Why call this course C/C++?
• C and C++ are both computer languages.
• They have the same fundamental structure and syntax.
• C++ has additional features that are not available in C.
• We cover the topics that are common to both languages. We also cover some topics that are in C++ but NOT in C.
• If a topic is not the same in both languages, we point out the differences.

Course Description
This course introduces the student to the C/C++ programming languages. In this course, students write programs which emphasize the concepts of structured programming, top-down design, and user interaction utilizing C and C++. Topics include functions, control statements such as loops and decisions, input/output, pointers, arrays, and strings. One computer per student is assigned for the course.

Prerequisite: ITEC 1002
3 Credit Hours, 3 Lecture Hours, 0 Lab Hours

Prerequisite: ITEC 1002
• If you have had ITEC 1002, you should know the concepts:
  – program design and program flow
  – input and output (both unformatted and formatted)
  – arithmetic calculations
  – decisions
  – loops
  – disk files
  – using a text editor and testing the program
  – syntax and logic errors

Prerequisite: ITEC 1002
• If you have not had ITEC 1002 but you have had experience programming, make sure that you know the concepts listed previously.
• Be aware that many of the examples given during the lectures will use QBASIC for comparison to emphasize the similarity of programming concepts, syntax, and styles.
Primary Course Goals

- To familiarize the student with the fundamental topics of the C and C++ programming languages
  - An introduction to both languages

- To prepare the student for future study of the C and C++ programming languages
  - Additional courses provide depth and breadth of both C and C++.

Problem-solving as programmers

- As with all programming, you will continue to develop skill in problem solving.

To attain the course goals the student will:

- flowchart, code, run and test C/C++ programs
- know the acceptable C/C++ data types
- be able to perform I/O
- be able to use arithmetic operators and write assignment statements
- understand C/C++ looping mechanisms
- understand C/C++ decision structures
- be able to use file I/O
- be able to write functions and use pointers
- be able to manipulate arrays and strings

NO LAB WILL BE ACCEPTED LATE

THE SCHEDULE IS TENTATIVE.
An Introduction to C/C++ Programming

History of C
• developed in 1972 by Dennis Richie of Bell Labs
  – old by comparison to some languages
  – new by comparison to some languages

History of C
• developed in 1972 by Dennis Richie of Bell Labs
• developed for use in designing the UNIX operating system
  – must be very powerful if you are to write an operating system
  – UNIX was predecessor of LINUX

History of C
• chief goal was to be a useful language for working programmers
  – for the use of professionals
  – unlike some other programming languages that were intended to make programming available to all:
    • COBOL – for business and accounting users
    • Common Business Oriented Language
    • Fortran – for engineers and scientists
    • Formula Translator
    • Basic – for students
    • Beginners All-purpose Symbolic Instruction Code
History of C

• Chief goal was to be a useful language for working programmers
  – A double-edged sword
    • Gives you lots of power
    • Requires you to know what you're doing

History of C++

• C++ is a superset of C
• C is a subset of C++
  – Can be viewed either way
  – What is in C is still available in C++
  – C++ contains features that are not available in C

History of C++

• C++ is a superset of C
• C is a subset of C++
• Developed in 1983-85 by Bjarne Stroustrup of Bell Labs
  – C was already a well-established language at the time C++ was created

History of C++

• C++ is a superset of C
• C is a subset of C++
• Developed in 1983-85 by Bjarne Stroustrup of Bell Labs
  – Originally called "C with Classes"
  – Classes won't be discussed in detail until a later course
  – You will use objects of classes in this course in certain cases

History of C++

• C++ is a superset of C
• C is a subset of C++
• Developed in 1983-85 by Bjarne Stroustrup of Bell Labs
  – Originally called "C with Classes"
  • Object-oriented features a new way of thinking
    – Won't be covered until a later course
  • Still supports procedural programming
    – Like you did in ITEC 1002
Why C/C++?

- modular design
  - structured programming
  - top-down design
  - look at the big picture first... then divide into parts
  - facilitates project teams
    - individuals work on a piece (module) of the project
    - modules are combined into the completed project

- features of both high-level and low-level programming languages
  - can do work at the bit level (low-level)
  - can work on complex data structures (high-level)

- compact size
  - few syntax rules
    - both good and bad
      - at times, you’ll wish you had gotten a syntax error
    - few keywords (page 43)
      - compare to the full page of reserved words in QBasic
    - produces fast & efficient code

- portable
  - can be ported among different computers easily

- powerful external library of functions
  - much of the functionality of programs is done by code contained in libraries which are simply included by the programmer
  - for example, input and output is accomplished by code in libraries
    - there is no PRINT statement
    - there is no INPUT statement
    - input and output are done by function calls
      » you’ll hear lots more about function calls later

- in QBasic, you primarily dealt with two types of data
  - numeric
  - string

- Although you won’t be working with all of the types in C and C++, there are a variety of storage types:
  - int, char, float, double, bool
  - long, short, signed, unsigned, long double
  - pointer, register, constant, volatile
  - local, global, auto, static, external
  - composite types: arrays, strings, structures, unions, ADTs (abstract data types), user-defined types
Why C/C++?

• in QBASIC, you used
  +  -  *  /  ^
  >  >=  <  <=  ==

• lots of operators:
  +  -  *  /  %  ++  --
  =  +=  -=  /=  %=
  ==  >=  <  <=  !=
  &&  ||  !
  >>  <<
  ?:  .  ->

Why C/C++?

• basic control sequences
  – sequence
  • you know this from ITEC 1002
  – if then else
  • you know this from ITEC 1002
  – loops (3 types)
  • you know some of these from ITEC 1002
  • you had some available in ITEC 1002 that are not available here
  – functions

Comparison of C/C++ and QBASIC

QBASIC

LET sum = sum + grade

or

sum = sum + grade

because the LET was optional

C/C++

sum = sum + grade;

or

sum += grade;

which is a shortcut

Notice the use of new operators:

ctr = ctr + 1;

or

ctr += 1;

Think about C++ as added on to C.
### Comparison of C/C++ and QBASIC

**QBASIC**

```qbasic
IF ctr = 50 THEN
  ...
END IF
```

**C/C++**

```c/c++
if (ctr == 50)
{
  ...
}
```

### Comparison of C/C++ and QBASIC

**QBASIC**

```qbasic
IF hours >= 12 THEN
  tuition = 625
  status$ = "F"
ELSE
  tuition = 50 * hours
  status$ = "P"
END IF
```

**C/C++**

```c/c++
if (hours >= 12)
{
  tuition = 625;
  status = 'F';
}
else
{
  tuition = 50 * hours;
  status = 'P';
}
```

### Comparison of C/C++ and QBASIC

**QBASIC**

```qbasic
ctr = 0
DO WHILE ctr < 10
  clr = clr + 1
  PRINT clr
LOOP
```

**C/C++**

```c/c++
ctr = 0;
while (ctr < 10)
{
  clr = clr + 1;
  cout << clr;
}
```
Steps in C/C++ programming process
1. Analyze the purpose of the program
2. Visualize the running program; sketch the user interface
3. Model the program using design tools (hierarchy chart, flowchart, pseudocode)
4. Check the model for errors (desk checking)
5. Enter the program into the computer and compile it
6. Correct errors and recompile until there are none
7. Execute (run) the program
8. Correct execution errors, recompile, rerun
9. Validate results: does the output meet what was defined in step 1?

Chapter 1
Introduction to Computers and Programming

Why Program?
Computer – programmable machine designed to follow instructions
Program – instructions in computer memory to make it do something
Programmer – person who writes instructions (programs) to make computer perform a task
SO, without programmers, no programs; without programs, a computer cannot do anything

Main Hardware Component Categories:
1. Central Processing Unit (CPU)
2. Main Memory
3. Secondary Memory / Storage
4. Input Devices
5. Output Devices

Central Processing Unit (CPU)
Comprised of:
Control Unit
Retrieves and decodes program instructions
Coordinates activities of all other parts of computer
Arithmetic & Logic Unit
high-speed numeric calculation
true/false, yes/no decisions

Main Hardware Component Categories

Figure 1-1
Main Memory
- It is volatile - erased when program terminates or computer is turned off
- Also called Random Access Memory (RAM)
- Organized as follows:
  - bit: smallest piece of memory. Has values 0 (off, false) or 1 (on, true)
  - byte: 8 consecutive bits. Bytes have addresses.
- Addresses – Each byte in memory is identified by a unique number known as an address.

In Figure 1-3, the number 149 is stored in the byte with the address 16, and the number 72 is stored at address 23.

Secondary Storage
- Non-volatile: data retained when program is not running or computer is turned off
- Comes in a variety of media:
  - magnetic: floppy disk, zip disk, hard drive
  - optical: CD-ROM
  - Flash drives, connected to the USB port

Input Devices
- Devices that send information to the computer from outside
- Many devices can provide input:
  - Keyboard, mouse, scanner, digital camera, microphone
  - Disk drives and CD-ROM

Output Devices
- Output is information sent from a computer program to the outside world.
- The output is sent to an output device
- Many devices can be used for output:
  - Computer monitor and printer
  - Floppy, zip disk drives
  - Writable CD drives

Software – Programs That Run on a Computer
- Categories of software:
  - Operating system: programs that manage the computer hardware and the programs that run on them. Examples: Windows, UNIX, Linux
  - Application software: programs that provide services to the user. Examples: word processing, games, programs to solve specific problems
Programs and Programming Languages

• A program is a set of instructions that the computer follows to perform a task.

• We start with an algorithm, which is a set of well-defined steps.

Example Algorithm for Calculating Gross Pay

1. Display a message on the screen asking “How many hours did you work?”
2. Wait for the user to enter the number of hours worked. Once the user enters a number, store it in memory.
3. Display a message on the screen asking “How much do you get paid per hour?”
4. Wait for the user to enter an hourly pay rate. Once the user enters a number, store it in memory.
5. Multiply the number of hours by the amount paid per hour, and store the result in memory.
6. Display a message on the screen that tells the amount of money earned. The message must include the result of the calculation performed in Step 5.

Machine Language

• Although the previous algorithm defines the steps for calculating the gross pay, it is not ready to be executed on the computer.
• The computer only executes machine language instructions.
• Machine language instructions are binary numbers, such as 1011010000000101.
• Rather than writing programs in machine language, programmers use programming languages.

Programs and Programming Languages

• Types of languages:
  – Low-level: used for communication with computer hardware directly. Often written in binary machine code (0’s/1’s) directly.
  – High-level: closer to human language

Some Well-Known Programming Languages

<table>
<thead>
<tr>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASIC</td>
<td>Beginners All purpose Symbolic Instruction Code. A general programming language originally designed to be simple enough for beginners to learn.</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>Formula Translators. A language designed for programming complex numerical algorithms.</td>
</tr>
<tr>
<td>COBOL</td>
<td>Commercial Business-Oriented Language. A language designed for business applications.</td>
</tr>
<tr>
<td>Pascal</td>
<td>A structured, general-purpose language designed primarily for teaching programming.</td>
</tr>
<tr>
<td>C</td>
<td>A high-level, general-purpose language developed at Bell Laboratories. Often used in software development.</td>
</tr>
<tr>
<td>C++</td>
<td>An object-oriented language derived from C.</td>
</tr>
<tr>
<td>Java</td>
<td>An object-oriented language designed for software development.</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>A Microsoft programming language and software development environment that allows programmers to quickly create Windows-based applications.</td>
</tr>
</tbody>
</table>

From a High-level Program to an Executable File

a) Create file containing the program with a text editor.
b) Run preprocessor to convert source file directives to source code program statements.
c) Run compiler to convert source program into machine instructions.
d) Run linker to connect hardware-specific code to machine instructions, producing an executable file.
• Steps b–d are often performed by a single command or button click.
• Errors detected at any step will prevent execution of following steps.
From a High-Level Program to an Executable File

Integrated Development Environments (IDEs)
- An integrated development environment, or IDE, combine all the tools needed to write, compile, and debug a program into a single software application.
- Examples are Microsoft Visual C++, Borland C++ Builder, CodeWarrior, etc.

Integrated Development Environments (IDEs)

What Is a Program Made Of?
- Common elements in programming languages:
  - Key Words
  - Programmer-Defined Identifiers
  - Operators
  - Punctuation
  - Syntax

Program 1-1
```
1 // This program calculates the user's pay.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7    double hours, rate, pay;
8    // Get the number of hours worked.
9    cout << "How many hours did you work? ";
10   cin >> hours;
11   // Get the hourly pay rate.
12   cout << "How much do you get paid per hour? ";
13   cin >> rate;
14   // Calculate the pay.
15   pay = hours * rate;
16   // Display the pay.
17   cout << "You have earned $" << pay << endl;
18   return 0;
19 }
```

Program 1-1
```
1 // This program calculates the user's pay.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7    double hours, rate, pay;
8    // Get the number of hours worked.
9    cout << "How many hours did you work? ";
10   cin >> hours;
11   // Get the hourly pay rate.
12   cout << "How much do you get paid per hour? ";
13   cin >> rate;
14   // Calculate the pay.
15   pay = hours * rate;
16   // Display the pay.
17   cout << "You have earned $" << pay << endl;
18   return 0;
19 }
```
```cpp
// This program calculates the user's pay.
#include <iostream>
using namespace std;

int main()
{
    double hours, rate, pay;
    cout << "How many hours did you work? ";
    cin >> hours;
    cout << "How much do you get paid per hour? ";
    cin >> rate;
    pay = hours * rate;
    cout << "You have earned $" << pay << endl;
    return 0;
}
```

**Key Words**

- Also known as reserved words
- Have a special meaning in C++
- Can not be used for any other purpose

**Programmer-Defined Identifiers**

- Names made up by the programmer
- Not part of the C++ language
- Used to represent various things: variables (memory locations), functions, etc.
Operators

- Used to perform operations on data
- Many types of operators:
  - Arithmetic - ex: +, -, *, /
  - Assignment – ex: =

Operators

1 // This program calculates the user's pay.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     double hours, rate, pay;
8     // Get the number of hours worked.
9     cout << "How many hours did you work? ";
10    cin >> hours;
11
12     // Get the hourly pay rate.
13     cout << "How much do you get paid per hour? ";
14     cin >> rate;
15
16     // Calculate the pay.
17     pay = hours * rate;

Punctuation

- Characters that mark the end of a statement, or that separate items in a list

Punctuation

1 // This program calculates the user's pay.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     double hours, rate, pay;
8     // Get the number of hours worked.
9     cout << "How many hours did you work? 
10    cin >> hours;
11
12     // Get the hourly pay rate.
13     cout << "How much do you get paid per hour? 
14     cin >> rate;
15
16     // Calculate the pay.
17     pay = hours * rate;

Syntax

- The rules of grammar that must be followed when writing a program
- Controls the use of key words, operators, programmer-defined symbols, and punctuation

Variables

- A variable is a named storage location in the computer's memory for holding a piece of data.
Variable Definitions

• To create a variable in a program you must write a variable definition (also called a variable declaration)
• The variable definition specifies the
  – type of data a variable can hold
  – the variable name
• double hours, rate, pay;

Input, Processing, and Output

Three steps that a program typically performs:
1) Gather input data:
   • from keyboard
   • from files on disk drives
2) Process the input data
3) Display the results as output:
   • send it to the screen
   • write to a file