This instruction relies on the C language described in

C++:
The Complete Reference
Third Edition
By Herbert Schildt
Background

- The C language was developed at Bell Labs by Dennis Richie and documented by Brian Kernighan and Dennis Ritchie.
- The initial version of C was defined for the Unix operating system.
- Unix was originally developed for the HP PDP-8 computer in assembly language. Porting Unix to the HP PDP-11 required significant rewrite. C was developed to rewrite Unix in a higher level language than Assembly to make it more portable.
- C is considered a middle level language; it is higher level than Assembly which consists of machine code mnemonics and high level languages (like FORTRAN and COBOL) that fully abstract the computer.
Background - C is a compiler language

- Computer languages provide instructions to the Central Processing Unit (CPU) to perform operations on data or peripheral devices.
- The lowest level language is Assembly: Assembly uses mnemonics (text strings) that represent machine code instructions used by the CPU.
- Higher level languages (like Middle level C or High level FORTRAN) provide English-like strings that provide instructions.
- Higher level languages have to be translated to lower level languages like Assemble of machine code: the process that translates higher level languages is called a compiler.
Compiler languages, like natural languages (like English) have three essential elements:

- **Syntax** – Syntax is the grammar that specifies how instructions are produced.
- **Semantics** – Semantics is the meanings applied to the instructions.
- **Lexicon** – Lexicon, or vocabulary, is the definition of strings or symbols used to construct instructions.

Instructions can be statements or decisions (conditional statements).

Instructions have context based on the Lexicon and Semantics used to define them.
Background – Computer Architecture

- C Language is dependent on the compiler implementation, the compiler implementation is dependent on the computer and its architecture and operating system.

- The Technician training course uses IBM compatible personal computers (PCs) with the Windows (New Technology – NT) operating system. The development tool used is Visual Studio and the Visual C compiler. This training is based on these constraints.
The IBM PC uses the Intel or AMD CPUs. These CPUs are referred to as the X86 or X86-64 architecture. The X86 is a 32-bit CPU and the X86-64 (or X64) is a 64-bit CPU. The 32 and 64 designation indicates how many bits maximum are used internally in the CPU (this is also referred to as the data Word size).

The CPU can reference memory as real memory (in real mode) or virtual memory (in protected mode).

The 32-bit CPU can access 3.5 GB (3.5 Billion bytes) of memory, while the 64-bit CPU can access 192 GB (16 GB on the Home version).

Applications on the 64-bit Windows can reference up to 8 TB (TeraBytes) of virtual memory.
Background – Computer Memory Architecture

- Left image shows a notional layout of CPU memory for an application.
- Memory layout represents a virtual memory space. All running applications (indicated as Process A) has the same memory mapping.
- Applications (Processes) are allocated actual memory based on need; some memory may be paged (as fixed amount of memory) in real memory or a disk file called the pagefile.
- The Kernel-Mode space is not the actual operating system kernel; it is an access interface layer the application uses to call (connect) to the kernel.
Background – Application Memory Architecture

- The X86/X64 CPU provides four memory segments. These segments have access protections assigned to them.
- The Code Segment is read and execute. This is where the program executable instructions and static unchangeable data is allocated.
- The Data and Extended Segments are read and write enabled, but cannot be executed. The Data and Extended segment address register starts at the low address and increments as used.
- The Stack Segment is read and write enabled, but cannot be executed. The stack segment address register starts at the high address and decrements as used.
Brief Introduction to the C Language
C Language Basics

- C Language has the following elements
  - Precompile directives
  - Data declarations
  - Code instructions

- Precompile directives changes the source before compilation. These directives can direct source information to be added to the program from other files, give descriptive names to data constants, define macro (give names for a set of instructions that are used more than once, or control what source is included in a compilation.
C Language Basics

- Data declarations either define data or reference data
  - Data that is specified such that memory is allocated is referred to as a data definition declaration; data that does not allocate memory is called a data reference declaration.
  - Data declarations have scope: that is, the data is recognized by the compiler when described within the grouping of source being compiled. Data can have global scope (that is data defined external to any group of source instructions: data defined outside of a group of instructions called a “function” is global data and can be used by all instructions. Data can have local scope: data defined within a “function” has function local scope and can only be used within the function.
C Language Basics

- Code instructions
  - Code instruction tell the computer what to do.
  - Code instructions use data
  - Code instructions are grouped into units called functions
  - Code instructions are read and execute only and can not be changed during execution.
  - Code instructions can be simple or complex statements.
  - Code instructions can be decision groups.
C Language Compiler

- C compiler uses the following reserved words: auto, double, int, struct, break, else, long, switch, case, enum, register, typedef, char, extern, return, union, const, float, short, unsigned, continue, for, signed, void, default, goto, sizeof, volatile, do, if, static, and while.

- These reserved words constitute the C language Lexicon and can only be used as described in the C language specification.

- Symbols are used to define operations; those symbols include =, +, -, *, /, %, ?::, ;; <, >, ~, #, (,), {, }. 
A C program consists of one or more source files.

- A source file can be a header file designated by a file extension of “.h”.
- A source file can be a code file with data and instructions designated with an extension of “.c” or “.cpp”.
- Header files generally have data declarations; although instructions may also be included in the file.

The C language instruction grammar consists of statements terminated with a semi-colon (;) or groups of statements indicated by an opening left brace ({}] and ended (completed) by a closing right brace ({}).
C Language Sample Program

1. // HelloWorld.cpp : Defines the entry point for the console application.
2. //
3. 4. #include "stdafx.h"
5. #include <stdlib.h>
6. #define LOOP_LIMIT 3
7. 8. char Phrase1[] = "said";
9. char Phrase2[] = "said";
10. char Phrase3[] = "will say";
11. char *Phrases[3] = { Phrase1, Phrase2, Phrase3 };
12. 13. int main()
14. {
15.    int I1;
16. 17.    char *DayRef[] = { "Yesterday", "Today", "Tomorrow" };
18. 19.    for (int I2 = 0; I2 < LOOP_LIMIT; ++I2)
20.    {
21.        I1 = I2;
22.        printf("I %s %s: Hello World\n", Phrases[I1], DayRef[I2]);
23.    }
24.
25.    printf("The final index is %d\n", I1);
26.    //printf("The final index is %d\n", I2);
27.    system("Pause");
28.    return 0;
29. }
Explanation of Sample HelloWorld

- Lines 1, 2, and 26 are comments. Comments are not compiled and are valuable means of documenting the program. There are two types of comments: the sample demonstrates the use of a single line comment. Multi-line comments are enclosed in “/* */” designators.

- Lines 4 and 5 are precompile directives telling the compiler to locate the identified files and replace the include directive with the contents of the file. There are two forms of the include directive, as shown. When the file name is enclosed in double quotes, the file is contained in the source directory or the absolute drive-directory-file name is given. If the include file name in enclosed in less-than and greater than symbols (<>), the compiler searches the of include directories to locate the file.
Explanation of Sample HelloWorld

- Line 6 is a precompile definition directive. It associates the word `LOOP_LIMIT` with the value 3. Wherever the word `LOOP_LIMIT` is encountered it is replaced with a 3.

- Lines 8, 9, and 10 are global data definition declarations of 8-bit character data. The example shows a special type of array of character type data: this is an example of a Null or Zero terminated character string array. The character array must be at least big enough to include all the characters in the double quotes plus the terminating zero.

- Line 11 is an example of an array of memory addresses. The memory address (called a pointer) of each character string array in contained in it.
Line 13 is a function definition declaration. The function definition specifies that the function returns an integer when it exits, has the name “main”, and has no input parameters.

Line 14 starts a group of instruction associated with the “main” function and line 31 ends the group of instructions for “main”.

Line 15 defines a “main” function local data definition of an integer called I1.

Line 17 defines a local array of character strings arrays and presets the memory addresses to the strings “Yesterday”, “Today” and “Tomorrow”.
Line 19 is a loop instruction group declaration. The loop definition specifies that an integer I2 is defined and initialized with the starting value of zero. The loop definition sets a termination condition when I2 is greater than or equal to LOOP_LIMIT (the value 3). Finally, I2 is incremented at the end of the loop and the loop continues by checking the termination condition.

Line 20 starts a group of instruction associated with the loop group and line 23 ends the group of instructions for the loop.

Line 21 sets (assigns) the integer I1 to the value contained in I2.

Lines 22 and 25 call (by name reference) a Windows service routine to print a string to the console. The compiler knows the characteristics associated with the function because the function reference is defined in one of the header files “include”d in the program.
Explanation of Sample HelloWorld

- Line 28 is a function call to another Windows service routine. This function takes as an input parameter a string containing a Windows command.
- Line 30 exits the “main” function and specifies zero as the return value.
C Language Detailed Descriptions

• The C Language consists of compiler Data declarations, Code statements, and pre-compilation directives.

• The language also includes features to compile independent parts of a complete program separately and bind them together.

• The language provides mechanism to bind object groups called libraries containing one or more source files with one or more source functions.

• The developers of the C Language compilers provide libraries to interact with a target operating system services.
Data Declaration Details

• There are 5 basic data types in C: char (8 bit character), int (word size integer), float (real numbers containing fractional values), double (values containing twice the number of bits as floating point), and void.

• The “void” type has limited use that will be explained later.

• The “char” type is meant for ANSI ASCII encoded data, but has been used for any 8-bit data.

• The “float” and “double” (or double precision floating point data) is based on a IEEE standard: hardware (CPU) and compiler developers are expected to follow and meet this standard.
Data Declaration Details (continued)

• The “int” data type is dependent on two factors: the internal data size supported by the CPU hardware and the integer size provided by the operating system as supported by the compiler.

• The integer data size is referred to as a “word”. The INTEL and AMD processors (starting with the 80386 through the Pentium family) were 32-bit word processor (internal data handling storage entities are called general data registers). The early DOS and Windows operating systems were 16-bit word based; starting with Windows 95 and Windows NT the word size changed to 32-bit (i.e., Win32).

• Today’s CPUs support 64-bit word sizes; however, many Windows applications are still based on the Win32 word size.

• The “int” type can also be declared as a short int (generally 16-bits) or short, or long int (generally 64-bits) or long.
Data Declaration Details (continued)

• The “char” and “int” data types are signed values by default. That means the range of values for the char type is -127 to 127.
• The sign can be ignored by declaring these data types as unsigned:
  Unsigned char    or    unsigned int.
• One could define “signed char” or “signed int”; but this is unnecessary since they are signed by default (but it is syntactically and semantically correct).
Data Declaration Details (continued)

• There are several other data type declarations depending on the compiler and its ISO compliance level.

• Other data types include “enum”, “typedef”, or “register”.

• Data declarations can be grouped together to form a larger data unit. This is accomplished by declaring a “struct” (a structure) or “union”. In the case of a struct, all declared data elements contribute to the memory footprint (amount of memory needed for a data definition) for the data. In the case of a union, the data element that allocates the largest memory footprint determines the size of the union.
Data Declaration Details (continued)

• Data declarations can be definitions; i.e., the declaration allocates memory space: 
  `int data_value;`

• Data declarations can be references; i.e., the declaration specifies data characteristics but does not allocate any memory. The data modifier “extern” is used to declare a data reference:

  `extern int data_value;`
Data Declaration Details (continued)

• There are two more data modifiers of interest at this time: const and volatile.

• The const, or constant, modifier is used to define data that is not modifiable. The value is assigned at compile (or link) time and does not change during execution of the code.

• The volatile modifier is used to direct the compiler to always get (fetch) the data before using. This modifier is used extensively in multi-threaded applications where more than one thread uses the data. Typically, compilers attempt to optimize code by putting data in a register and reusing that register anytime the data is used. This modifier stops that optimizing behavior.
Data Declaration Details (continued)

• The C Language provides another method for accessing data defined elsewhere without using a reference declaration (extern).

• C allows data to be accessed using memory addresses. Every variable (that defines and allocates memory) can be accesses using the memory address where the data is allocated.

• Data that uses an address reference is declared the same (or similar) to the data to be accessed with an asterisk (*) preceding the variable name:

```c
int data_value;
int *data_value_address;
```

The first is a definition for an integer; the second is a definition of a variable to hold the address of an integer.

• Variables defined to hold the address of a data entity is called a pointer type variable, and the memory address storage in the variable is called a pointer to a type of data.

• Pointer variables can be defined as void types: void *data_entity; this is one case where “void” can be used to define data (the size of an address is known).
Data Declaration Details (continued)

• C provides the capability to “cast” data from one type to another.
• The compiler will automatically apply casting rules when data is cast from one type to another:
  ```c
  int Int_Data;
  char Char_Data;
  Int_Data = Char_Data;  // (the char data is changed to an integer
                           // and sign filled)
  Char_Data = Int_Data;  // (the integer data is truncated to 8 bits and
                          // saved in the char type. The compiler
                          // typically issues a warning that data maybe lost)
  ```
• The developer can force casting when the compiler default behavior might produce incorrect results.
  ```c
  char Char_value1; unsigned int Int_value2;
  Int_value2 = (unsigned char) Char_value1;  // (forces Char_value1 to be treated as
                                             // unsigned data to prevent any warning messages or sign extension)
  ```
Arrays

• An array is any data declaration that contains more than one instance of the data. Arrays can be any of the four basic types that have memory size (char, int, float, double) or structure (struct or union);

• Values in an array are accessed by using an index or item number. Arrays are zero based; i.e., the first item in the array is index or item zero (0).

• For example, int val[10]; declares an integer array of 10 items. Note the use of the opening and closing brackets ([ ]) to assign the number of items. val[0] = 5; assigns the value of 5 to the first item; val[6] = 12; assigns the value 12 to the seventh item.

• Arrays can have a single dimension (int val[10]) or multiple dimensions (int valset[10][5]).
Arrays (Continued)

• C recognizes a special type of character array called a zero terminated string (or C string). The declaration `char cdat[10]` defines an array of 10 characters. The declaration `char sdat[] = "A string";` will define an array of 9 characters (8 characters for the constant data “A string” and a zero ninth character). Be sure to allocate one more item in the character array for the zero.

• The C compiler is installed with helper routines; some of the helper routines support zero terminated character arrays.
Pointers and References

• A pointer is a variable declared as one of the previously described types that contains the address to that type of data. For example, int *val; defines an integer type of data (somewhere in memory) that val contains the address for (pointer to) it.

• A pointer declaration contains an asterisk character preceding the name of the variable.

• A reference is the address of a data entity. int val; int *pval = &val; will assign the address (as a reference) of val to the pointer variable pval;
Naming Conventions

• A quick word about naming data and code: this is up to the developer or team of developers.

• There are several different approaches to naming entities in a set of code. One good rule of thumb: Be consistent, use meaningful names and use comments to describe the intent of the entity.

• There are two types of comments: one line comments distinguished by double forward slashes (//) and multi-line comments distinguished by a forward slash and asterisk (/*), at least one white space (blank, new line, tab) and an asterisk and forward slash (*/).

• One line comments can be on a line by itself, or can be added after some code. Anything on the line after the two forward slashes is a comment and is ignored by the compiler.

• Multi-line comments can be used to add comment blocks to an application to describe a set of data or code statements, or can be used to comment out a set of code that is not desired (but which may be wanted at a later time).
The C language has 32 reserved words to define program elements; of those, 13 are code related. The code list consists of break, else, switch, case, return, continue, for, default, goto, sizeof, do, if, and while.

C programs consist of one of more code grouping entities call functions. Functions are designated by a return type (which can be a type of void), a function name (that is not a reserved word or data name), an opening parenthesis (()), one or more parameter declarations comma separated, a closing parenthesis ()), an opening brace ({}) and a closing brace (}). The functions code statements are entered between the opening and closing braces.

A function declaration that ends in a semicolon (;) and does not have opening and closing braces is called a function prototype.
Code Detailed Descriptions

• A program must contain at least one function and that function must be named “main”.

• Other functions can appear before or after “main” or may appear in a separate file.

• Any function that appears after “main”, or is in another file, must be declared by a function prototype.

• Files that contain other functions can be “included” (using the #include precompile directive) or compiled separately and included at link time.
Code Detailed Descriptions

• Functions can define data entities; this data is referred to as function local data and has local scope.

• Functions contain statements. Most statements end with the semicolon (;).

• Data definitions, assignment statements, function calls and single statement conditionals end with the semicolon.

• Conditional statements that contain a statement group, starts and ends the group with the opening and closing braces ({}); this group can be thought of as a conditional function.
Code Detailed Descriptions

• Assignment statement use a single equal sign (=) to indicate an assignment.
• The left side variable must be associated with a memory location and cannot
  be a constant (defined as const).
• The right side can be a variable, a constant (#define or const), an equation or
  function reference that returns a value.
• An equation is any set of variables and operators that result in a value. The
  operators can be add (+), subtract (-), multiple (*), divide (/), modulo or
  remainder (%) or conditional.
• A common conditional used in assignment statements is as follows:
  (conditional test ? Value returned on true : value returned on false)

  char result = (status < 0 ? 0 : 1); // zero indicates failure, one means success
There are several types of conditional statements.

The “if” statement is a simple test. The value being tested can be a variable compared to a constant (==, !=), a variable compared to a range or series of constant values (<,>,<=,>=), or a return value from a function call. Any test result that produces a zero status is considered a false condition; any result that produces a nonzero status is considered a true condition.

An if statement that has a true status can contain a single operation like an assignment or function call and ends with a semicolon.

An if statement that has a true status can contain a group identified by an opening and closing brace.

The if conditional can have an alternate statement or group of statements; the alternate set is specified by the “else” reserved word.

```plaintext
if (a < b) a = b;
if (a == c) {
    a = b;
}
if (a < b && a != c)    // the double ampersand (&&) means the logical And of the two conditions
{
    a = c;
}
else
{
    a = b;
}
```
• There are three loop statements that contain conditionals
• The “for” statement defines a starting value for the loop, the end test condition, and the loop change value (which is optional).
• Basic form of the “for” statement:
  
  ```
  for (index = 0;  index < 10; index = index + 1) {}
  ```

  The for statement sets index to zero to start, will loop until index is equal to or great than 10, and if the loop is not at the end will execute the loop statements, and finally add one to index and test it.
• The “for” loop can define a for loop local variable:
  
  ```
  for (int indx = 0; indx < 10; indx = indx + 1) {}
  ```
• Note:

Any time a variable is increased by one, a pre-evaluation increment or post-evaluation operation can be used.

“++index” indicates that index will be incremented by one unit before it is used.

“index++” indicates that index will be incremented by one unit after it is used.

If index is to be increment by more than one there is a shorthand method that can be used:

```cpp
index = index + 3; // skip two items and move to the third
index += 3;        // same as the previous statement
```
Code Detailed Descriptions

• The “while” and “do” “while” loop conditionals define the loop exit conditions. The initial starting value (state) must be set before the loop starts and the exit value (state) must be set inside the loop statement group.

• The “while” conditional is called a top test loop; the test condition is evaluated before the loop starts.

• The “do” “while” conditional is called a bottom test loop; the test condition is evaluated after to loop group is executed. This loop type insures that the group of statement is executed at least once.

```c
while (indx < EndLoop) {}     do {} while (indx < EndLoop);
```
• If there is a set of values that have to be evaluated, there is conditional statement to handle that case. The “switch” statement tests the value of a variable, and the “case” statements identify the values that are tested. A “case” statement can only identify one value. If there is a possibility that none of the identified values (as indicated in “case” statements) will be found and there is a need to handle that case, the “default” statement can be included in the conditional.

```c
switch (error_condition)  // must evaluate to one of the basic types
{
    case err1:  statement1; statement2;
    case err2:
    case err3:  statement3;  break;
    default:  statement4; break;
}
```

• The example introduced the word “break”. The first case will execute statement1, statement2 and statement3; if there is no operation or action to stop the execution, the first case will “fall” through to the second and third cases. The second case performs the same processing as the third case, and falls through to the third case. The third case executes statement3, and then executes the break operation. The break operation exits the current group. If the group is and if or an else, processing leaves the if conditional without executing any more statements within the group. If the group is a loop (for, while or do while), the loop is exited without any further processing.
Visual Studio Basics
Visual Studio Introduction

• This section describes Windows Visual Studio Community Version 15
• Each version of Visual Studio has slightly different features and capabilities.
• The compilers (C, Basic, C#), linker and support libraries may not be compatible with code and project files developed using other versions.
Visual Studio Start Page

The Initial Start Page is configurable: this example shows the typical default configuration.

On the left are the options for starting a project. At the top are the option to create a new project, open an existing project, or, if Visual Studio is connected to a source control application like Source Safe, a project can be started from source control.

The left area also contains a list of known previous projects.

The far right and bottom windows are useful only after a project is open and active.

The middle area is informational with links to useful topics.

The next slides occur when New Project is selected.
The New Project Dialog allows the developer to choose which language this project will be based (selected on the left). Note, under C++ there are several different types of C++ projects. For this example, and for the class projects, the Win32 Console Application project is selected.

At the bottom of the dialog are three entries: the project name, the directory/file area to save the project files, and the work or solution name of the application. The solution name defaults to the project name (it is best to leave the default name).

The create directory for solution is checked; this will create all the main project directory and subdirectories. This option will warn the user if the named solution directory already exists; this helps to prevent overwriting exiting files.

When done select OK.
This is the next dialog that Visual Studio displays. If there are no specific project options to select (shown in next slide) the user can select Finish at the bottom.

For this example, the Next > option is selected. It is advised for class projects that students select Next >.
Visual Studio New Project Options Selection

This dialog allows the user to fine tune the project creation. By default the Console application radio option is selected (earlier, the Win32 Console Application project type was selected). If the project is a console application library development, the DLL or Static library option should be selected.

For the class, Console application default is correct.

Under the Additional options check boxes, deselect Precompiled header and Security Development Lifecycle options.

The Empty project option can be selected if the user already has the needed source files and only needs a project file structure created.

For this example, a basic new project will be created.

When done, select Finish and Visual Studio will generate/create the project.
Visual Studio Project Display

This is the Visual Studio Project display. The main window contains a tabbed list of open source files. This example shows the default file created as a new project. Note, code created is sufficient to compile and run a basic simple program. The basic example application shown on an earlier slide was created this way.

On the right is the solution explorer that shows the type of project data including files. The include and source files are listed.

The solution name, HelloWorld, is highlighted and selected. When the solution is selected by right clicking on it a popup menu is displayed. There are several options available; at the bottom of the list is the Properties. When Properties is selected, the following dialog will be displayed.
The General properties are initially displayed by default. Any other options selected during the process of setting options will set the next default to the currently active display.

These options will be discussed, or the student can reference further descriptions at Microsoft’s developers site.

The C/C++ options have been expanded to show the additional options available.

The next slide shows the options available when VC++ Directories is selected.
Visual Studio Project VC Directories

This dialog shows the default directories provided by Visual Studio. These are the directories where Visual Studio, the compiler and linker will obtain the files needed to build and run the application.

The three directories of interest are the Include, Library and Source directory lists. The developer can change these lists as needed. The user will usually add directories where the user positions files used by this (and possible other) project(s).
Directories and Files Created by VS

This shows the project root directory. The file with the .sln extension is the project control file. It contains all of the project property settings needed to track the files used by the project.
This shows the detailed project files including the source and header files created (as shown in the Visual Studio example) and the project properties settings files.
Advance Topics