Fundamentals of Structured Programming

Course Logistics – Staff

- **Teachers**
  - Prof. Zaki Taha (Computer Science Department)
  - Dr. Sally Saad
    - **Contact:** SallySaad@gmail.com
    - **Office:** FCIS, 3rd floor, "glass room".
    - **Office hours:** **Sunday** 11:00-13:00
- **Credit to Dr.Salma Hamdy for content preparation**
- **Course Dropbox Folder**

https://www.dropbox.com/sh/85vnrgkfqgrzhwn/AABdwKLJZqZs26a7u-y0AFwia?dl=0

Course Logistics – General

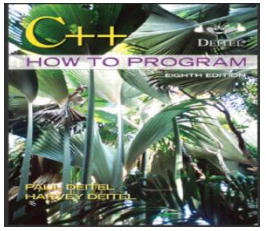
- **Class meets:** Wednesday **(G1)** 8:00-11:00 - Dr. Fahmy Tolba Hall
  (SW) 12:00-14:00 - Class 3
  Thursday **(BIO)** 8:00-11:00 -Dr. Saeid Abd ElWahab Hall
  (G2) 12:00 – 15:00 -Geneidy Hall
- **(3-hour lecture + 2-hour lab) a week.**
- **(SW: 2-hour lecture + 2 hour lab) a week.**
- **Assessment (125 points)**

Final Term Examination	90
Midterm	10
Practical Work	25
(Lab Tasks, Project, Practical Exam)	
Bonus	5

Course Logistics – Textbook

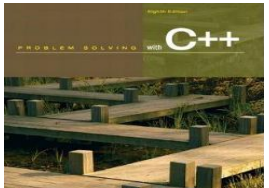
Available References:

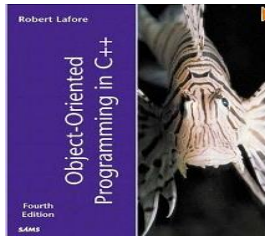
- **C++ How to Program**, 5th edition, Deitel & Deitel, Prentice Hall- Pearson Education International, 2005.
- **Competitive Programming, Handbook for ACM ICPC and IOI Contestants,** Steven Halim, Felix Halim, 2013. *(Recommended by the late **Youssef El-Kayyali- ACM Head of Training Committee**)*



Other References

- **Problem Solving with C++**, 8th edition, Walter Savitch, Addison Wesley-Pearson Education International, 2012.
- **Object Oriented Programming in C++**, 4th edition, Robert Lafore, Pearson Education, 2002.
- **Internet**: ENDLESS list of C++Tuturials.





Course Logistics – Ethics

Lecture

1. Time management
2. Attendance
3. Manners
4. Mind + hands (if possible)
5. Your rights.
6. My rights.
7. Questions

Lab

1. Time management
2. Attend ONLY in your class.
3. Manners
4. Mind + hands (BOTH!!!)
5. Your rights.
6. TA's rights.
7. Questions

Cheating and copied assignments, programs, projects, or code segments, will not be tolerated.

Course Logistics – Outline

1. Introduction + Review on previous skills
2. Arrays (1D) + Tracing and Debugging your Program
3. Arrays (2D)
4. Struct + Array of struct
5. Problem Solving
6. Functions I(Built-in) Math+ Strings + File Streams
7. Functions II (Pass by value) + Function Overloading.
8. Functions III (Pass by reference)
9. Pointers I
10. Pointers II
11. More Problem Solving

Course Logistics – Outcomes

During and upon the completion of this course, you will:

- Have a basic understanding of structured programming concepts and how computers understand your code.

- Know how to design, analyze, and code simple and sophisticated programs using C++ language that execute efficiently.

- Make use of simple data structures, linear algebra, and other knowledge to solve real problems.

- Require substantial programming and good software engineering practice.

- Develop and deploy a C++ application of your own design.

- Meet deadlines and be a team player.

- Be able to present your work and describe your solutions.

Review

Contents

1. Problem Solving
2. Programming Languages
3. Programming Techniques
4. Structured Programming using C++
5. Control Structures

Why Programming?

1. ..
2. ..
3. ..

…

Are we only developers?

Our Career is not only in implementing the code!

You can work as

1. Software Analyst (System Analyst)
2. Software Designer
3. Software Developer
4. Quality Engineer (Software Tester)

And many other titles…

1. Problem Solving

Steps of Solving a Problem:

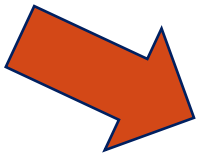
1. Define the problem
   - Understand input and output
   - Change the problem
   - Simplify the problem

1. Problem Solving – (cont.)

Steps of Solving a Problem:

1. Define the problem
2. Design a solution
   - How to reach output from input
   - Plan a solution
   - List steps of solution

Analysis



Algorithm

1. Problem Solving – (cont.)

- An algorithm is a procedure for solving a problem in terms of the actions to be executed and the order in which those actions are to be executed.
- An algorithm is merely the sequence of steps taken to solve a problem.
- The steps are normally "sequence," "selection, " or "iteration," statements.

1. Problem Solving – (cont.)

- A pseudo code is a kind of structured English *for describing algorithms*. It allows the designer to focus on the logic of the algorithm *without being distracted by details of language syntax (language independent)*.
- The pseudo code needs to be complete. It describes the entire logic of the algorithm so that implementation becomes a task of translating line by line into source code.

1. Problem Solving – (cont.)

- **Pseudo code common format**

Input: READ, OBTAIN, GET
Output: PRINT, DISPLAY, SHOW
Compute: COMPUTE, CALCULATE, DETERMINE
Initialize: SET, INIT
Add one: INCREMENT, BUMP

NOT a language syntax

1. Problem Solving – (cont.)

- **Pseudo code common format**

IF-THEN-ELSE

```
IF condition THEN
        sequence 1
ELSE
        sequence 2
ENDIF
```

WHILE

```
WHILE condition
        sequence
ENDWHILE
```

FOR

```
FOR iteration bounds
        sequence
ENDFOR
```

NOT a language syntax

1. Problem Solving – (cont.)

Example 1: Student pass?

1. GET Grade
2. IF Grade is greater than or equal to 50 THEN
3. Print "passed"
4. ELSE
5. Print "failed“
6. ENDIF

1. Problem Solving – (cont.)

Example 2: Average of ten grades

1. SET Total to zero
2. SET Grade Counter to one
3. WHILE Grade Counter is less than or equal to ten THEN
    4. GET the Next Grade
    5. ADD the Next Grade to the Total
6. ENDWHILE
7. SET the Class Average to the Total divided by

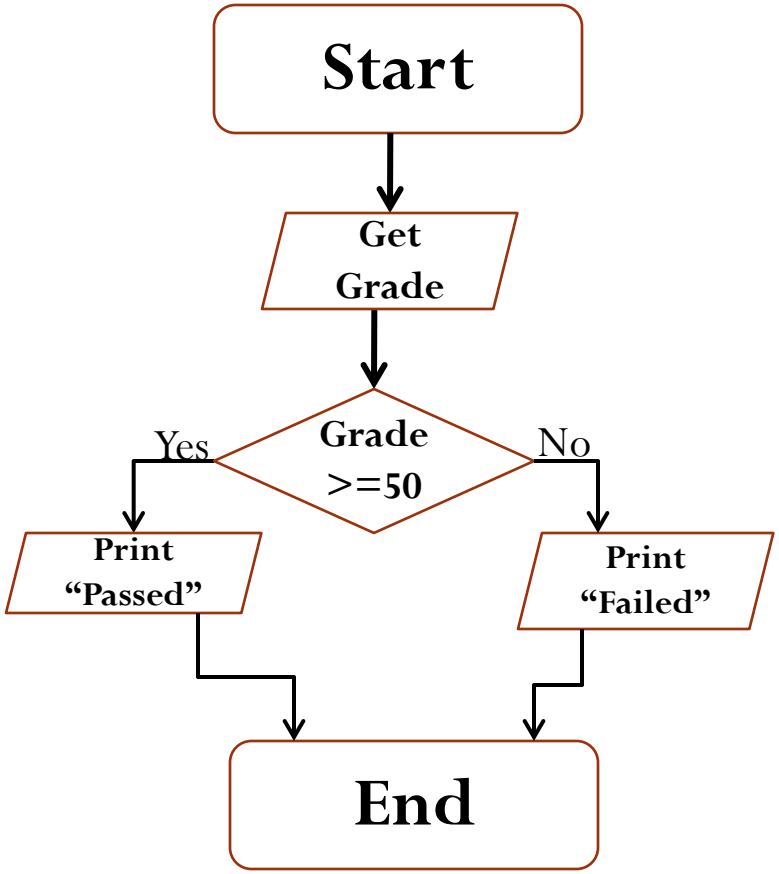
1. Problem Solving – (cont.)

- A **flowchart** is diagram of the steps you will use to reach your goal (steps of the algorithm).
- Various standard shapes.

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

1. Problem Solving – (cont.)

Example 1: Student pass



1. Problem Solving – (cont.)

Example 2: Design a flow chart to compute the Average of ten grades (Bonus)

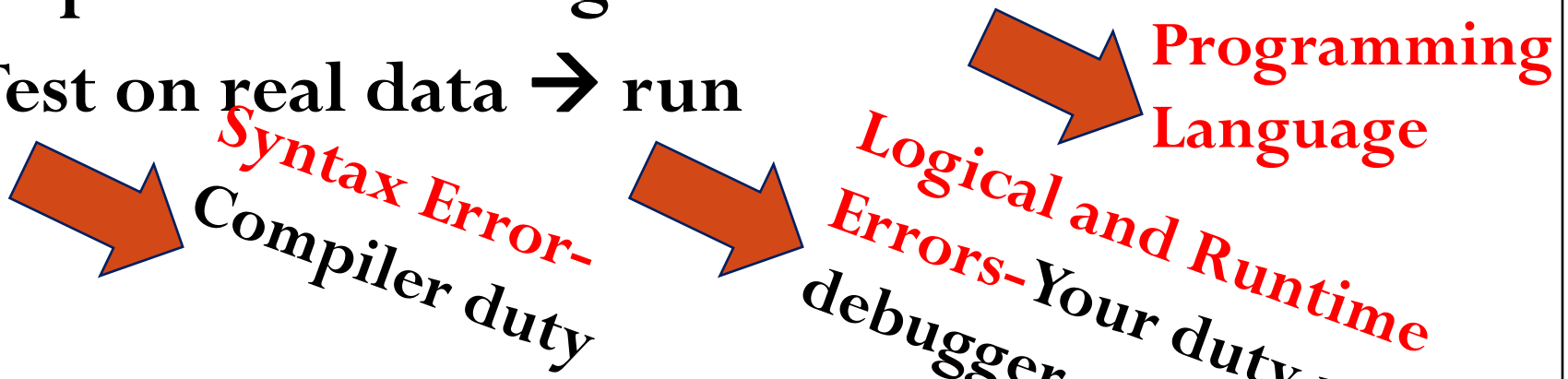
Conclusion:

Pseudo Code Vs. Flowcharts ?!

1. Problem Solving – (cont.)

Steps of Solving a Problem:

1. Define the problem.
2. Design a solution.
3. Test on sample data → trace
4. Implement the "algorithm"
5. Test on real data → run



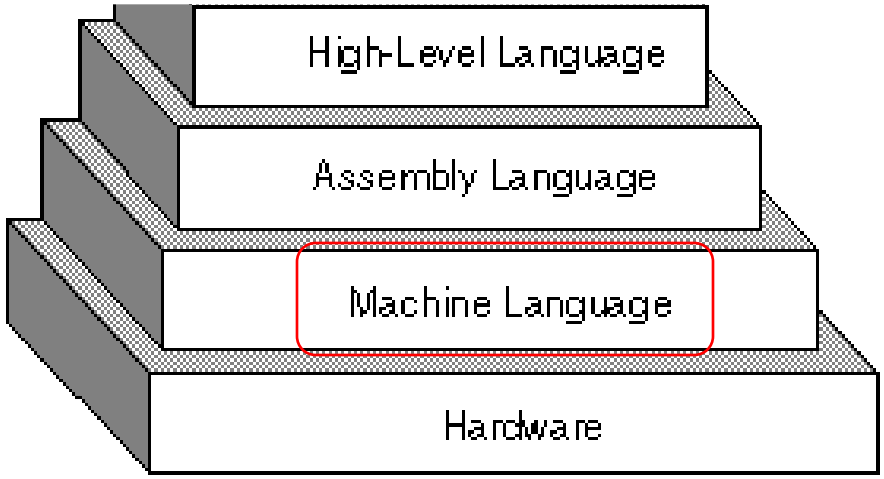
6. Modify your solution / algorithm and Enhance your code to utilize less resources.

2 Programming Languages

- Talking to your Computer

1- Machine Language

```
00  2007  # jump past sensor and constant locations
01  0000  # read right sensor value here
02  0000  # read left sensor value here
03  0000  # write right motor power level here
04  0000  # write left motor power level here
05  0000  # store motor-off constant here
06  0100  # store motor-on constant here
07  4011  # load right sensor value into register 1
08  4022  # load left sensor value into register 2
09  1123  # subtract 1 from 2 and store result in 3
10  4105  # load motor-off constant into register 1
11  4206  # load motor-on constant into register 2
12  3316  # if the left sensor is greater than the
13  5203  # right then turn the right motor on
14  5104  # and turn the left motor off
15  2007  # and then jump to beginning of the loop
16  5204  # else turn the left motor on
17  5103  # and turn the right motor off
18  2007  # and then jump to beginning of the loop
19  0000  # this location is not used by the program
```



- Directly understandable by computer.
- String of numbers (1s and 0s)
- Machine dependant.
- Too tedious for programmer.
- Too many instructions.

2 Programming Languages – (cont.)

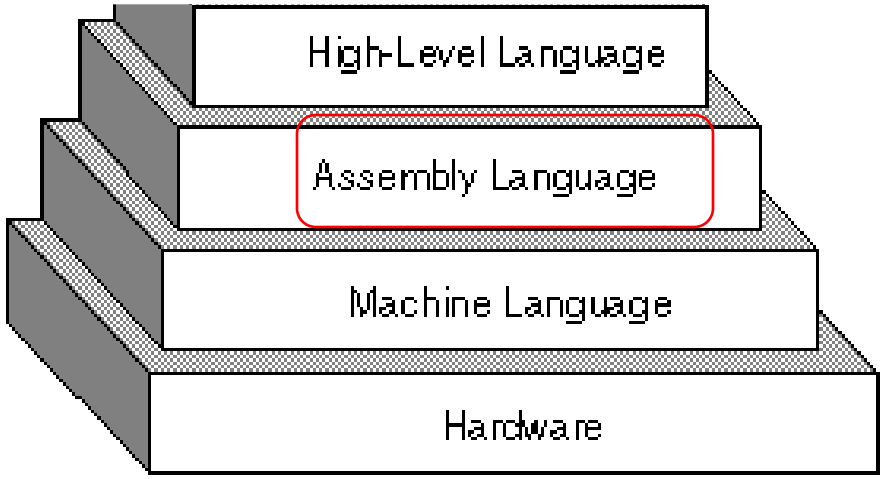
- Talking to your Computer

1- Machine Language

2- Assembly Language

```
i = j + k;                1    ILOAD j    // i = j + k     0x15 0x02
if (i == 3)               2    ILOAD k                    0x15 0x03
    k = 0;                3    IADD                       0x60
else                      4    ISTORE i                   0x36 0x01
    j = j - 1;            5    ILOAD i    // if (i < 3)    0x15 0x01
                          6    BIPUSH 3                   0x10 0x03
                          7    IF_ICMPEQ L1               0x9F 0x00 0x0D
                          8    ILOAD j    // j = j - 1     0x15 0x02
                          9    BIPUSH 1                   0x10 0x01
                         10    ISUB                       0x64
                         11    ISTORE j                   0x36 0x02
                         12    GOTO L2                    0xA7 0x00 0x07
                         13 L1:          BIPUSH 0         // k = 0 0x10 0x00
                         14    ISTORE k                   0x36 0x03
                         15 L2:
```

(a) (b) (c)



- English-like abbreviations.
- Requires assembler.
- Still too many instructions.

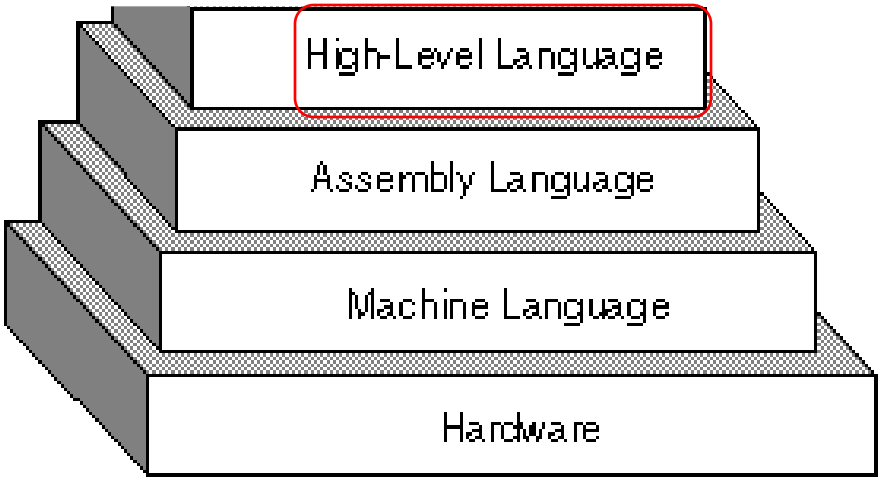
2 Programming Languages – (cont.)

- Talking to your Computer

1- Machine Language

2- Assembly Language

3- High Level Language



2 Programming Languages – (cont.)

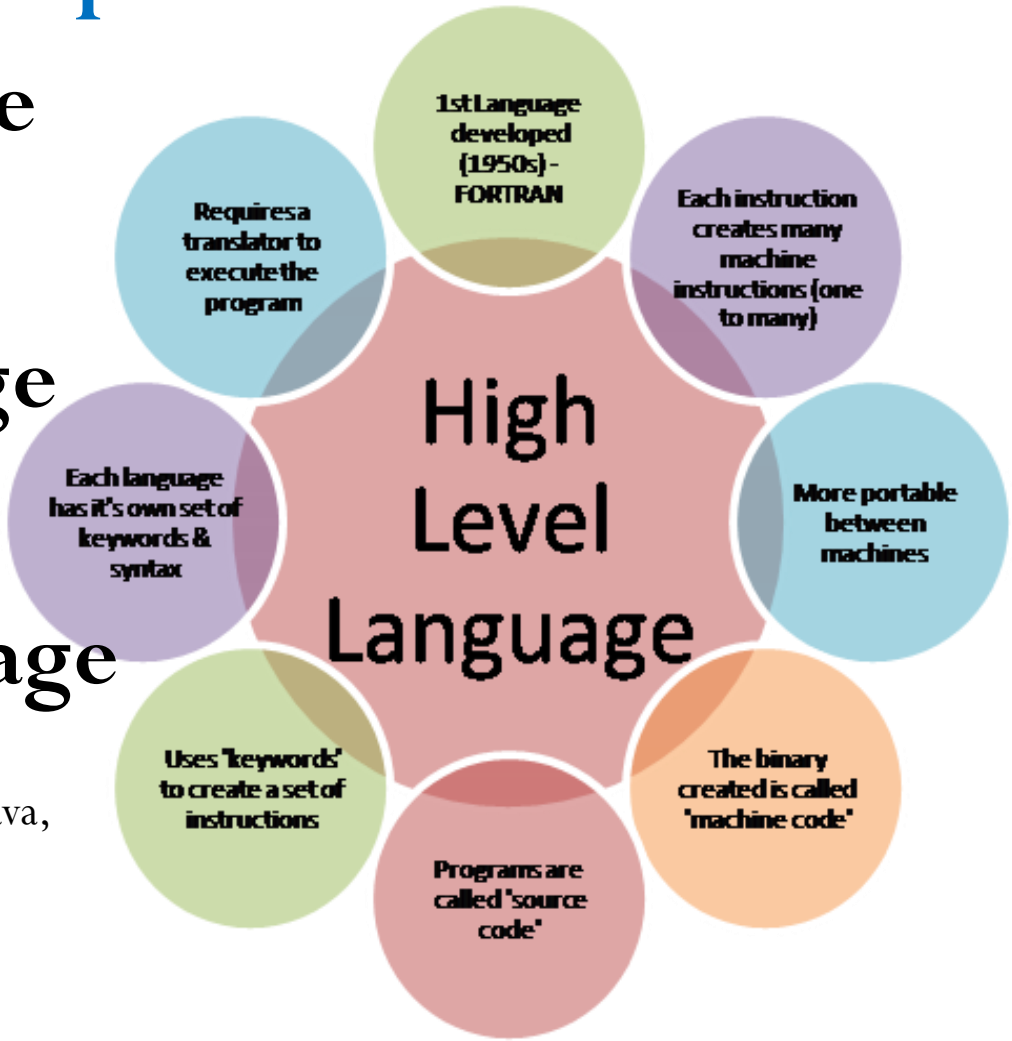
- Talking to your Computer

1- Machine Language

1- Assembly Language

3- High Level Language

C++, C, Microsoft's .NET languages (e.g., Visual Basic, Visual C++ and Visual C#), Java, and Python



3. Programming Techniques

Programming Techniques

1. Unstructured Programming
2. Procedural Programming
3. Modular Programming
4. Object Oriented Programming
5. Logic Programming

3. Programming Techniques – (cont.)

1.Unstructured Programming

- One main program.
- A sequence of commands or *statements*, which modify data that is *global* throughout the whole program.
- Disadvantages.

3 Programming Techniques – (cont.)

2.Procedural Programming

- A single program, which is divided into small pieces called procedures.
- Combine returning sequences of statements into one single place.
- A *procedure call* is used to invoke the procedure.
- Structured but not Object Oriented.

3 Programming Techniques – (cont.)

3.Modular (Structured) Programming

- Procedures of a common functionality are grouped together into separate *modules*.
- Program is now divided into several smaller parts which interact through procedure calls.
- Each module can have its own data. This allows each module to manage an internal *state*.
- One state per module

2.2 Programming Techniques – (cont.)

4.Object Oriented Programming (OOP)

- Treat data and the procedures that act upon the data as a single "object"--a self-contained entity with an identity and certain characteristics of its own.
  - Encapsulation
  - Data Hiding
  - Inheritance
  - Polymorphism

2.2 Programming Techniques – (cont.)

5.Logical Programming

- Focus on telling the system what to do, rather than how to do it.
- Describe problems in a declarative manner (versus procedural).
- Clues: Facts and rules.
- Solve problems by asking questions.
- Deductive reasoning and Predicate logic.
- Example: Prolog Language


Break

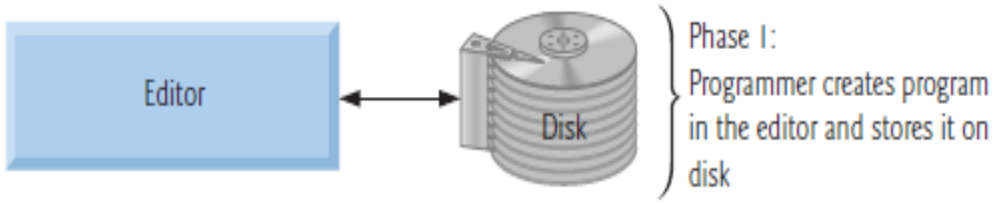
4. Structured Programming using C++

- C++ Integrated Development Environment (IDE)
- C++ programs consist of pieces called classes and functions.
- Even if you don't follow the OOP approach, you will still take advantage of the rich collections of classes and functions in the C++ Standard Library.

4. Structured Programming using C++

- **C++ Development Environment**

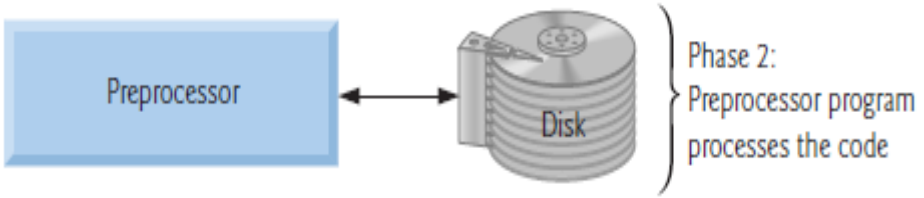
1- Creating Program



- Source code

- **Popular IDEs:** Microsoft® Visual Studio 2012 Express Edition, Dev C++, NetBeans, Eclipse.

2- Preprocessing



- Automatically

- Preprocessor directives (#include, #define)

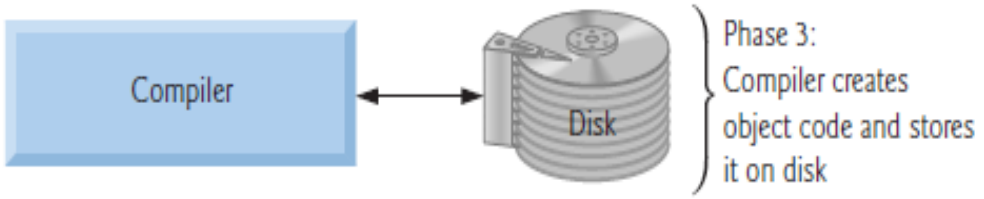
4. Structured Programming using C++

- **C++ Development Environment**

3- Compiling

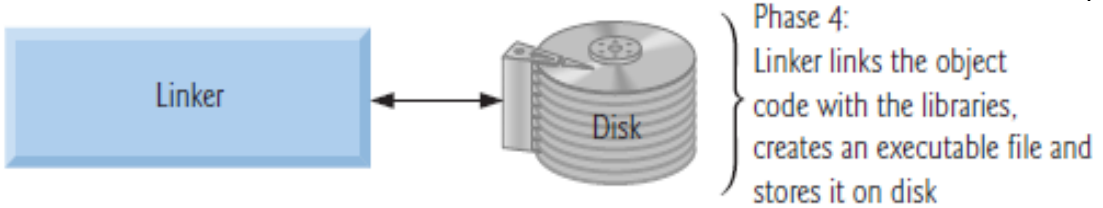
- Object code

(detecting syntax errors)



4- Linking

- Object code to needed parts

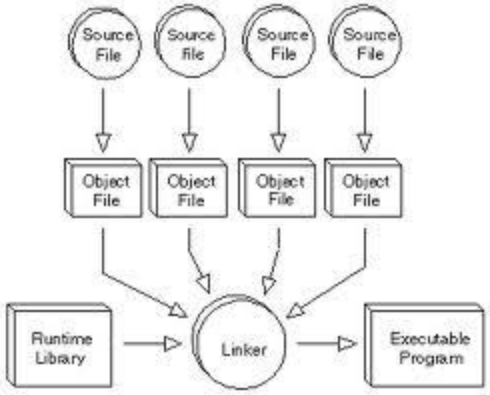


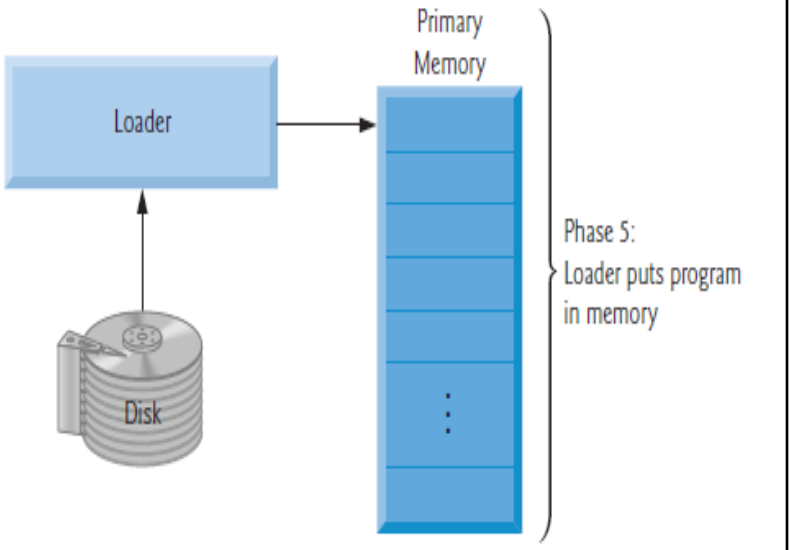
4. Structured Programming using C++

- C++ Development Environment

5- Loading

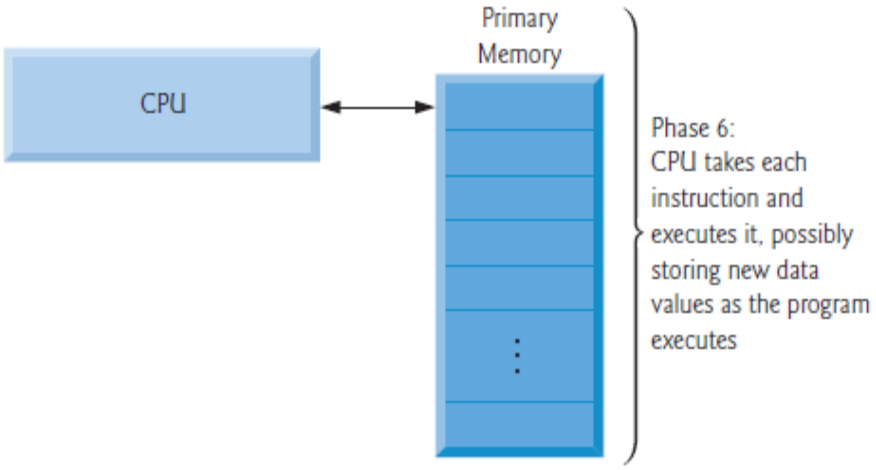
- With other needed libraries.





6- Executing

- One instruction at a time.



4. Structured Programming using C++

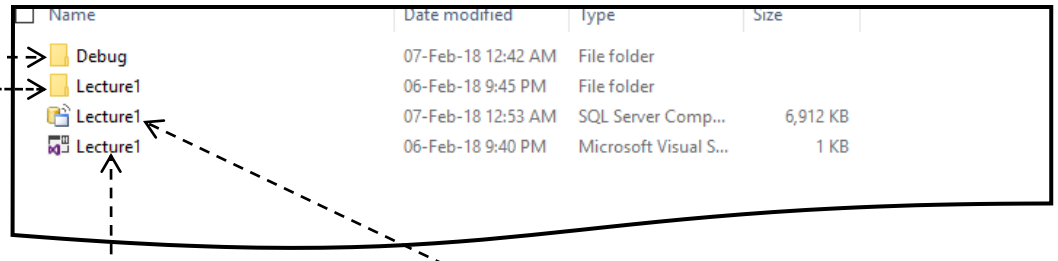
- **C++ Created Files/Folders**

Project Folder include:

Debug folder

*•Includes **.exe** file (Application itself)*

•Includes other files related to the debug or the release of the project



***.sln** file

Opens the whole solution on the IDE

database file

Beyond our scope ☺

Solution Folder

•Includes different Project(s) within your solution

*•Includes Source file(s)**.cpp file(s)***

*•Includes other projects files like **.h file(s)***

*•Includes other files like **log** files*

4. Structured Programming using C++

Display 1.1 A Sample C++ Program

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

3   int main( )
4   {
5       int numberOfLanguages;

6       cout << "Hello reader.\n"
7            << "Welcome to C++.\n";

8       cout << "How many programming languages have you used? ";
9       cin >> numberOfLanguages;

10      if (numberOfLanguages < 1)
11          cout << "Read the preface. You may prefer\n"
12               << "a more elementary book by the same author.\n";
13      else
14          cout << "Enjoy the book.\n";

15      return 0;
16  }
```

4. Structured Programming using C++

Display 1.1 A Sample C++ Program

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

3  int main( )
4  {
5      int numberOfLanguages;

6      cout << "Hello reader.\n"
7           << "Welcome to C++.\n";

8      cout << "How many programming languages have you used? ";
9      cin >> numberOfLanguages;

10     if (numberOfLanguages < 1)
11         cout << "Read the preface. You may prefer\n"
12              << "a more elementary book by the same author.\n";
13     else
14         cout << "Enjoy the book.\n";

15     return 0;
16 }
```

Libraries and namespaces

4. Structured Programming using C++

Display 1.1 A Sample C++ Program

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

3  int main( )
4  {
5      int numberOfLanguages;

6      cout << "Hello reader.\n"
7           << "Welcome to C++.\n";

8      cout << "How many programming languages have you used? ";
9      cin >> numberOfLanguages;

10     if (numberOfLanguages < 1)
11         cout << "Read the preface. You may prefer\n"
12              << "a more elementary book by the same author.\n";
13     else
14         cout << "Enjoy the book.\n";

15     return 0;
16 }
```

Libraries and namespaces

functions

4. Structured Programming using C++

Display 1.1 A Sample C++ Program

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

3  int main( )
4  {
5      int numberOfLanguages;

6      cout << "Hello reader.\n"
7           << "Welcome to C++.\n";

8      cout << "How many programming languages have you used? ";
9      cin >> numberOfLanguages;

10     if (numberOfLanguages < 1)
11         cout << "Read the preface. You may prefer\n"
12              << "a more elementary book by the same author.\n";
13     else
14         cout << "Enjoy the book.\n";

15     return 0;
16 }
```

Libraries and namespaces

functions

Input/output

4. Structured Programming using C++

Display 1.1 A Sample C++ Program

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

3  int main( )
4  {
5      int numberOfLanguages;

6      cout << "Hello reader.\n"
7           << "Welcome to C++.\n";

8      cout << "How many programming languages have you used? ";
9      cin >> numberOfLanguages;

10     if (numberOfLanguages < 1)
11         cout << "Read the preface. You may prefer\n"
12              << "a more elementary book by the same author.\n";
13     else
14         cout << "Enjoy the book.\n";

15     return 0;
16 }
```

Libraries and namespaces

functions

Variables

Input/output

3. C++ Programming – (cont.)

Display 1.1 A Sample C++ Program

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

3  int main( )
4  {
5      int numberOfLanguages;

6      cout << "Hello reader.\n"
7           << "Welcome to C++.\n";

8      cout << "How many programming languages have you used? ";
9      cin >> numberOfLanguages;

10     if (numberOfLanguages < 1)
11         cout << "Read the preface. You may prefer\n"
12              << "a more elementary book by the same author.\n";
13     else
14         cout << "Enjoy the book.\n";

15     return 0;
16 }
```

Libraries and namespaces

functions

Variables

Input/output

Control Structure

Variables - Declaration

- C++ Identifiers
  - Keywords/reserved words vs. Identifiers
  - Case-sensitivity and validity of identifiers
  - Meaningful names!
- Variables
  - A memory location to store data for a program
  - Must declare all data before use in program

VARIABLE NAMING Conventions

Variables – Data Types

Display 1.2 **Simple Types**

TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
`short` (also called `short int`)	2 bytes	–32,768 to 32,767	Not applicable
`int`	4 bytes	–2,147,483,648 to 2,147,483,647	Not applicable
`long` (also called `long int`)	4 bytes	–2,147,483,648 to 2,147,483,647	Not applicable
`float`	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
`double`	8 bytes	approximately 10^{-308} to 10^{308}	15 digits

Variables – Data Types

`long double`	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits
`char`	1 byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
`bool`	1 byte	`true`, `false`	Not applicable

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types `float`, `double`, and `long double` are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

Variables - Assignment

- Initializing data in declaration statement
  - Results "undefined" if you don't!
    - int myValue = 0;
- Assigning data during execution
  - Lvalues (left-side) & Rvalues (right-side)
    - Lvalues must be variables
    - Rvalues can be any expression
    - Example:
      distance = rate * time;
      Lvalue: "distance"
      Rvalue: "rate * time"

Variables - Assignment

- C++ offers some shorthand notations in assignment statements.
- Incrementing a variable: var = var+1;

Shorthand increment operator:

Post-increment: var++;

Or Pre-increment: ++var;

- Decrementing a variable: var= var-1;

Shorthand decrement operator:

Post-increment: var--;

Or Pre-decrement: --var;

Variables - Assignment

Example: Evaluate the following

```
Result=10
x=6
Result=12
x=6
```

```
#include <iostream>
using namespace std;
void main()
{
int x=5, result1, result2;
result1 = (x++) * 2;     //?
cout<<"Result="<<result1<<"\nx="<<x<<endl;
x=5;
result2= (++x) *2;   //?
cout<<"Result="<<result2<<"\nx="<<x<<endl;

}
```

Variables - Assignment

Assigning Data: Shorthand Notations

EXAMPLE	EQUIVALENT TO
`count += 2;`	`count = count + 2;`
`total -= discount;`	`total = total - discount;`
`bonus *= 2;`	`bonus = bonus * 2;`
`time /= rushFactor;`	`time = time/rushFactor;`
`change %= 100;`	`change = change % 100;`
`amount *= cnt1 + cnt2;`	`amount = amount * (cnt1 + cnt2);`

Variables - Assignment

Are we allowed to declare a variable of some type and assign to it a value of a different type?

- **Yes!**
- **And No!**

Variables - Assignment

- Compatibility of Data Assignments
  - Type mismatches
    - General Rule: Cannot place value of one type into variable of another type
  - intVar = 2.99; // 2 is assigned to intVar!
    - Only integer part "fits", so that's all that goes
    - Called "implicit" or "automatic type conversion"

Variables – Type Casting

- Two types
  - Implicit—also called "Automatic"
    - Done FOR you, automatically
      17 / 5.5
      This expression causes an "implicit type cast" to take place, casting the 17 → 17.0
  - Explicit type conversion
    - Programmer specifies conversion with cast operator
      (double)17 / 5.5
      Same expression as above, using explicit cast
      (double)myInt / myDouble
      More typical use; cast operator on variable
  - Scope of Variables (global vs. local)

Literals

- Literals
  - Examples:
    - 2 // Literal constant int
    - 5.75 // Literal constant double
    - "Z" // Literal constant char
    - "Hello World" // Literal constant string
- Considered constants: Cannot change values during execution
- Called "literals" because you "literally typed" them in your program!

Constants

- Naming your constants
  - Literal constants are "OK", but provide little meaning
    - e.g., seeing 24 in a programgm, tells nothing about what it represents
- Use named constants instead
  - Meaningful name to represent data
    const int NUMBER_OF_STUDENTS = 24;
    - Called a "declared constant" or "named constant"
    - Now use its name wherever needed in program
    - Added benefit: changes to value result in one fix
  - You can aslo use **#define** dircetive

#define NUMBER_OF_STUDENTS 24

- **#define Vs const ?**

Display 1.4 Named Constant

```
1   #include <iostream>
2   using namespace std;
3
4   int main( )
5   {
6       const double RATE = 6.9;
7       double deposit;

8       cout << "Enter the amount of your deposit $";
9       cin >> deposit;
10      double newBalance;
11      newBalance = deposit + deposit*(RATE/100);
12      cout << "In one year, that deposit will grow to\n"
13           << "$" << newBalance << " an amount worth waiting for.\n";

14      return 0;
15  }
```

SAMPLE DIALOGUE

Enter the amount of your deposit $**100**
In one year, that deposit will grow to
$106.9 an amount worth waiting for.

Arithmetic Operators

- Standard Arithmetic Operators
  - Precedence rules – standard rules

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm or $b \cdot m$	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Modulus	%	$r \bmod s$	`r % s`

Arithmetic Operators

- Standard Arithmetic Operators
  - Precedence rules – standard rules

Algebra: $z = pr\%q + w/x - y$

C++:

```
z = p * r % q + w / x - y;
```

6 1 2 4 3 5

Relational Operators

Comparing Expressions

Display 2.1 Comparison Operators

MATH SYMBOL	ENGLISH	C++ NOTATION	C++ SAMPLE	MATH EQUIVALENT
=	Equal to	==	`x + 7 == 2*y`	x + 7 = 2y
≠	Not equal to	!=	`ans != 'n'`	*ans* ≠ *'n'*
<	Less than	<	`count < m + 3`	*count* < m + 3
≤	Less than or equal to	<=	`time <= limit`	*time* ≤ *limit*
>	Greater than	>	`time > limit`	*time* > *limit*
≥	Greater than or equal to	>=	`age >= 21`	*age* ≥ 21

Boolean Expressions

- Recall that declaring a variable allocates memory for that variable's specific type.
- Arithmetic types include int and float.
- Symbolic types include and string.
- Another type is the Boolean (only one byte that is either true or false).

Boolean Expressions

Data Types:

`long double`	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits
`char`	1 byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
`bool`	1 byte	`true`, `false`	Not applicable

The values listed here are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system. *Precision* refers to the number of meaningful digits, including digits in front of the decimal point. The ranges for the types `float`, `double`, and `long double` are the ranges for positive numbers. Negative numbers have a similar range, but with a negative sign in front of each number.

true, false are predefined library constants

Boolean Expressions

Be careful!

- **In C++, 0 is false BUT any value other than 0 is considered true.**
- **For example:**

```
int x= 0; // Boolean false
int y = 1; // Boolean true
int z = 3; // Boolean true
sum= x+y; //Boolean true
y--; // Boolean false
```

```
z-- >=3  //true
6==x+7   //false
sum==x+y  //false
```

Logical Operators

Evaluating Boolean Expressions

- Logical Operators
  - Logical AND (&&)
  - Logical OR (||)

Display 2.2 **Truth Tables**

AND

Exp_1	*Exp_2*	*Exp_1* && *Exp_2*
true	true	true
true	false	false
false	true	false
false	false	false

OR

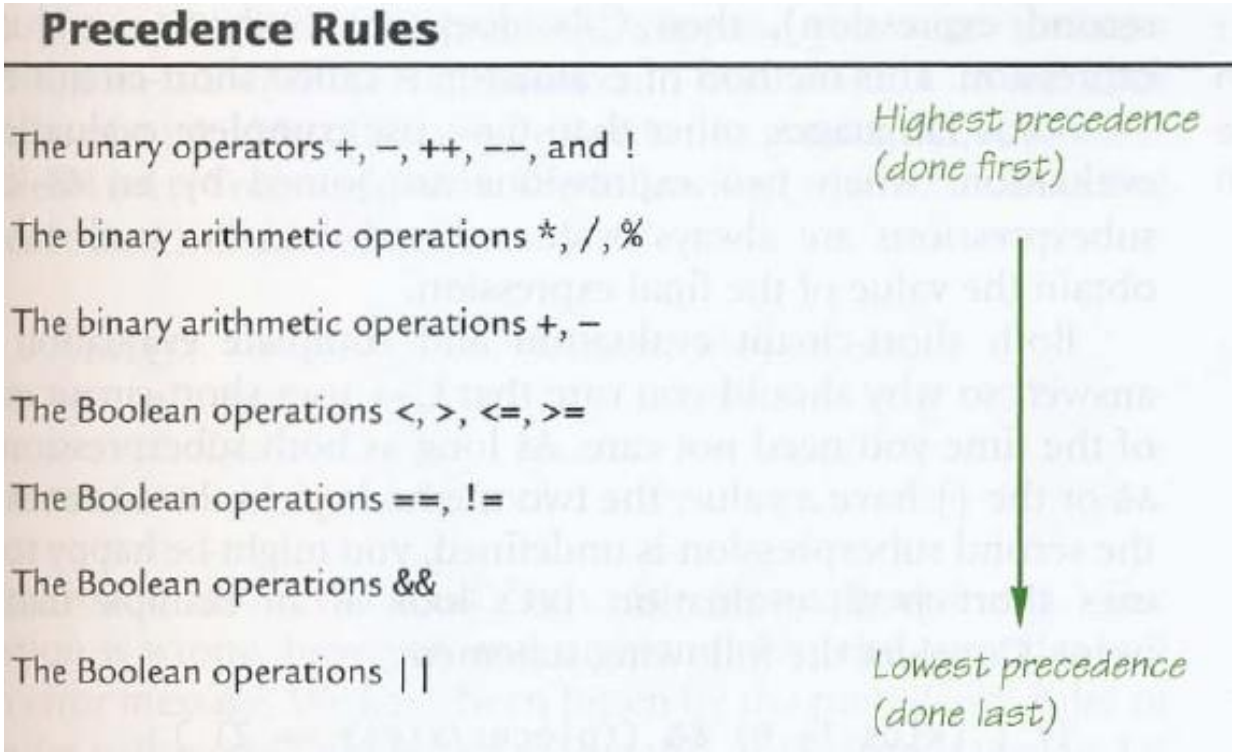
Exp_1	*Exp_2*	*Exp_1* || *Exp_2*
true	true	true
true	false	true
false	true	true
false	false	false

NOT

Exp	!(*Exp*)
true	false
false	true

Operators

Operators Precedence



Display 2.3
Precedence of Operators (1 of 4)

Display 2.3 Precedence of Operators

`::`	Scope resolution operator
`.`	Dot operator
`->`	Member selection
`[]`	Array indexing
`( )`	Function call
`++`	Postfix increment operator (placed after the variable)
`--`	Postfix decrement operator (placed after the variable)
`++`	Prefix increment operator (placed before the variable)
`--`	Prefix decrement operator (placed before the variable)
`!`	Not
`-`	Unary minus
`+`	Unary plus
`*`	Dereference
`&`	Address of
`new`	Create (allocate memory)
`delete`	Destroy (deallocate)
`delete[]`	Destroy array (deallocate)
`sizeof`	Size of object
`( )`	Type cast

Highest precedence (done first)

Display 2.3
Precedence of Operators (2 of 4)

*	Multiply
/	Divide
%	Remainder (modulo)
+	Addition
–	Subtraction
<<	Insertion operator (console output)
>>	Extraction operator (console input)

Lower precedence (done later)

Display 2.3
Precedence of Operators (3 of 4)

Display 2.3 Precedence of Operators

All operators in part 2 are of lower precedence than those in part 1.

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal
!=	Not equal
&&	And
||	Or

Display 2.3
Precedence of Operators (4 of 4)

=	Assignment
+=	Add and assign
−=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulo and assign
? :	Conditional operator
throw	Throw an exception
,	Comma operator

Lowest precedence (done last)

Console Input/output

- I/O objects cin, cout
- Defined in the C++ library called <iostream>
- Must have these lines (called pre-processor directives) near start of file:
  - #include <iostream>
    using namespace std;
  - Tells C++ to use appropriate library so we can use the I/O objects cin, cout, …

Console Input/output

- cin for input, cout for output
- Differences:
  - ">>" (extraction operator) points opposite
    - Think of it as "pointing toward where the data goes"
  - Object name "cin" used instead of "cout"
  - No literals allowed for cin
    - Must input "to a variable"
- cin >> num;
  - Waits on-screen for keyboard entry
  - Value entered at keyboard is "assigned" to num

Console Output

- What can be outputted?
  - Any data can be outputted to display screen
    - Variables
    - Constants
    - Literals
    - Expressions (which can include all of above)
  - cout << numberOfGames << " games played.";
    2 values are outputted:
    "value" of variable numberOfGames,
    literal string " games played."
- Cascading: multiple values in one cout

Console Output – Separating Lines

- New lines in output
  - Recall: "\n" is escape sequence for the char "newline"
- A second method: object endl
- Examples:

  cout << "Hello World\n";
  - Sends string "Hello World" to display, & escape sequence "\n", skipping to next line

  cout << "Hello World" << endl;
  - Same result as above

Console Output – Formatting Numbers

- Formatting numeric values for output
  - Values may not display as you'd expect!
    cout << "The price is $" << price << endl;
    - If price (declared double) has value 78.5, you might get:
      - The price is $78.500000 or:
      - The price is $78.5
- We must explicitly tell C++ how to output numbers in our programs!

Console Output – Formatting Numbers

- "Magic Formula" to force decimal sizes:
  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout.precision(2);
- These statements force all future cout'ed values:
  - To have exactly two digits after the decimal place
  - Example:
    cout << "The price is $" << price << endl;
    - Now results in the following:
      The price is $78.50
- Can modify precision "as you go" as well!

Console Output – Escape Sequences

- "Extend" character set
- Backslash, \ preceding a character
  - Instructs compiler: a special "escape character" is coming
  - Following character treated as "escape sequence char"

Display 1.3
Some Escape Sequences (1 of 2)

Display 1.3 Some Escape Sequences

SEQUENCE	MEANING
\n	New line
\r	Carriage return (Positions the cursor at the start of the current line. You are not likely to use this very much.)
\t	(Horizontal) Tab (Advances the cursor to the next tab stop.)
\a	Alert (Sounds the alert noise, typically a bell.)
\\	Backslash (Allows you to place a backslash in a quoted expression.)

Display 1.3
Some Escape Sequences (2 of 2)

\'	Single quote (Mostly used to place a single quote inside single quotes.)
\"	Double quote (Mostly used to place a double quote inside a quoted string.)
The following are not as commonly used, but we include them for completeness:	
\v	Vertical tab
\b	Backspace
\f	Form feed
\?	Question mark

Libraries

- C++ Standard Libraries
- #include <Library_Name>
  - Directive to "add" contents of library file to your program
  - Called "preprocessor directive"
    - Executes before compiler, and simply "copies" library file into your program file
- C++ has many libraries
  - Input/output, math, strings, etc.

Namespaces

- Namespaces defined:
  - Collection of name definitions
- For now: interested in namespace "std"
  - Has all standard library definitions we need
- Examples:
  #include <iostream>
  using namespace std;
    - Includes entire standard library of name definitions
- #include <iostream>

using std::cin;
using std::cout;

  - Can specify just the objects we want

Adding Style to your Program

- Make programs easy to read and modify
- Comments, two methods:
  - // Two slashes indicate entire line is to be ignored
  - /*Delimiters indicates everything between is ignored*/
  - Both methods commonly used
- Indentation (arranging and spacing your lines of code)
- Identifier naming
  - ALL_CAPS for constants: NUMBER_OF_STUDENTS
  - lowerToUpper for variables: numStudents
  - Most important: MEANINGFUL NAMES!: not x,y,x1,y1,ahmed…etc

Be Professional from day one!

4. Examples – (cont.)

- Take a number from user and display

Number	Squared	Predecessor	Successor
3	9	2	4

- Write a program to take two numbers and divide them putting the integer and remainder results in separate variables.

4. Examples – (cont.)

Given the following fragment that purports to convert from degrees Celsius to degrees Fahrenheit, answer the following questions:

```
double c = 20;
double f;
f = (9/5) * c + 32.0;
```

a. What value is assigned to `f`?

b. Explain what is actually happening, and what the programmer likely wanted.

c. Rewrite the code as the programmer intended.

4. Bonus Exercise ***

Solve and send on my email before Wednesday 14/2

7. An employee is paid at a rate of $16.78 per hour for the first 40 hours worked in a week. Any hours over that are paid at the overtime rate of one and one half times that. From the worker's gross pay, 6% is withheld for social security tax, 14% is withheld for federal income tax, 5% is withheld for state income tax, and $10 per week is withheld for union dues. If the worker has three or more dependents, then an additional $35 is withheld to cover the extra cost of health insurance beyond what the employer pays. Write a program that will read in the number of hours worked in a week and the number of dependents as input, and will then output the worker's gross pay, each withholding amount, and the net take-home pay for the week. For a harder version, write your program so that it allows the calculation to be repeated as often as the user wishes. If this is a class exercise, ask your instructor whether you should do this harder version.

4. Bonus Exercise **Sample Run**

Solve and send on my email before Wednesday 14/2

```
Hours worked per week: 30
Number of dependents: 0
**********Taxes**********
Gross pay: $503.40$
Social security tax: 30.20$
Federal income tax: 70.48$
State income tax: 25.17$
Union dues: 10.00$
Health Insurance <Optional>: 0.00$
**********Net Pay**********
Net pay: 367.55$
**********
Hours worked per week: 30
Number of dependents: 4
**********Taxes**********
Gross pay: $503.40$
Social security tax: 30.20$
Federal income tax: 70.48$
State income tax: 25.17$
Union dues: 10.00$
Health Insurance <Optional>: 35.00$
**********Net Pay**********
Net pay: 332.55$
**********
```

```
Hours worked per week: 50
Number of dependents: 0
**********Taxes**********
Gross pay: $922.90$
Social security tax: 55.37$
Federal income tax: 129.21$
State income tax: 46.15$
Union dues: 10.00$
Health Insurance <Optional>: 0.00$
**********Net Pay**********
Net pay: 682.18$
**********
Hours worked per week: 50
Number of dependents: 3
**********Taxes**********
Gross pay: $922.90$
Social security tax: 55.37$
Federal income tax: 129.21$
State income tax: 46.15$
Union dues: 10.00$
Health Insurance <Optional>: 35.00$
**********Net Pay**********
Net pay: 647.18$
**********
```

Thank You