INTRODUCTION TO COMPUTERS & C

SDN 150S – Week 1

Lecturer: Mr. Stephen Ekwe

Cape Peninsula University of Technology

creating futures

Outline

HISTORY OF COMPUTER

INTRO TO SDLC

INTRO TO C PROGRAMMING

What is a Computer

- A computer is a programmable electronic device designed to accept some input data, which is processed in line with a sequence of instructions called programs and provides the output in the desired format.

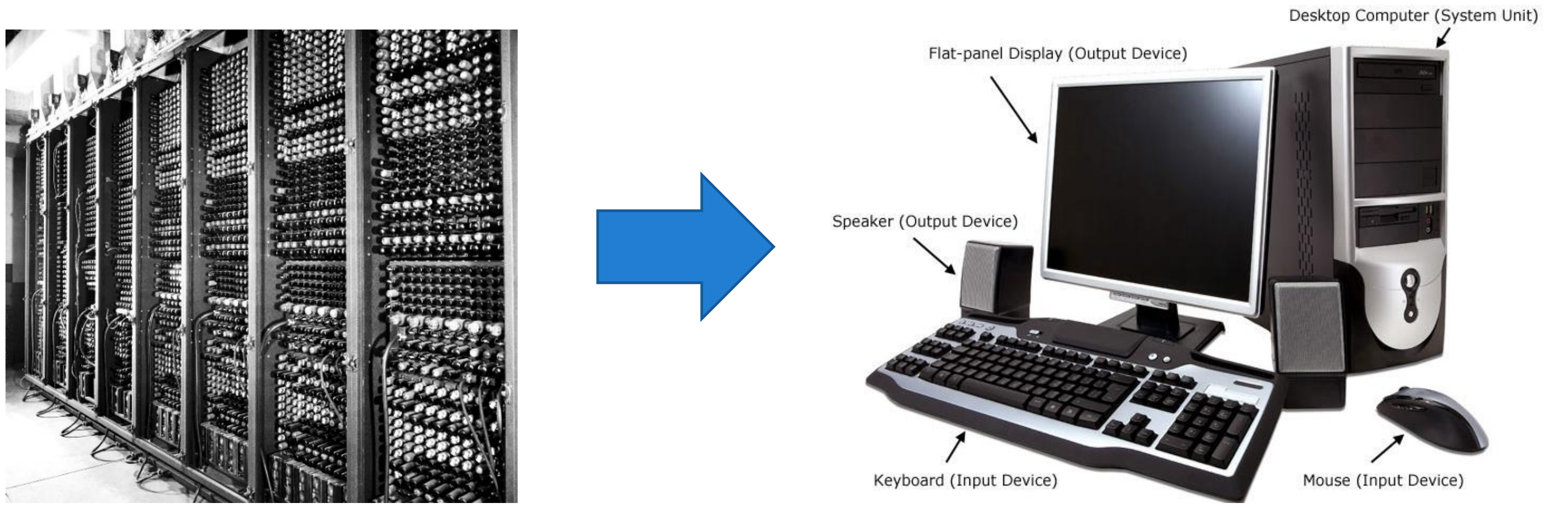


Figure 1.1: Evolution of computers (1st and 5th generation)

History of Computer

Generation	1st Gen 1940s-1950s	2nd Gen 1950s-1960s	3rd Gen 1960s-1970s	4th Gen 1970s-1980s	5th Gen Present
Electronic Component	Vacuum tube	Transistor	Integrated circuits	Very Large-Scale Integration (VLSI) and Microprocessor	Ultra Large-Scale Integration (ULSI) technology, AI, parallel processing method
Memory	Magnetic drums	Magnetic core	Large magnetic core	Semiconductor memory (RAM, ROM, etc.)	Huge Storage Capacity
Programming Language	Machine language	Assembly language	High level language (FORTRAN, BASIC, C, etc.)	High level language (Python, C#, Java, etc.)	Natural language
Energy Consumption	Very high	High	Low	Low	Very Low
Examples	IBM 650, IBM 701	BM 1401. IBM 7090	IBM 360, IBM 370	IBM PC, STAR 1000, Apple Macintosh, etc.	Current desktop, laptop, tablets device, etc.

Hardware & Software

- Computers can perform calculations and make logical decisions phenomenally faster than human beings can.
- Fujitsu's Fugaku, which is currently the world's fastest supercomputer can perform 442 quadrillion calculations per second (442 petaflops).
- That is almost 58 million calculations for every person on the planet in one second.
- Computers process data under the control of sequences of instructions called programs.
- These programs guide the computer through ordered actions specified by people called computer programmers

Hardware & Software

- A computer consist of various physical devices referred to as hardware, which aid the operation of a collection of programs called software.
- Over the years, computing costs are dropping dramatically due to rapid advancements in hardware and software technologies.
- This trend was first observed In 1965 by Gordon E. Moore, the co-founder of Intel, and it later became known as Moore's Law.
- Moore's Law states that the number of transistors on a microchip doubles every two years, though with a 50% reduction in cost.
- This means that the cost of computers will grow cheaper while their speed and capability keeps increasing every few years.

Computer Organization

Regardless of physical differences, computers can be divided into various functional units.

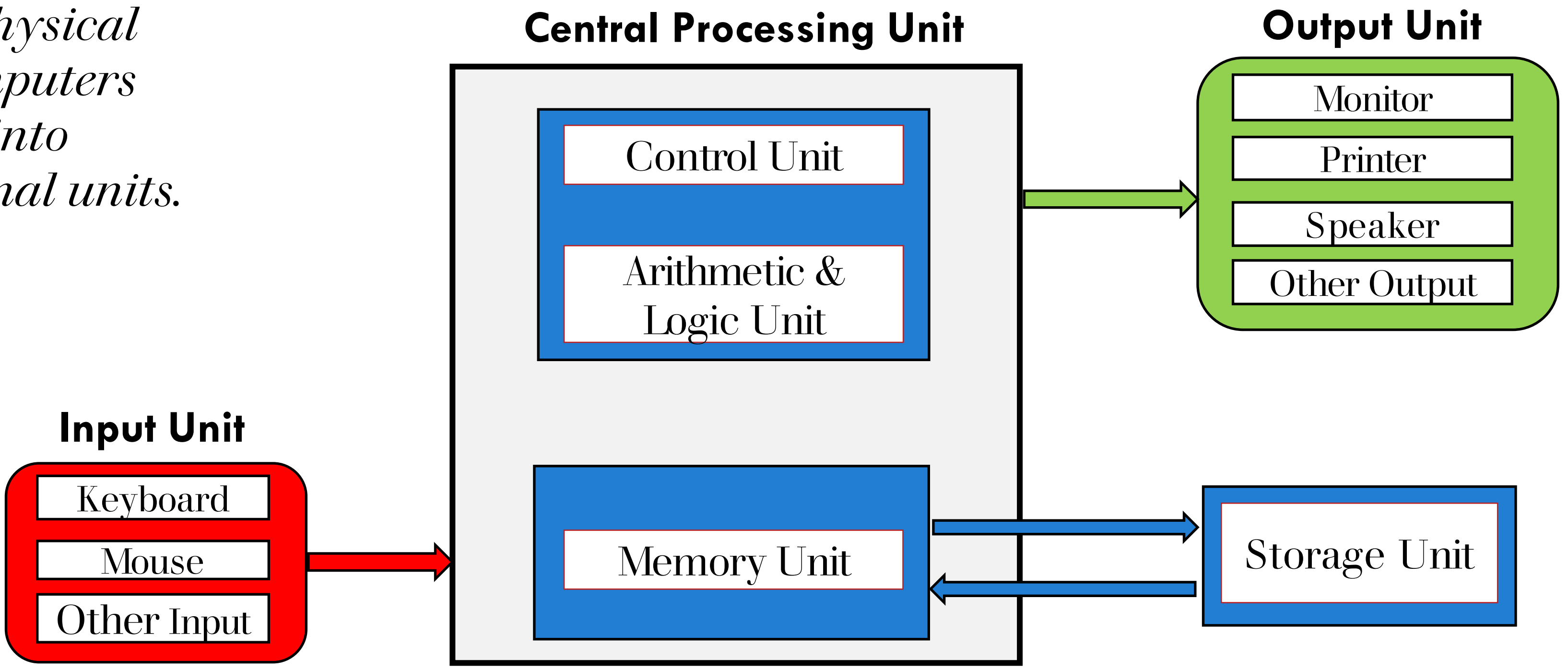


Figure 1.2: Basic organization of a computer system

Computer Organization

Input Unit

- This takes information (data and computer programs) from input devices and places it at the other units' disposal for processing.
- The most used input devices are keyboards, mouse, joysticks, trackballs, microphones, etc.
- The most well-known input device is a keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Central Processing Unit

- This is the electronic circuitry called a processor within a computer that carries out the instructions given by a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

Computer Organization

Control Unit

- This is a component of a computer's central processing unit that coordinates the operation of the processor.
- It tells the computer's memory, arithmetic/logic unit and input and output devices how to respond to a program's instructions.
- The control unit is also known as the nerve centre of a computer system.

Arithmetic & Logic Unit

- Most of all the arithmetic and logic operations of a computer are executed in the ALU (Arithmetic and Logic Unit) of the processor.
- It performs arithmetic operations like addition, subtraction, multiplication, division and also the logical operations like AND, OR, NOT operations.

Computer Organization

Memory Unit

- This refers to as the storage area that holds data and the programs used in processing data.
- The Memory unit can be categorized in two ways namely, primary memory and secondary memory.
- It enables a processor to access running execution applications and services that are temporarily stored in a specific memory location.
- Primary memory is the fastest memory that operates at electronic speeds. It contains a large number of semiconductor storage cells, capable of storing a bit of information.
- A bit (short for “binary digit”) is either 0 or 1. The word length of a computer is between 16-64 bits. While a byte is given as 8 bits.
- The most common examples of primary memory are RAM (Random Access Memory) and ROM (Read only Memory).

Computer Organization

- The Memory unit is also known as the volatile form of memory, since it losses anything contained in RAM when the computer is shut down.
- Cache memory is also a kind of memory which is used to fetch the data very soon. They are highly coupled with the processor.
- Secondary memory is used when a large amount of data and programs have to be stored for a long-term basis.
- It is also known as the Non-volatile memory form of memory, means the data is stored permanently irrespective of shut down.
- The most common examples of secondary storage include solid-state drives (SSDs), hard disk drives (HDD), Universal Serial Bus (USB) flash drives, and read/write Blu-ray drives.

Computer Organization

Output Unit

- This functional unit send the processed data in a desirable format to the user via an output device.
- Output devices are pieces of equipment that are designed to present information in a way that the user can understand. The most common examples of output devices are monitor, printer, and speaker.

Computer Program

- A computer program is a collection of instructions that performs a specific task when executed by a computer.
- A computer program is usually written by a computer programmer in either high-level or low-level language, depending on the task and the hardware of the computer system.
- A programming language Is a formal language used to translate a set of instructions to a computer for various kinds of output.
- The most computer programming languages are written in a high-level programming language, which uses the common English language to help make the code more understandable and to speed up the process of writing and debugging programs.

Computer Program

- However, computer use their own language to communicate with one another, which is written using a binary code called Machine language.
- A machines language is a low-level language that consists of bits (1s and 0s) put together into chunks like bytes (8 bits), and lots of other larger sizes.
- An Assembly language is also a low-level language, however, its a little easier than machine language.
- It uses alphanumeric code to describe the huge strings of 1s and 0s, making it both easier and more memorable to type in instructions.

Computer Program

- With assembly language, the computer interprets alphanumeric code as binary code, which allows for the use of English-like strings instead of just 1s and 0s.

Table 1: Differences between Low-level and High-level Languages

Languages	Examples	Description	Example Instructions

High-level Language	Python, Visual basic, Java, C	One statement translates into many lines of machine code instructions, using a complier or interpreter independent of hardware.	Salary = 35 478;
			Tax = 6520;
			NetPay = Salary – Tax

Low-level Language	Machine Language	Executes binary code produced by a complier, interpreter, or assembler	1101010010100111
	Assembly Language	One statement translates into one line of machine code instructions using an assembler.	LDA181 ADD93 STO185

Introduction to Software Development Life Cycle (SDLC)

What is SDLC

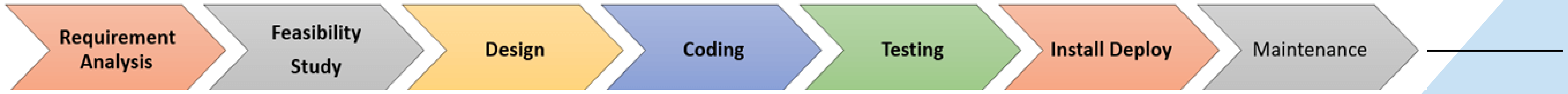
- SDLC is an abbreviation for Software Development Life Cycle, which is also known as the Application Development Life Cycle.
- SDLC is a comprehensive plan that describes how to plan, build, and maintain specific software.
- Each phase of the SDLC life cycle has its own set of processes and deliverables that feed into the next.
- The SDLC process aims to produce high-quality software that meets the needs of customers.
- The development should be completed within the time and budget constraints.

Why SDLC

- It offers a basis for project planning, scheduling, and estimating
- Provides a framework for a standard set of activities and deliverables
- It is a mechanism for project tracking and control
- Increases visibility of project planning to all involved stakeholders of the development process
- Increased and enhance development speed
- Improved client relations
- Helps you to decrease project risk and project management plan overhead

SDLC Phases

- Phase 1: Requirement collection and analysis
- Phase 2: Feasibility study
- Phase 3: Design
- Phase 4: Coding
- Phase 5: Testing
- Phase 6: Installation/Deployment
- Phase 7: Maintenance



Phase 1: Requirement collection and analysis

- The SDLC process begins with the requirements. It is led by senior team members with input from all stakeholders and industry domain experts.
- This stage provides a clearer picture of the overall project's scope as well as the anticipated issues, opportunities, and directives that triggered the project.
- This stage also includes planning for quality assurance requirements and recognizing the risks involved.
- Requirements teams must gather detailed and precise requirements during this stage to assists businesses in finalizing the necessary timeline to complete the system's work.

Phase 2: Feasibility study

- This phase is carried out with the assistance of the Software Requirement Specification (SRS) document.
- It encompasses everything that must be designed and developed throughout the project's life cycle.
- There are five main types of feasibility checks:
  - **Economic**: Can we finish the project within the budget?
  - **Legal**: Are we able to handle this project in light of cyber law and other regulatory frameworks/compliances?
  - **Operation feasibility**: Can we create the operations that the client expects?
  - **Technical**: Determine whether the current computer system is capable of supporting the software.
  - **Schedule**: Determine whether the project can be completed within the time frame specified.

Phase 3: Design

- SRS serves as a resource for product architects to develop the best architecture for the product under development.
- Typically, multiple design approaches for the product architecture are proposed and documented in a DDS (Design Document Specification) based on the requirements specified in the SRS.
- This DDS is reviewed by all important stakeholders, and the best design approach for the product is chosen based on various parameters such as risk assessment, product robustness, design modularity, budget, and time constraints.

Phase 4: Coding

- The actual development of the product begins at this stage of the SDLC. The programming code is generated in accordance with DDS.
- Code generation can be accomplished without much difficulty if the design is detailed and organized. Hence the developers must adhere to their organization's coding guidelines.
- Programming tools such as compilers, interpreters, debuggers, and so on are used to generate code. Various high-level programming languages such as C, C++, Python, Java, and PHP are used for coding. The programming language is chosen based on the type of software being created.

Phase 5: Testing

- As in modern SDLC models, testing activities are mostly involved in all SDLC stages, this stage is usually a subset of all stages.
- However, this stage refers to the product's testing only stage, during which defects are reported, tracked, fixed, and retested until the product meets the quality standards defined in the SRS.

Phase 6: Installation/Deployment

- Once the product has been thoroughly tested and is ready for deployment, it is formally released in the appropriate market.
- Product deployment may occur in stages depending on the organization's business strategy.
- The product may first be released in a limited market segment and tested in a real-world business environment (UAT).
- The product may then be released as is or with suggested enhancements in the targeted market segment based on the feedback.

Phase 7: Maintenance

- Once the system is deployed and customers start using the developed system, the following three activities occur:
- Bug fixing: bugs are reported because of some scenarios that are not tested at all.
- Upgrade: upgrading the application to the newer versions of the software
- Enhancement: Adding some new features to the existing software
- The focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specifications mentioned in the first phase.

Introduction to C Programming

C Programming

- C is a general-purpose, high-level language that was originally implemented by Dennis M. Ritchie in 1972, to develop the UNIX operating system at Bell Laboratories.
- C is a successor of B language which was introduced around the early 1970s, and was later formalized in 1988 by the American National Standard Institute (ANSI).
- Most of the state-of-the-art software are written in either C and/or C++.
- C is a structured language that is easy to learn, produces efficient programs, handle low-level activities, and also compiles efficiently on a variety of computing platforms.

- The current update on the C standard document is referred to as ISO/IEC 9899:2018.

C Programming

- This update which was first approved in 2011 and updated in 2018, both refined and expanded the functionalities of the C programming language capabilities.
- The C program can vary from 3 lines to millions of lines of code depend on the nature of the program being developed .
- It can be written into one or more text files with extension "`.c`"; for example, "`main.c`".
- C is widely used for system development work, which requires performance, such as in operating systems, embedded systems, real-time systems and communications systems.
- C was adopted as a system development language because it produces code that runs nearly as fast as the code written in assembly language.

A Simple C Program: Printing a Line of Text

```c
// example01_01.c
// my first program in C.
#include <stdio.h>

// function main begins program execution
int main(void) {
   printf("Welcome to C!\n");
} // end function main
```

- **OUTPUT:**

  -
    ```
    Welcome to C!
    ```

A Simple C Program: Printing a Line of Text

Comments

- Begin with //
- Insert comments to document programs and improve program readability
- Comments do not cause the computer to perform actions
- Help other people read and understand your program
- Can also use /*...*/ multi-line comments
- Everything between /* and */ is a comment

A Simple C Program: Printing a Line of Text

#include Preprocessor Directive

- `#include <stdio.h>`
- This line of the program is a preprocessor directive that tells a C compiler to include `stdio.h` file before going to actual compilation.
- Preprocessor handles lines beginning with # before compilation
- This directive includes the contents of the standard input/output header (`<stdio.h>`)
- Contains information the compiler uses to ensure that you correctly use standard input/output library functions such as `printf`

A Simple C Program: Printing a Line of Text

Blank Lines and White Space

- Blank lines, space characters and tab characters make programs easier to read
- Together, these are known as white space
- Generally ignored by the compiler

A Simple C Program: Printing a Line of Text

The main Function

- Begins execution of every C program
- Parentheses after main indicate that main is a function
- C programs consist of functions, one of which must be main
- Precede every function by a comment stating its purpose
- Functions can return information
- The keyword int to the left of main indicates that main “returns” an integer (whole number) value – for now simply mimic this

A Simple C Program: Printing a Line of Text

- Although functions can receive information,“`void`” in parentheses explicitly informs the main function not to return any information
- A left brace, { , begins each function’s body
- A corresponding right brace, } , ends each function’s body
- A program terminates upon reaching main’s closing right brace
- The braces form a complete block of instruction.

A Simple C Program: Printing a Line of Text

An Output Statement

- `printf("Welcome to C!\n");`
- Where “f” in `printf` stands for “formatted”
- This is another function that performs an action to displays the string in the quotes
- A string is also called a character string, a message or a literal
- The entire line is called a statement
- Every statement ends with a semicolon statement terminator
- Characters usually print as they appear
- Notice the characters `\n` were not displayed.

A Simple C Program: Printing a Line of Text

Escape Sequences

- In a string, backslash (\) is an escape character
- Compiler combines a backslash with the next character to form an escape sequence
- \n means newline
- When `printf` encounters a newline in a string, it positions the output cursor to the beginning of the next line

A Simple C Program: Printing a Line of Text

Escape Sequence	Description
`\n`	Moves the cursor to the beginning of the next line.
`\t`	Moves the cursor to the next horizontal tab stop.
`\a`	Produces a sound or visible alert without changing the current cursor position.
`\\`	Because the backslash has special meaning in a string, **`\\`** is required to insert a backslash character in a string.
`\″`	Because strings are enclosed in double quotes, **`\″`** is required to insert a double-quote character in a string.

A Simple C Program: Printing a Line of Text

The Linker and Executables

- When compiling a `printf` statement, the compiler merely provides space in the object program for a “call” to the function
- The compiler does not know where the library functions are, but the linker does
- When the linker runs, it locates the library functions and inserts the proper calls to these functions in the object program
- Now the object program is complete and ready to execute
- The linked program is called an **executable**

A Simple C Program: Printing a Line of Text

Indentation Conventions

- Indent the entire body of each function
- one level of indentation within the braces that define the function's body
- It emphasizes a program's functional structure and help makes it easier to read
- Set an indentation convention and uniformly apply that convention
- It is often recommend using spaces rather than tabs when indenting.

A Simple C Program: Printing a Line of Text

Using Multiple **`printfs`**

- `example01_02.c` in the next slide uses two statements to produce the same output as `example01_01.c` slide 32.
- Each `printf` function resumes printing where the previous one finished
- Line 7 displays `Welcome` followed by a space (but no newline)
- Line 8's `printf` begins printing immediately following the space

```
1.  // example01_02.c
2.  // Printing on one line with two printf
    statements.
3.  #include <stdio.h>
4.
5.  // function main begins program execution
6.  int main(void) {
7.     printf("Welcome ");
8.     printf("to C!\n");
9.  } // end function main
```

- **OUTPUT:**

  -
    ```
    Welcome to C!
    ```

A Simple C Program: Printing a Line of Text

Displaying Multiple Lines with a Single ***printfs***

- One `printf` can display several lines
- Each `\n` moves the output cursor to the beginning of the next line

```c
// example01_02.c
// Printing on one line with two printf
statements.
#include <stdio.h>

// function main begins program execution
int main(void) {
   printf("Welcome\nto\nC!\n");
} // end function main
```

- **OUTPUT:**

```
Welcome
to
C!
```

A Simple C Program: Printing a Line of Text

Tokens

- Tokens are the fundamental element of a C program.
- A token can either be a keyword, an identifier, a constant, a string literal, or a symbol.
- For example, the printf statement in Line 7, **`example01_01.c`** consist of five tokens:

```
1. printf
2. (
3. "Welcome to C!\n"
4. )
5. ;
```

Another Simple C Program: Adding Two Integers

```c
// example01_04.c
// Addition program.
#include <stdio.h>

// function main begins program execution
int main(void) {
   int integer1 = 0; // will hold first number user enters
   int integer2 = 0; // will hold second number user enters
    printf("Enter first integer: "); // prompt
   scanf(“%d“, &integer1); // prompt
   printf("Enter second integer: "); // prompt
   scanf ("%d“, &integer2); // read an integer
   int sum = 0; // variable in which sum will be stored
   sum = integer1 + integer2; // assign total to sum

   printf("Sum is %d\n", sum); // print sum
} // end function main
```

OUTPUT:

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Another Simple C Program: Adding Two Integers

Variables and Variable Definitions

- In `example01_04.c` Lines 7 and 8 are definitions.
- The names `integer1` and `integer2` are variables, which are locations in memory that can store values for later use
- `integer1` and `integer2` have a data type int, they'll hold only whole-number integer values
- Lines 7 and 8 initialize each variable to 0
- All variables must be defined with a name and a type before they can be used in a program
- You can place each variable definition anywhere in main before that variable's first use in the code

Another Simple C Program: Adding Two Integers

Identifiers and Case Sensitivity

- A C identifier is a name used to identify a variable, function, or any other user defined item.
- An identifier starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9).
- C does not allow punctuation characters such as @, $, and % within identifiers.
- C is a case-sensitive programming language. Thus, Manpower and manpower are two different identifiers in C.

Another Simple C Program: Adding Two Integers

Prompting Messages

- Line 9 displays "Enter first integer: "
- This message is called a prompt. It tells the user to take a specific action

The scanf Function and Formatted Inputs

- Line 10 uses scanf to obtain input value from the user (usually read from the keyboard)
- The "%d" is the format control string, which indicates the type of data the user should enter (an integer)
- Second argument begins with an ampersand (&) followed by the variable name
- Tells scanf the location (or address) in memory of the variable
- scanf stores the value the user enters at that memory location

Another Simple C Program: Adding Two Integers

Prompting for and Inputting the Second Integer

- Line 11 prompts the user to enter the second integer
- Line 12 obtains a value for variable integer2 from the user.

The Defining the sum Variable

- Line 14 defines the int variable sum and initializes it to 0 before we use sum in line 15.

Assignment Statement

- The assignment statement in line 15 calculates the total of integer1 and integer2, then assigns the result to variable sum using the assignment operator (=)
- Read as, "sum gets the value of the expression integer1 + integer2."
- Most calculations are performed in assignments

Another Simple C Program: Adding Two Integers

Binary Operators

- The = operator and the + operator are binary operators—each has two operands
- Place spaces on either side of a binary operator to make the operator stand out and make the program more readable

Printing with a Format Control String

- The format control string “Sum is %d\n” in line 17 contains some literal characters to display ("Sum is ") and the conversion specification %d, which is a placeholder for an integer
- The sum is the value to insert in place of %d

Another Simple C Program: Adding Two Integers

Combining a Variable Definition and Assignment Statement

- You can initialize a variable in its definition
- For example, lines 14 and 15 can add the variables integer1 and integer2, then initialize the variable sum with the result:
- `int sum = integer1 + integer2;` // assign total to sum

Calculations in printf Statements

- Actually, we do not need the variable sum, because we can perform the calculation in the `printf` statement
- Lines 14–17 can be replaced with:
- `printf("Sum is %d\n", integer1 + integer2);`