl;
                     break;
}
```

Switch Statements

- Allows for multiple selection that is easier to follow than nested or extended if statements.

```
switch (age)    //age is of type int
{
   case 18:     <statement1>
                break;
   case 19:     <statement2>
                break;
   case 20:     <statement3>
                break;
   default:     <default statement>
                break;
}
```

Switch Statement Example 2

```
switch (grade)                    //grade is of type char
{
    case ‘A’ :
    case ‘B’ :     cout<<“Good work!”<<endl;
                   break;
    case ‘C’ :     cout<<“Average work”<<endl;
                   break;
    case ‘D’ :
    case ‘F’ :     cout<<“Poor work”<<endl;
                   break;
    default  :     cout<<grade<<“ is not a valid letter grade.”;
                   break;
}
```

Switch: Flow of Execution

- The selector (argument) for switch must be of an ordinal type (not double)
  - switch (age)
  - The variable “age” is called the selector in our example.
  - If the first instance of the variable is found among the labels, the statement(s) following this value is executed until reaching the next break statement.
  - Program control is then transferred to the next statement following the entire switch statement.
  - If no value is found, the default statement is executed.

Errors with Switch Statements

- Do NOT place a **;** (semicolon) directly after the command *switch* in a switch structure:
- Don’t do this for example:

```
switch (character);  //Don’t do this!
{
     case ‘A’:      cout << “The character entered was A.” <<endl;
                    break;
     case ‘B’:      cout << “the character entered was B.” <<endl;
                    break;
     default:       cout << “Please enter an A or B”  <<endl;
                    break;
}
```

Program Testing

- Use test data that tests every branch or selection in the program.
- Test the true statement(s)
- Test the else statement(s)
- Test the border, edge, extreme cases.
- Test carefully nested and extended selection statements and their paths.
- Test every case in a switch statement.