C++ Programming

Instructor: Rita Kuo
Office: CS 520E
Phone: Ext. 4405
E-mail: rita.kuo@uvu.edu

Mapping zyBooks Chapters

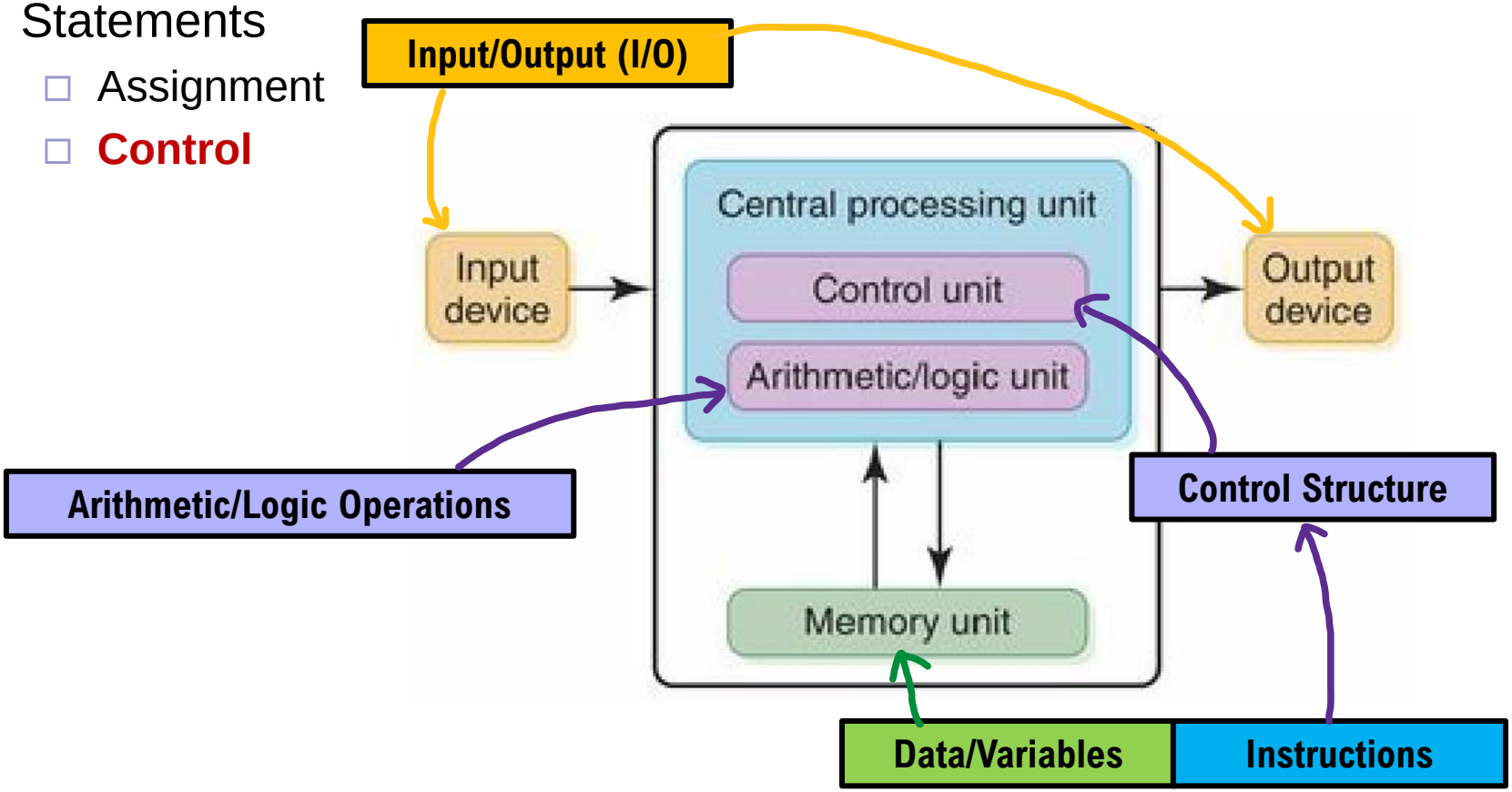
Topics on the slides	Chapters in zyBooks
Function Basic	6.1
Logical Expression	3.5, 3.6, 3.10
Selection Structure – if Statement	3.2, 3.7, 3.8, 3.13, 3.18
Selection Structure – switch Statement	3.12
Iteration Structure – while and do-while Statement	4.2, 4.3, 4.8
Iteration Structure – for Statement	4.4, 4.5
Increment/Decrement Operators	4.5
String Operations	3.15, 3.14, 3.16, 3.17
For Each Statement	4.6
Floating-point Comparison	3.19
Jump Statements - break and continue	4.10
Bitwise Operator	3.11
Enumerations	4.12

Not in the slides but should be covered in the pre-req courses: 3.20, 4.7, 4.9, 4.11

Structure Programming

Review - Elements in Programming Language

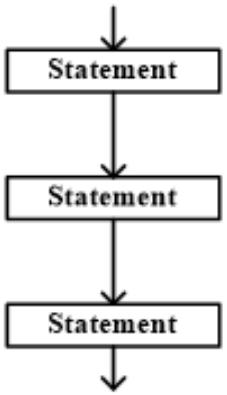
- Input/Output (I/O)
- Variables
- Expression
- Statements
  - Assignment
  - **Control**



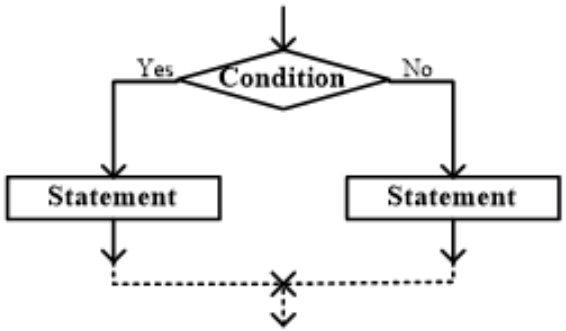
Structured Programming

- A programming paradigm
  - Improve the clarity quality and development time of computer program
  - Make extensive use of the **structured control flow**:

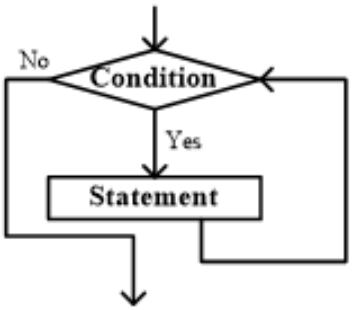
sequence



selection



iteration



  - Subroutines: functions, methods
  - Blocks: treat groups of statements as one statement

```
for (i = 0; i < 10 ; i ++)
{
    count = count + 1;
    num = num + i;
}
```
braces

```
while current <= n:
    sum = sum + current
    current = current + 1
```
indentation

Function Basics

Review - The First Program

- A simple C++ program form

```
directives

int main()
{
    statements
}
```

- Example

```
#include <iostream>
using namespace std;

int main() {
  int wage;

  wage = 20;

  cout << "Salary is ";
  cout << wage * 40 * 52;
  cout << endl;

  return 0;
}
```

A program starts in `main()` function

- Execute the statements within braces {}
- One statement at a time
- Each statement ends with a semicolon, as English sentences end with a period

Functions

- A C++ program is a collection of functions
- The form of a basic function:

```
return-val name-of-func (args) {
        body of function
}
```

Functions

- What is a function?
  - A small program, with its own declarations and statements
- Benefits
  - Divide a program into small pieces that are easier for people to understand and modify
  - Avoid duplicating code that's used more than once
- The form of a function in C++

```
return-val name-of-function (list of formal parameters)
{
    body of function
}
```

- Example

```
int main()
{
  cout << "Hello World!"<< endl;
  return 0;
}
```

Functions

- Example: a function returns computed square

```
#include <iostream>
using namespace std;


/*   Implement the ComputeSquare function here */


int main() {
   int numSquared;

   numSquared = ComputeSquare(7);
   cout << "7 squared is " << numSquared << endl;

   return 0;
}
```

Functions

- Example: a function returns computed square

```
#include <iostream>
using namespace std;

int ComputeSquare(int numToSquare) {
   return numToSquare * numToSquare;
}

int main() {
   int numSquared;

   numSquared = ComputeSquare(7);
   cout << "7 squared is " << numSquared << endl;

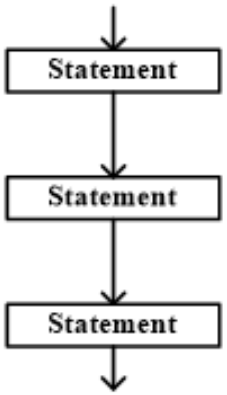
   return 0;
}
```

Control Flow

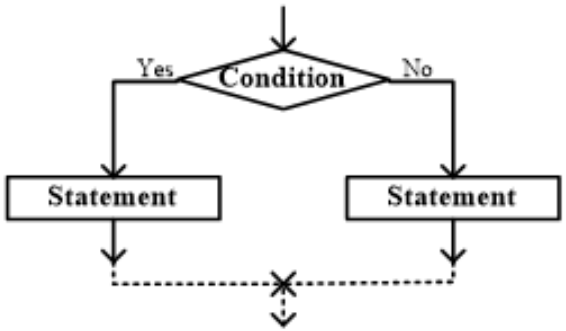
Review - Structured Programming

- A programming paradigm
  - Improve the clarity quality and development time of computer program
  - Make extensive use of the **structured control flow**:

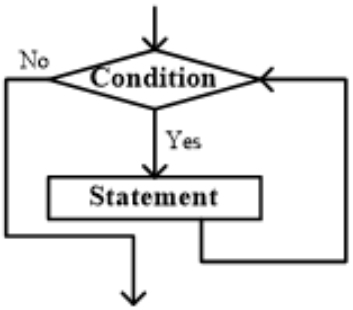
sequence



selection



iteration



  - Subroutines: functions, methods
  - Blocks: treat groups of statements as one statement

```
for (i = 0; i < 10 ; i ++)
{
    count = count + 1;
    num = num + i;
}
```
braces

```
while current <= n:
    sum = sum + current
    current = current + 1
```
indentation

Structured Control Flow

- Selection Statements in C++
  - `if … else` selection statement:
    Performs an action if a condition is true and perform a different action if the condition is false
  - `switch` selection statement:
    Performs one of many different actions, depending on the value of an expression.
- Iteration Statements in C++
  - `while`
  - `do … while`
  - `for`
  - Foreach
- Jump Statements
  - `break`, `continue` statements

Logical Expression

Logical Expression

- The expression evaluates to true of false
- Relational and Equality Operators

Standard algebraic equality or relational operator	C++ equality or relational operator	Sample C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Logical Expression

- Logical Operators

a	b	a AND b
false	false	false
false	true	false
true	false	false
true	true	true

a	b	a OR b
false	false	false
false	true	true
true	false	true
true	true	true

a	NOT a
false	true
true	false

- Examples

Let x = 7, y = 9

(x > 0) AND (y < 10) true
true true

(x > 0) AND (y < 5) false
true false

(x < 0) OR (y > 10) false
false false

(x < 0) OR (y > 5) true
false true

NOT (x < 0) true
false

NOT (x > 0) false
true

Logical Expression

- Operator Precedence and Associativity

Operators	Associativity	Type
`::`	left to right	scope resolution
`()`		grouping parentheses
`++ -- static_cast< type >()`	left to right	unary (postfix)
`++ -- + - !`	right to left	unary (prefix)
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`<< >>`	left to right	insertion/extraction
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`&&`	left to right	logical AND
`||`	left to right	logical OR
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment
`,`	left to right	comma

Selection Structure – `if` Statement

Review - Structured Control Flow

- Selection Statements in C++
  - `if … else` selection statement:
    Performs an action if a condition is true and perform a different action if the condition is false
  - `switch` selection statement:
    Performs one of many different actions, depending on the value of an expression.
- Iteration Statements in C++
  - `while`
  - `do … while`
  - `for`
  - Foreach
- Jump Statements
  - `break`, `continue` statements

`if` Statements

- The basic form of the **`if`** statement

```
if (expression) {
        /* if body */
}
```

  - The expression evaluates to **`true`** or **`false`**
  - In C++, **`true`** is any non-zero value, while **`false`** is zero
  - C++ has the built-in data type **`bool`** for representing Boolean quantities.
    Ex. `bool isOverweight = true; bool hasHighBP = false;`
  - In C++ 11 or 14 standard:
    If the source type is `bool`, the value `false` is converted to zero and the value `true` is converted to one.
  - C++ expressions that evaluate to true or false, typically involve
    - relational operators: **`<`**, **`<=`**, **`>`**, **`>=`**,
    - equality operators: **`==`**, **`!=`**,
    - boolean operators: **`&&`**, **`||`**, **`!`**, and/or
    - function return values

https://stackoverflow.com/questions/5369770/bool-to-int-conversion
https://stackoverflow.com/questions/5764956/which-is-faster-if-bool-or-ifint

`if-else` Statements

- The basic form of the **`if`** statement

```
if (expression) {
        /* if body */
}
```

- Have optional else section

```
if (expression) {
        /* if body */
} else {
        /* else body */
}
```

  - else body is executed if the expression in the parentheses has the value 0

`if-else` Statements

- Example: Find the maximum of two numbers

```
#include <iostream>
using namespace std;

int max(int a, int b)
{
        if (a > b) {
                return a;
        } else {
                return b;
        }
}

int main(void)
{
        int a = 16;
        int b = 9;

        cout << "max of " << a << " and " << b << " is " << max(a, b) << endl;

        return 0;
}
```

`if-else` Statements

- Ternary Operator
  - Produce one of two values depending on the value of a condition
  - Consists of two symbols (**?** and **:**), which must be used together:
    expr1 **?** *expr2* **:** *expr3*
  - Should be read "if *expr1* then *expr2* else *expr3*"
  - The following statements

```
if (a > b) {
        return a;
} else {
        return b;
}
```

  - can be transferred to

```
return a > b ? a : b;
```

  - Another example:

```
cout << (i == 100 ? "you win" : "you lose") << endl;
```

`if-else` Statements

- The basic form of the `if` statement with optional `else` section

```
if (expression) {
        /* if body */
} else {
        /* else body */
}
```

  - There are no restrictions on what kind of statements can appear inside an `if` statement.

```
if (i > j)
    if (i > k)
        max = i;
    else
        max = k;
else
    if (j > k)
        max = j;
    else
        max = k;
```

`if-else if-else` Statements

- Cascaded if Statement (`if-else if-else` Statement)
  - Test a series of conditions, stopping as soon as one of them is true

```
if (expression) {
          /* if body */
} else if (expression) {
/* else if body */
} else if (expression) {
/* as many else if you need */
} else {
          /* else body */
}
```

  - Example

```
if (n < 0)
  cout << "n is less than 0" << endl;
else
  if (n == 0)
    cout << "n is equal to 0" << endl;
  else
    cout << "n is greater than 0" << endl;
```

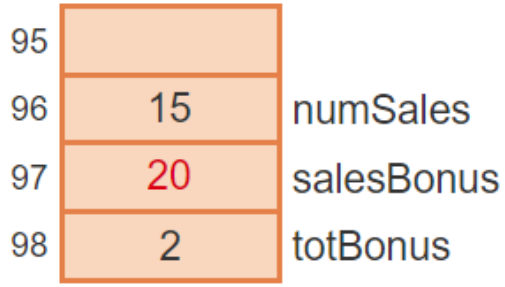
```
if (n < 0)
  cout << "n is less than 0" << endl;
else if (n == 0)
  cout << "n is equal to 0" << endl;
else
  cout << "n is greater than 0" << endl;
```

Common Errors

- Common Error

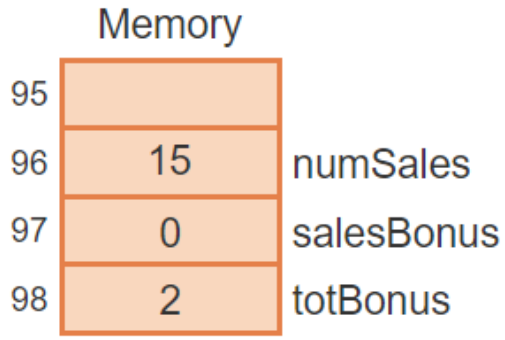
```
if (numSales < 20)      15 < 20
   salesBonus = 0;
else
   totBonus = totBonus + 1;
   salesBonus = 20;
```

Indentation is irrelevant.
salesBonus = 20; is not part of else,
so always executes.




```
if (numSales < 20) {    15 < 20
   salesBonus = 0;
}
else {
   totBonus = totBonus + 1;
   salesBonus = 20;
}
```

Always using braces avoids
the above common error.



Common Errors

- To which `if` statement does the `else` clause belong?

```
if (y != 0)
    if (x != 0)
        result = x / y;
else
    printf("Error: y is equal to 0\n");
```

Common Errors

- To which `if` statement does the `else` clause belong?

```
if (y != 0)
    if (x != 0)
        result = x / y;
else
    printf("Error: y is equal to 0\n");
```

  - C follow the rule that an `else` clause belongs to the nearest `if` statement that hasn't already been paired with an `else`.
  - To make the `else` clause part of the outer `if` statement, we can enclose the inner `if` statement in braces

```
if (y != 0) {
    if (x != 0)
    result = x / y;
} else
    printf("Error: y is equal to 0\n");
```

Selection Structure – `switch` Statement

Review - Structured Control Flow

- Selection Statements in C++
  - `if … else` selection statement:
    Performs an action if a condition is true and perform a different action if the condition is false
  - `switch` selection statement:
    Performs one of many different actions, depending on the value of an expression.
- Iteration Statements in C++
  - `while`
  - `do … while`
  - `for`
  - Foreach
- Jump Statements
  - `break`, `continue` statements

`switch` Statements

- `switch` selection statement:
  - Simplifies the logic of `if-else if-else` for integer types when they involve tests for equality
  - Key: work with **integer types only**! (`char`, `short`, `int`, `long`)
  - Example

```
if (grade == 4)
    cout << "Excellent\n";
else if (grade == 3)
    cout << "Good\n";
else if (grade == 3)
    cout << "Average\n";
else if (grade == 3)
    cout << "Poor\n";
else if (grade == 3)
    cout << "Failing\n";
else
    cout << "Illegal grade\n";
```

```
switch (grade) {
case 4:  cout << "Excellent\n";
         break;
case 3:  cout << "Good\n";
         break;
case 2:  cout << "Average\n";
         break;
case 1:  cout << "Poor\n";
         break;
case 0:  cout << "Failing\n";
         break;
default: cout << "Illegal grade\n";
         break;
}
```

`switch` Statements

- `switch` selection statement:
  - What happen if there is no `break` statement in each case?

```
switch (grade) {
case 4:  cout << "Excellent\n";
case 3:  cout << "Good\n";
case 2:  cout << "Average\n";
case 1:  cout << "Poor\n";
case 0:  cout << "Failing\n";
default: cout << "Illegal grade\n";
}
```

  - Without `break` (or some other jump statement), control will flow from one case into the next.
  - E.g., if the value of grade is 3, the message printed is

```
GoodAveragePoorFailingIllegal grade
```

`switch` Statements

- Please revise the previous program to make the grade as letter grades (A, B, C, D, F). User's input should be considered.

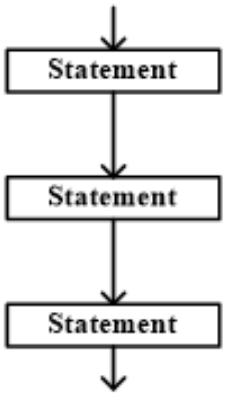
```
switch(grade) {
case 'A':
case 'a':  printf("Excellent");
           break;
case 'B':
case 'b':  printf("Good");
           break;
case 'C':
case 'c':  printf("Average");
           break;
case 'D':
case 'd':  printf("Poor");
           break;
case 'F':
case 'f':  printf("Failing");
           break;
default:   printf("Illegal grade");
           break;
}
```

Iteration Structure – `while` and `do-while` Statement

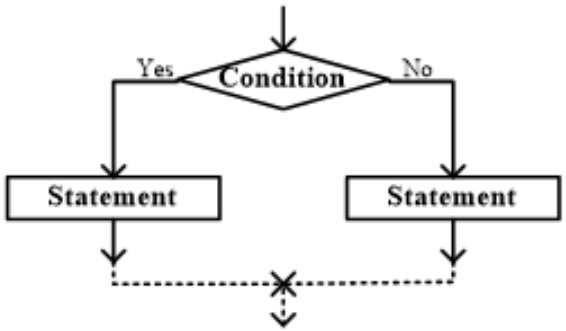
Review - Structured Programming

- A programming paradigm
  - Improve the clarity quality and development time of computer program
  - Make extensive use of the **structured control flow**:

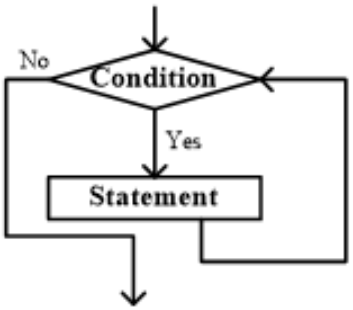
sequence



selection



iteration



  - Subroutines: functions, methods
  - Blocks: treat groups of statements as one statement

```
for (i = 0; i < 10 ; i ++)
{
    count = count + 1;
    num = num + i;
}
```
braces

```
while current <= n:
    sum = sum + current
    current = current + 1
```
indentation

Review - Structured Control Flow

- Selection Statements in C++
  - `if … else` selection statement:
    Performs an action if a condition is true and perform a different action if the condition is false
  - `switch` selection statement:
    Performs one of many different actions, depending on the value of an expression.
- Iteration Statements in C++
  - `while`
  - `do … while`
  - `for`
  - Foreach
- Jump Statements
  - `break`, `continue` statements

while statements

- Syntax
  - **while (condition) {**
    /* while body */
    }
  - Example:

```
int n = 10;
int i = 1;
while (i < n)    /* controlling expression */
        i = i * 2;       /* loop body */
```

  - Example

```
int i = 10;
while (i > 0) {
        printf("T minus %d and counting\n", i);
        i--;
}
```

`while` statements

- Example: GCD

```
while (numA != numB) { // Euclid's algorithm
    if (numB > numA) {
       numB = numB - numA;
    }
    else {
       numA = numA - numB;
    }
 }

 cout << "GCD is: " << numA << endl;
```

`do-while` statements

- Syntax
  - ```
    do {
      /* do body */
    } while (condition)
    ```
  - Do-loops always run the body of the loop once and then the test (while) is performed
  - If the test is false, the do-loop stops executing; if the test is true, the body of the do-loop executes again.
  - Use a `do … while` loop when you always want to do the loop at least once.
    ```

`do-while` statements

- Example

```
#include <iostream>
using namespace std;

int main() {
  char fill;

  fill = '*';

  do {
     cout << fill << fill << fill << endl;
     cout << fill << fill << fill << endl;
     cout << fill << fill << fill << endl;

     cout << "Enter char (q to quit): ";
     cin >> fill;
     cout << endl;
  } while (fill != 'q');

  return 0;
}
```

```
***
***
***
Enter char (q to quit):  x

xxx
xxx
xxx

Enter char (q to quit): q
```

Iteration Structure – `for` Statement

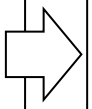
Review - Structured Control Flow

- Selection Statements in C++
  - `if … else` selection statement:
    Performs an action if a condition is true and perform a different action if the condition is false
  - `switch` selection statement:
    Performs one of many different actions, depending on the value of an expression.
- Iteration Statements in C++
  - `while`
  - `do … while`
  - `for`
  - Foreach
- Jump Statements
  - `break`, `continue` statements

`for` statement

- Syntax
  - `for (initialization; condition; update ) {`
    `/* for loop body */`
    `}`
- Example
  - Print “hello world!” for 10 times

```
cout << "hello world!" << endl;
cout << "hello world!" << endl;
cout << "hello world!" << endl;
cout << "hello world!" << endl;
cout << "hello world!" << endl;
cout << "hello world!" << endl;
cout << "hello world!" << endl;
cout << "hello world!" << endl;
cout << "hello world!" << endl;
cout << "hello world!" << endl;
```

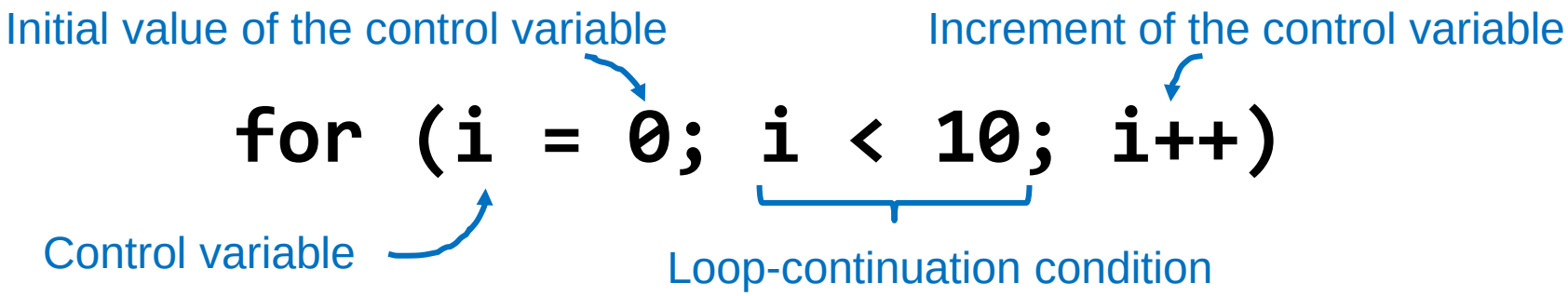


```
int i = 0;

for (i = 0; i < 10; i++) {
        cout << "hello world!" << endl;
}
```

`for` statement

- Syntax
  - ```
    for (initialization; condition; update ) {
      /* for loop body */
    }
    ```
- Example
  -



- The `initialization`, `condition`, and `update` all work on the same variable
    ```

`for` statement

- What happens in the loop `for (i = 0; i < 10; i++)`
  1. `i` is assigned the value `0`. This is the start value of the for loop
  2. The condition `(i < 10)` is evaluated.
     - The condition contains the end value of the for loop, in this case `10`.
     - What happens next depends on whether the condition is true or false

3a. If the condition is true then the following happens
- The body of the for loop executes
- The value of i is updated (`i++`)
- Repeat step 2 with the new value of `i`

3b. If the condition is false, then the for loop terminates and execution picks up after the `}`

`for` statement

- Another example:
  - Print the squares of number between 1 and 100 that are divisible by 3
  - Solution

```
int i = 0;

for (i = 1; i <= 100; i++) {
        if (i % 3 == 0)
                cout << i << "^2 = " << i * i << endl;
}
```

  - Or you could write the loop as

```
int i = 0;

for (i = 3; i <= 100; i += 3) {
        cout << i << "^2 = " << i * i << endl;
}
```

  - `i += 3` is shorthand for the statement `i = i + 3`

Iteration

- while statement syntax
  - ```
    while (condition) {
      /* while body */
    }
    ```
- for statement syntax
  - ```
    for (initialization; condition; update ) {
      /* for loop body */
    }
    ```
- for statement is closely related to the while statement.
  - for loop can be replaced by an equivalent while loop
  - ```
    initialization;
    while (condition) {
            /* while body */
            update;
    }
    ```
    ```

Increment/Decrement Operators

Postfix Operator

- Example

```
int main(void)
{
        int a;
        int b = 8;

        a = b++;

        /* what does this print?  */
        cout << "a = " << a << "; b = " << b << endl;
        cout << "a = " << a++ << "; b = " << b++ << endl;
        cout << "a = " << a << "; b = " << b << endl;
        return 0;
}
```

  - postfix operator x++, increments x after its value is used

Prefix Operator

- Example

```
int main(void)
{
        int a;
        int b = 8;

        a = ++b;

        /* what does this print?  */
        cout << "a = " << a << "; b = " << b << endl;
        cout << "a = " << ++a << "; b = " << ++b << endl;
        cout << "a = " << a << "; b = " << b << endl;
        return 0;

}
```

  - prefix operator x++, increments x before its value is used

Exercise

```
int a = 0;
int b = 8;

a = b--;

a = --b;
b += 2;

/* what does this print?  */
cout << "a = " << a << "; b = " << b << endl;
cout << "a = " << --a << "; b = " << --b << endl;

b -= 3;
cout << "a = " << a-- << "; b = " << b-- << endl;

b *= 4;
cout << "a = " << ++a << "; b = " << ++b << endl;

b /= 7;
cout << "a = " << a++ << "; b = " << b++ << endl;

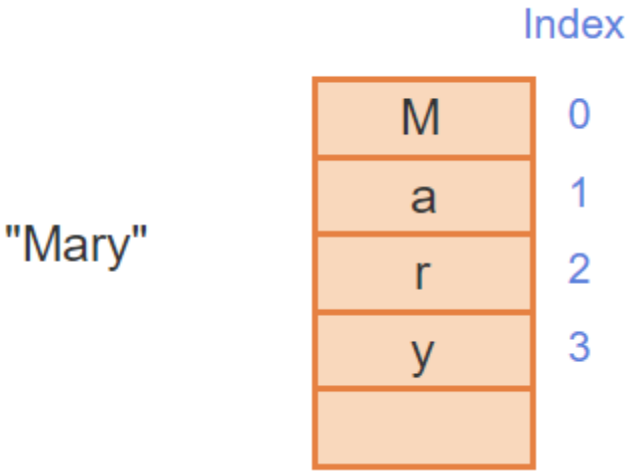
cout << "a = " << a << "; b = " << b << endl;

return 0;
```

String Operations

String Access Operations

- String character indices
  - A string is a sequence of characters in memory.
  - Each string character has a position number called an index, starting with 0



- Access the string character with index operator
  - `cout << userWord[3];`

String Access Operations

- Access the string character with **`at(pos)`** member function
  - Syntax: **`someString.at(pos)`**
  - **`pos`**: the character position in the string
- Example

```
#include <iostream>
#include <string>
using namespace std;

int main() {
   string userWord;

   cout << "Enter a 5-letter word: ";
   cin  >> userWord;

   cout << "Scrambled: ";
   cout << userWord.at(3);
   cout << userWord.at(1);
   cout << userWord.at(4);
   cout << userWord.at(0);
   cout << userWord.at(2);
   cout << endl;

   return 0;
}
```

```
Enter a 5-letter word: water
Scrambled: earwt

...

Enter a 5-letter word: Quick
Scrambled: cukQi

...

Enter a 5-letter word: 98765
Scrambled: 68597
```

The difference is that the at function checks to see if your index is within bounds, so be careful using []. Using [] is faster since it does no checking.

Changing a Character in a String

- Using index

```
string s = "hello";

s[0] = 'H';
cout << s << endl;
```

- Using `at()`

```
string s = "hello";

s.at(0) = 'H';
cout << s << endl;
```

String Length

- Using `size()` or `length()`
  - Example:

```
string s = "hello";

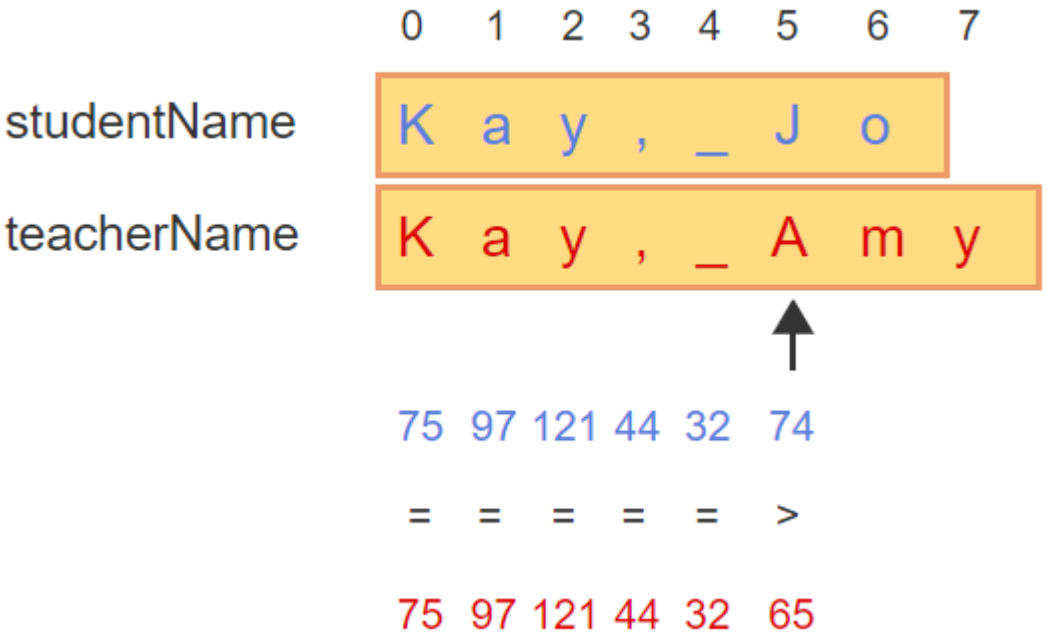
cout << "s.size(): " << s.size() << endl;
cout << "s.length(): " << s.length() << endl;
```

- Difference?
  - `size()` and `length()` are synonyms
  - `size()` is there to be consistent with other STL containers (like `vector`, `map`, etc.)
  - `length()` is to be consistent with most peoples' intuitive notion of character strings

https://stackoverflow.com/questions/905479/stdstring-length-and-size-member-functions

String Comparison

- Equal strings have the same number of characters, and each corresponding character is identical
- Strings are sometimes compared relationally (less than, greater than), as when sorting words alphabetically.
- A comparison begins at index 0 and compares each character until the evaluation results in false, or the end of a string is reached.



String Equality Example

- Using equality operator (==)

```
#include <iostream>
#include <string>
using namespace std;

int main() {
   string userWord;

   cout << "Enter a word: ";
   cin  >> userWord;

   if (userWord == "USA") {
      cout << "United States of America";
   }
   else {
      cout << userWord;
   }
   cout << endl;

   return 0;
}
```

```
Enter a word: Sally
Sally

...

Enter a word: USA
United States of America

...

Enter a word: usa
usa
```

Other String Operations

- Add a string (`s2`) at the end of an existing string (`s1`)
  - `s1.append(s2)`
  - Example:

```
string s = "hello";

s.append(" world");
cout << s << endl;
```

- Find the index of a pattern's first occurrence
  - `s1.find(pattern)`

```
cout << s.find('l') << endl;    // returns 2 (1st occurrence)
cout << s.find("ll") << endl;   // returns 2
cout << s.find('a') << endl;    // returns string::npos
cout << s.find('l', 3) << endl; // start finding patter at index 3
```

  - `string::npos`: a static member constant value with the greatest possible value for an element of type `size_t`

https://cplusplus.com/reference/string/string/npos/

Other String Operations

- Retrieve the substring of a string
  - `s.substr(index, length)`
  - Example:

```
string s = "hello";

cout << s.substr(0, 3) << endl;
```

- Other modification methods are listed in Table 3.17.2

Character Operations

- Character-handling Library (<cctype>)
  - Functions that perform useful tests and manipulations of character data.

isalpha(c)	true if alphabetic: a-z or A-Z	`isalpha('x') // true`
		`isalpha('6') // false`
		`isalpha('!') // false`
				toupper(c)	Uppercase version	`letter = toupper('a') // A`
						`letter = toupper('A') // A`
						`letter = toupper('3') // 3`

isdigit(c)	true if digit: 0-9.	`isdigit('x') // false`
		`isdigit('6') // true`
				tolower(c)	Lowercase version	`letter = tolower('A') // a`
						`letter = tolower('a') // a`
						`letter = tolower('3') // 3`

isspace(c)	true if whitespace.	`isspace(' ') // true`
		`isspace('\n') // true`
		`isspace('x') // false`				

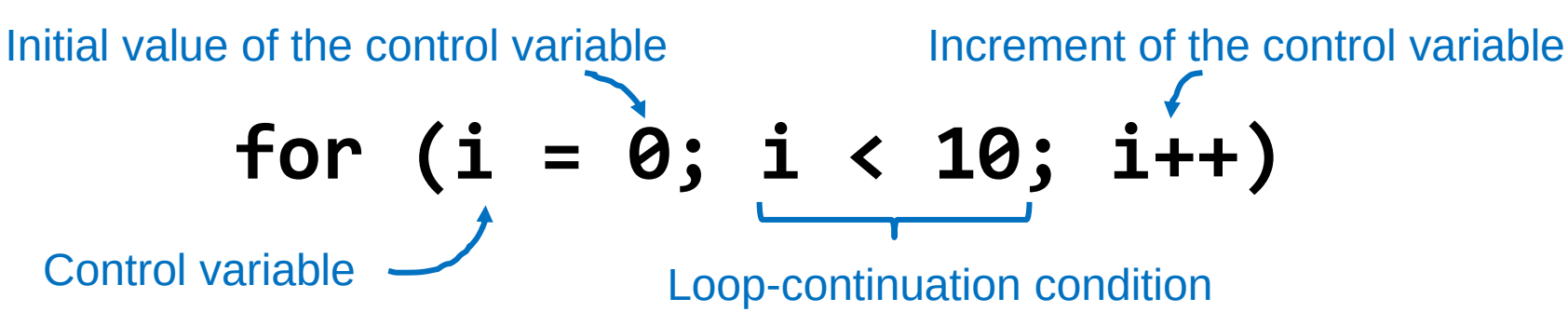
Note: Above, false is zero, and true is non-zero.

For Each Statement

Review - Structured Control Flow

- Selection Statements in C++
  - `if … else` selection statement:
    Performs an action if a condition is true and perform a different action if the condition is false
  - `switch` selection statement:
    Performs one of many different actions, depending on the value of an expression.
- Iteration Statements in C++
  - `while`
  - `do … while`
  - `for`
  - Foreach
- Jump Statements
  - `break`, `continue` statements

Review - **`for`** statement

- Syntax
  - ```
    for (initialization; condition; update ) {
      /* for loop body */
    }
    ```
- Example
  - 

- The `initialization`, `condition`, and `update` all work on the same variable
    ```

Using for Statement in a String

- Please write a C++ program to calculate how many uppercase letters are in a string

```
#include <iostream>
#include <cctype>

using namespace std;

int main() {
    string s;
    int count = 0;

    cout << "Please enter a string: " << endl;
    getline(cin, s);









    cout << "There are " << count << " upper case letters in the string" << endl;

    return 0;
}
```

Using `for` Statement in a String

- Please write a C++ program to calculate how many uppercase letters are in a string

```
#include <iostream>
#include <cctype>

using namespace std;

int main() {
    string s;
    int count = 0;

    cout << "Please enter a string: " << endl;
    getline(cin, s);

    for (int i = 0; i < s.length(); i++) {
        if (isupper(s.at(i))) {
            count++;
        }
    }

    cout << "There are " << count << " upper case letters in the string" << endl;

    return 0;
}
```

The Foreach Loop in C++

- Iterate over the elements of a containers (array, vectors etc) quickly without performing initialization, testing and increment/decrement
- Introduced in C++ 11
- Syntax

```
for (data_type  variable_name : container) {
     operations using variable_name
}
```

- Example:

`for` Statement

```
for (int i = 0; i < s.length(); i++) {
    if (isupper(s.at(i))) {
        count++;
    }
}
```

Foreach statement

```
for (char c : s) {
    if (isupper(c)) {
        count++;
    }
}
```

https://www.geeksforgeeks.org/g-fact-40-foreach-in-c-and-java/

Floating-point Comparison

Review - Floating-Point Variables

- Floating-Point Variables in C/C++
Declaration	Size	Supported number range
float x;	32 bits	-3.4×10^{38} to 3.4×10^{38}
double x;	64 bits	-1.7×10^{308} to 1.7×10^{308}

Check the example in Figure 2.19.1

- Choosing a variable type (**`double`** vs. **`int`**)
  - Integer variables are typically used for values that are counted, like 42 cars, 10 pizzas, or -95 days.
  - Floating-point variables are typically used for measurements, like 98.6 degrees, 0.00001 meters, or -55.667 degrees.
  - Floating-point variables are also used when dealing with fractions of countable items, such as the average number of cars per household
- Inaccurate in floating-point data
  - https://www.baeldung.com/cs/floating-point-numbers-inaccuracy

Avoid using == in Floating-Point Numbers

- Some floating-point numbers cannot be exactly represented in the limited available memory bits
- Floating-point numbers expected to be equal may be close but not exactly equal.
- Example:

```
numMeters = 0.7;
numMeters = numMeters - 0.4;
numMeters = numMeters - 0.3;

// numMeters expected to be 0,
// but is actually 0.00000000000000555112

 if (fabs(numMeters - 0.0) < 0.001) {
    // Equals 0.
 }
 else {
    // Does not equal 0.
 }
```

Expected	Actual	
0.7	0.69999999999999999555910790	
0.4	0.40000000000000000222044605	
0.3	0.29999999999999999888977697	
0	-0.00000000000000000555111512	numMeters

```
if (numMeters == 0.0) {
   // Equals 0.
}
else {
   // Does not equal 0.
}
```

Bug

Floating-point Comparison

- Should be compared for "close enough“
  - E.g., If (x - y) < 0.0001, x and y are deemed equal.
  - The difference threshold indicating that floating-point numbers are equal is often called the epsilon
  - Because the difference may be negative, the equation should be

$$| x - y | < \varepsilon$$

- Compare floating-point numbers in C++
  - `fabs(x - y) < 0.0001`

Jump Statements - `break` and `continue`

Review - Structured Control Flow

- Selection Statements in C++
  - `if … else` selection statement:
    Performs an action if a condition is true and perform a different action if the condition is false
  - `switch` selection statement:
    Performs one of many different actions, depending on the value of an expression.
- Iteration Statements in C++
  - `while`
  - `do … while`
  - `for`
  - Foreach
- Jump Statements
  - `break`, `continue` statements

`break` Statement

- Used to jump out of a `while`, `do`, or `for` loop
- Example: check whether a number n is prime

```
for (i = 2; i < n; i ++)
        if (n % i == 0)
                break;

if (i < n)
    cout << n << " is divisible by " << i << endl;
else
    cout << n << " is prime" << endl;
```

  - Is useful in which the exit point is in the middle of the body rather than at the beginning or end.

`break` Statement

- Example: Which enclosing statements the `breaks` jump out?

```
int n;
int i;

while (1) {
    cout << "enter a number (enter a negative number to exit): " ;
    cin >> n;

          for (i = 0; i < n; i++) {
                    if (i > 4) {
                        cout << "I can only count to 5!" << endl;
                              break;
                    }
                    cout << (i + 1) << endl;
                    cout << "in for loop" << endl;
          }

          if (n < 0) {
                    break;
          }
          cout << "still in while loop" << endl;
}
```

`break` Statement

- The break statement transfers control out of the innermost enclosing `while`, `do`, `for`, or `switch` statement
  - Example:

```
while (...)
{
        switch (...) {
                ...
                break;
                ...
        }
}
```

  - The `break` statement transfers control out of the `switch` statement, but not out of the `while` loop.

`continue` Statement

- `break` statement
  - Transfer the control past the end of the loop
  - Control leaves the loop
  - Can be used in switch statement and loops
- `continue` statement
  - Transfer the control to a point just before the end of the loop body
  - Control remains inside the loop
  - Is limited to loops

`continue` Statement

- Example

```
while (1) {
  cout << "enter a char (~ to say goodbye): ";

  cin >> c;

  if (c >= 'a' && c <= 'z')
    cout << "lowercase digit: ASCII: " << c << " " << static_cast<int>(c) << endl;
  else if (c >= 'A' && c <= 'Z')
    cout << "uppercase digit: ASCII: " << c << " " << static_cast<int>(c) << endl;
  else if (c >= '0' && c <= '9')
    continue;
  else if (c == '~') {
    cout << "Goodbye..." << endl;
    break;
  }
  else
    cout << "something other than a tilde, lowercase, uppercase, or digit entered"
         << endl;
}
```

`goto` Statement

https://xkcd.com/292/

- Use a lot of `goto` statements easily leads to spaghetti code
  - The research of Bohm and Jacopini had demonstrated that programs could be written without any goto statement in 1966.

Bitwise Operator

Review - Logical Expression

- Logical Operators

a	b	a AND b
false	false	false
false	true	false
true	false	false
true	true	true

a	b	a OR b
false	false	false
false	true	true
true	false	true
true	true	true

a	NOT a
false	true
true	false

- Examples

Let x = 7, y = 9

(x > 0) AND (y < 10) true
true true

(x > 0) AND (y < 5) false
true false

(x < 0) OR (y > 10) false
false false

(x < 0) OR (y > 5) true
false true

NOT (x < 0) true
false

NOT (x > 0) false
true

Bitwise Operators

- Usage
  - Manipulate the bits of integral operands
- List of bitwise operators

Operator		Description
&	**bitwise AND**	Compares its two operands bit by bit. The bits in the result are set to 1 if the corresponding bits in the two operands are *both* 1.
|	**bitwise inclusive OR**	Compares its two operands bit by bit. The bits in the result are set to 1 if *at least one* of the corresponding bits in the two operands is 1.
^	**bitwise exclusive OR** (also known as **bitwise XOR**)	Compares its two operands bit by bit. The bits in the result are set to 1 if the corresponding bits in the two operands are different.
<<	**left shift**	Shifts the bits of the first operand left by the number of bits specified by the second operand; fill from the right with 0 bits.
>>	**right shift**	Shifts the bits of the first operand right by the number of bits specified by the second operand; the method of filling from the left is machine dependent when the left operand is negative.
~	**complement**	All 0 bits are set to 1 and all 1 bits are set to 0.

Bitwise Shift Operators

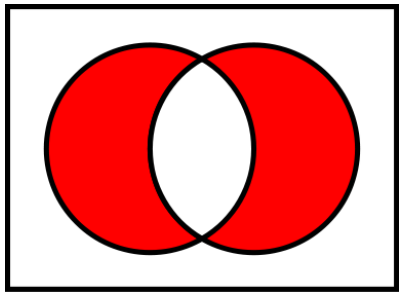
- Syntax
  - `i << j`
    `i` are shifted left by `j` places
  - `i >> j`
    `i` are shift right by `j` places
- Example
  - ```
    unsigned short i, j;
    i = 13;           /* i is now 13 (binary 0000000000001101) */
    j = i << 2;       /* j is now 52 (binary 0000000000110100) */
    j = i >> 2;       /* j is now 3  (binary 0000000000000011) */
    ```
  - Use compound assignment operators instead:
    ```
    i = 13;   /* i is now 13 (binary 0000000000001101) */
    i <<= 2;  /* i is now 52 (binary 0000000000110100) */
    i >>= 2;  /* j is now 3  (binary 0000000000000011) */
    ```
    ```

Bitwise Operators - ~, &, ^, |

- And (&), Inclusive Or (|), and Complement (~)

A	B	A & B	A | B	~A
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

- Exclusive Or (^)
  - Outputs true only when inputs differ



A	B	A ^ B
1	1	0
1	0	1
0	1	1
0	0	0

Bitwise Operators - ~, &, ^, |

- Example

```
unsigned short i, j, k;
i = 21;       /* i is now 21     (binary 0000000000010101) */
j = 56;       /* j is now 56     (binary 0000000000111000) */
k = ~i;       /* k is now 65514  (binary 1111111111101010) */
k = i & j;    /* k is now 16     (binary 0000000000010000) */
k = i ^ j;    /* k is now 45     (binary 0000000000101101) */
k = i | j;    /* k is now 61     (binary 0000000000111101) */
```

- Compound assignment operators
  - &=
  - ^=
  - |=

Using Bitwise Operators to Access Bits

- Setting a bit:
  - Use or operator

```
i = 0x0000;         /* i is now 0000000000000000 */
i |= 0x0010;        /* i is now 0000000000010000 */
```

  - Use shift operator

```
i = 0x0000;         /* i is now 0000000000000000 */
j = 3;
i |= 1 << j;        /* i is now 0000000000001000 */
```

- Cleaning a bit
  - Use and operator

```
i = 0x00ff;         /* i is now 0000000011111111 */
i &= ~0x0010;       /* i is now 0000000011101111 */
```

  - Use shift operator

```
i = 0x00ff;         /* i is now 0000000011111111 */
j = 3;
i &= ~(1 << j);     /* i is now 0000000011110111 */
```

Using Bitwise Operators to Access Bits

- Testing a bit
  - Use and operator

```
if (i & 0x0010) ... /* tests bit 4 */
```

  - Use shift operator

```
if (i & (1 << j)) ...        /* tests bit j */
```

- Example

```
#define BLUE      1
#define GREEN     2
#define RED       4

i |= BLUE;                   /* sets BLUE bit */
i &= ~BLUE;                  /* clears BLUE bit */
if (i & BLUE);               /* tests BLUE bit */

i |= BLUE | GREEN;           /* sets BLUE and GREEN bit */
i &= ~(BLUE | GREEN);        /* clears BLUE and GREEN bit */
if (i & (BLUE | GREEN));     /* tests BLUE and GREEN bit */
```

Enumerations

Enumerations

- Usage
  - Need variables that have only a small set of meaningful values
  - Example:
    - Boolean variable: only have two possible values, which are true and false
    - The suit of a playing card: only have four potential values, which are clubs, diamonds, hearts, and spades
- Previous methods:
  - An integer with a set of codes that represent the possible values:

```
int s; /* s will store a suit */
s = 2; /* 2 represents “hearts” */
```

  - Use macros to define a suit type and names for the suits:

```
#define SUIT int
#define CLUBS            0
#define DIAMONDS 1
#define HEARTS           2
#define SPADES           3
```

```
SUIT s;
s = HEARTS;
```

Enumerations

- Enumerated Type
  - A type whose values are listed ("enumerated") by the programmer
  - Syntax:

    ```
    enum identifier {enum_constant_list};
    ```

  - Example:
    - `enum suit {CLUBS, DIAMONDS, HEARTS, SPADES};`
    - The compiler assigns the integers 0, 1, 2, … to the constants in the particular enumeration.
  - Another way to assign values incremented from 1

    ```
    enum suit {CLUBS = 1, DIAMONDS, HEARTS, SPADES};
    ```

  - Choose different values for enumeration constants
    - Example:

```
enum suit {
        CLUBS = 10,
        DIAMONDS = 20,
        HEARTS = 30,
        SPADES = 40
};
enum suit s1 = CLUBS;
```

Removed Slides

Short Circuiting Evaluation

- If the machine can figure out if the expression is true or false with only partial information, the machine stops evaluating the expression
- Example:
  - `(A || B)` → if `A` is `TRUE`, is there any reason to evaluate `B`?
  - `(A && B)` → if `A` is `FALSE`, is there any reason to evaluate `B`?
- Do not use assignment in the if expression unless it is the first expression
  → There is no guarantee that the other expression will be evaluated
  - `if ((x = foo()) && (y > 9))`
    → this is ok
  - `if ((y > 9) && (x = foo()))`
    → `x = foo()` may never execute