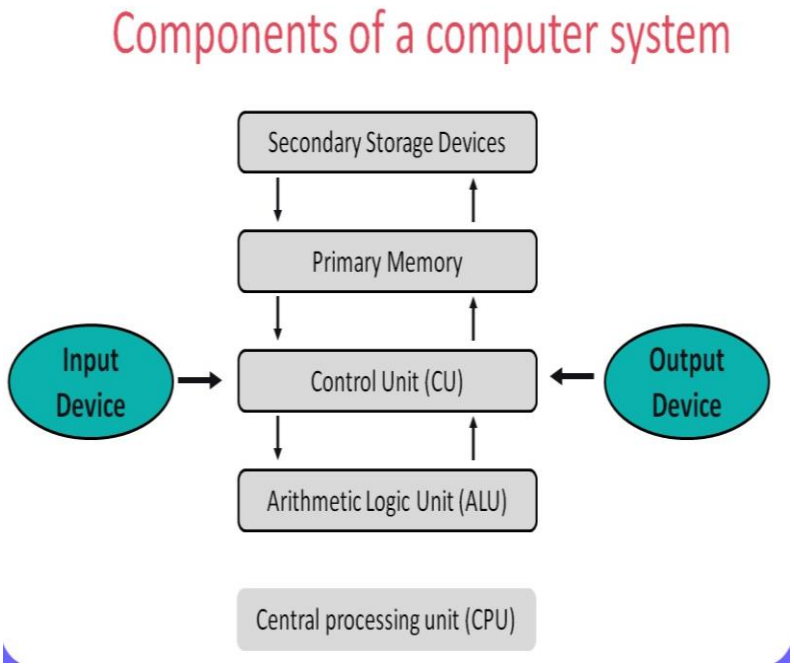
**Introduction**

A programming language is a set of symbols, grammars and rules with the help of which one is able to translate algorithms to programs that will be executed by the computer. The programmer communicates with a machine using programming languages. Most of the programs have a highly structured set of rules.

**Introduction to components of a computer system**

A computer system is made up of five basic components:

1. Central processing unit (CPU)
2. Memory
3. Input/output devices
4. Storage devices
5. Control unit (CU)




The components work together to produce the desired output. The CPU processes data and generates information that is displayed to the user through output devices. The information is stored in the computer's memory.

The five basic components of a computer are:

- Motherboard
- Graphics processing unit
- Random access memory (RAM)
- Hard disk or solid-state drive

Input devices include: Mice, Scanners, and Microphones.
Storage devices include:

- Memory cards
- Hard disks
- Solid-state drives

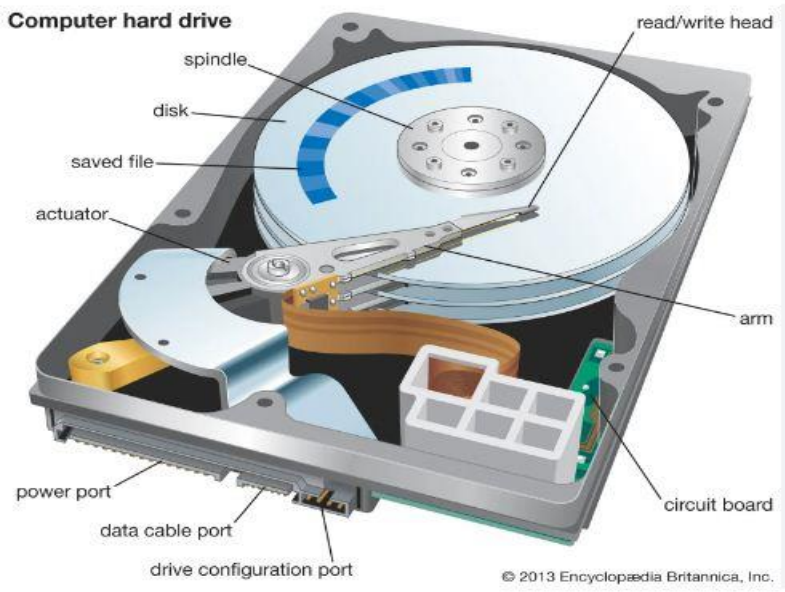
**Disk**

A disk is a flat, round plate that stores data. They are also known as diskettes. Hard disks are a common component of a computer's storage system. They are made of aluminum

or glass and coated with a magnetic material. Hard disks can store terabytes (trillions of bytes) of information.

Hard disks are non-volatile, which means the data retains when the computer shuts down. They are installed internally in computer systems.

Other types of disks include: Floppy disks, CD-ROMs, SSDs.




**You can improve hard disk performance by:**

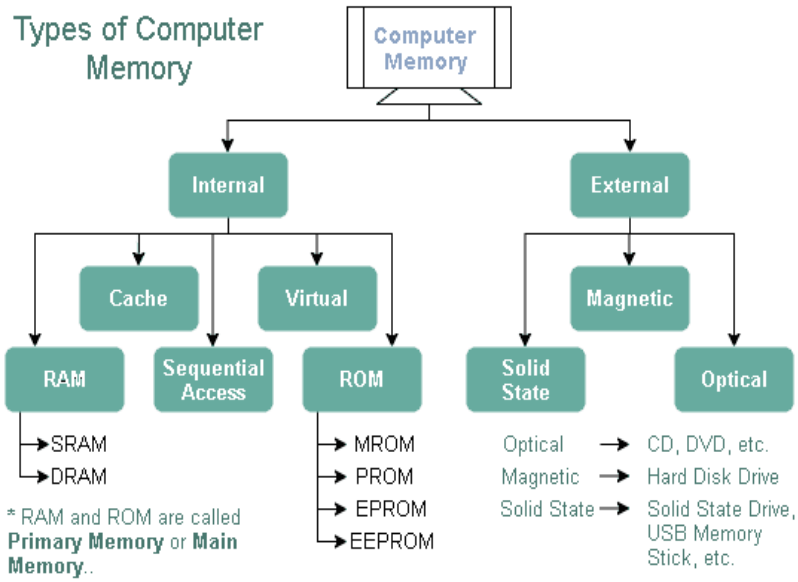
- Defragmenting your hard disk drive
- Deleting temporary files

**Memory**

Computer memory is the storage space in a computer that stores data and instructions. It's divided into small parts called cells.

**There are two types of memory in a computer:**

- **Primary memory:** Includes RAM and ROM, which allow quick data access and temporary storage for running programs.
- **Secondary memory:** Includes HDDs and SSDs, which provide long-term data storage.



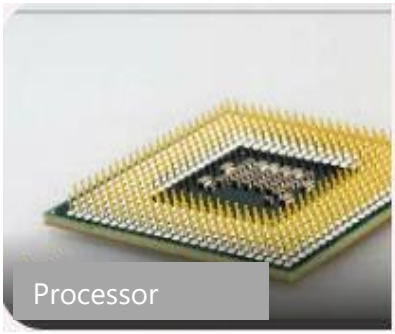

Computer memory uses semiconductor technology and is commonly called semiconductor memory

**Processor**

A processor, also known as a CPU, is a circuit board inside a computer that executes instructions on behalf of programs. It's the main chip on a computer and is responsible for interpreting most of the computer's commands.

**A processor performs the following functions:**

- Responds to and processes basic instructions that drive a computer
- Performs arithmetical, logical, input/output (I/O), and other basic instructions that are passed from an operating system (OS)
- Processes inputs into outputs
- Controls computer speed





A processor can process millions of instructions in a second. It's often referred to as the "Brain of a PC" because all the computations and processing are carried out directly or indirectly by the processor.

**Compiler**

A compiler is a special program that translates a programming language's source code into machine code, bytecode or another programming language. The source code is typically written in a high-level, human-readable language such as Java or C++.

A C compiler is a computer program that translates C source code into machine code. Machine code is a low-level programming language that can be understood directly by a computer's CPU. C compilers are typically used to create executable programs, which are files that can be run directly by the user.

C compilers are written in C or C++. They are typically very large and complex programs, as they need to be able to understand the full C language specification. C compilers also need to be able to optimize the generated machine code, in order to improve the performance of the resulting program.

There are many different C compilers available, both free and commercial. Some popular C compilers include GCC, Clang, and Intel's C compiler.

**Here are some examples of how C compilers are used:**

- To create executable programs.
- To compile C source code for use with other tools, such as linkers and debuggers.
- To generate machine code for embedded systems.
- To generate machine code for high-performance applications.

C compilers are a vital tool for any C programmer. They allow programmers to write code that can be executed on a variety of platforms, and they can help to improve the performance of C programs.

**Operating System**

An operating system (OS) is software that acts as an interface between a computer user and the computer's hardware. It performs basic tasks like:

- File management

- Memory management
- Process management
- Handling input and output
- Controlling peripheral devices such as disk drives and printers

An OS is the most important software that runs on a computer. It manages the computer's memory and processes, as well as all of its software and hardware. It also allows you to communicate with the computer without knowing how to speak the computer's language.

Every computer system must have at least one operating system to run other programs. Applications like Browsers, MS Office, and Notepad Games need an environment to run and perform their tasks.

The cost of an operating system may vary if your machine doesn't already have one. Many devices have built-in operating systems, but if you plan to use more memory or require more specific features, you may need to purchase one yourself.

**Example,** iOS, Windows 10, Windows 11, Chrome OS, macOS Sierra, Android, Windows 7, Ubuntu

**Algorithm**

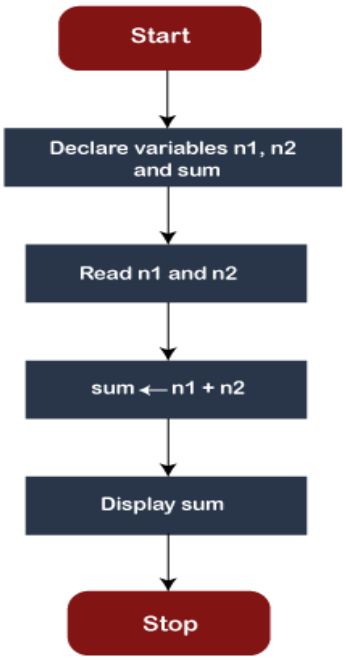
An algorithm is a sequence of instructions that are carried out in a predetermined sequence in order to solve a problem or complete a work. A function is a block of code that can be called and executed from other parts of the program. A set of instructions for resolving an issue or carrying out a certain activity.

**Algorithm 1: Add two numbers entered by the user**

Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.

sum←num1+num2

Step 5: Display sum
Step 6: Stop




**Flowchart**

In C programming, flowcharts are often used to represent algorithms or programs. They show the connections, flow of information, and processes within an algorithm or a program. For example, here's an if else flowchart that's often used in C programming.

**Pseudocode**

Pseudocode is an informal way of writing a program in simple English to make it easier for humans to understand. It's a high-level illustration of the program's logic and flow that can serve as a guide for developing the real code.

Pseudocode is language independent, meaning that it can be converted to any language code. It has no syntax like any of the programming language and thus can't be compiled or interpreted by the computer.

**Algorithm 1: Add two numbers entered by the user**

Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.
sum←num1+num2
Step 5: Display sum
Step 6: Stop

**Variables**

In programming, a variable is a container (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location.

Variable is basically nothing but the name of a memory location that we use for storing data. We can change the value of a variable in C or any other language, and we can also reuse it multiple times. We use symbols in variables for representing the memory location- so that it becomes easily identifiable by any user. Variables can be declared of different types, such as integers, floats, and characters.

For example:

```
int playerScore = 95;
int x;
float y;
```

**Rules for naming a variable**

A variable name can only have letters (both uppercase and lowercase letters), digits and underscore.

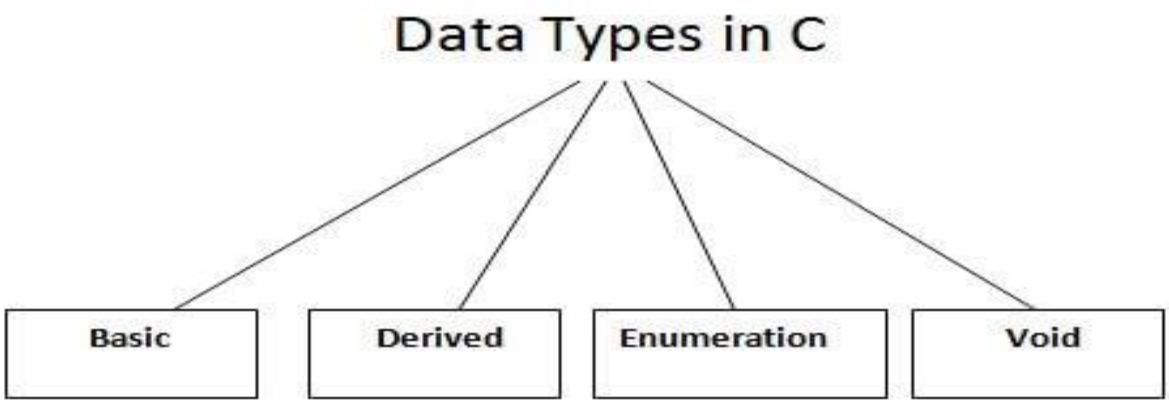
The first letter of a variable should be either a letter or an underscore.

There is no rule on how long a variable name (identifier) can be. However, you may run into problems in some compilers if the variable name is longer than 31 characters.

**Data Type**

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

There are the following data types in C language.




| Types | Data Types |
|---|---|
| Basic Data Type | int, char, float, double |
| Derived Data Type | array, pointer, structure, union |
| Enumeration Data Type | enum |
| Void Data Type | void |

**int:**

***Integers*** are entire numbers without any fractional or decimal parts, and the ***int data type*** is used to represent them.

It is frequently applied to variables that include ***values***, such as ***counts, indices***, or other numerical numbers. The ***int data type*** may represent both ***positive*** and ***negative numbers*** because it is signed by default.

An ***int*** takes up ***4 bytes*** of memory on most devices, allowing it to store values between around -2 billion and +2 billion.

**char:**

Individual characters are represented by the ***char data type***. Typically used to hold ***ASCII*** or ***UTF-8 encoding scheme characters***, such as ***letters, numbers, symbols***, or ***commas***. There are ***256 characters*** that can be represented by a single char, which takes up one byte of memory. Characters such as ***'A', 'b', '5',*** or ***'$'*** are enclosed in single quotes.

**float:**

To represent integers, use the ***floating data type***. Floating numbers can be used to represent fractional units or numbers with decimal places.

The ***float type*** is usually used for variables that require very good precision but may not be very precise. It can store values with an accuracy of about ***6 decimal places*** and a range of about ***3.4 x 1038*** in ***4 bytes*** of memory.

**double:**

Use two data types to represent ***two floating integers***. When additional precision is needed, such as in scientific calculations or financial applications, it provides greater accuracy compared to float.

***Double type***, which uses ***8 bytes*** of memory and has an accuracy of about ***15 decimal places, yields larger values***. C treats floating point numbers as doubles by default if no explicit type is supplied.

```
int age = 25;
char grade = 'A';
float temperature = 98.6;
double pi = 3.14159265359;
```

In the example above, we declare four variables: an ***int variable*** for the person's age, a ***char variable*** for the student's grade, a ***float variable*** for the temperature reading, and two variables for the ***number pi.***

## Derived Data Type

Beyond the fundamental data types, C also supports ***derived data types,*** including ***arrays, pointers, structures,*** and ***unions***. These data types give programmers the ability to handle heterogeneous data, directly modify memory, and build complicated data structures.

**Array:**

An ***array, a derived data type***, lets you store a sequence of ***fixed-size elements*** of the same type. It provides a mechanism for joining multiple targets of the same data under the same name.

The index is used to access the elements of the array, with a ***0 index*** for the first entry. The size of the array is fixed at declaration time and cannot be changed during program execution.

```
#include <stdio.h>

int main() {
    int numbers[5]; // Declares an integer array with a size of 5 elements
    // Assign values to the array elements
    numbers[0] = 10;
    numbers[1] = 20;
    numbers[2] = 30;
    numbers[3] = 40;
    numbers[4] = 50;
   // Display the values stored in the array
    printf("Values in the array: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");

    return 0;
}
```

**Output:**

```
Values in the array: 10 20 30 40 50
```

**Pointer:**

A ***pointer*** is a derived data type that keeps track of another data type's memory address. When a ***pointer*** is declared, the ***data type*** it refers to is ***stated first***, and then the ***variable name*** is preceded by ***an asterisk (*)***.

You can have incorrect access and change the value of variable using pointers by specifying the memory address of the variable. ***Pointers*** are commonly used in ***tasks*** such as ***function pointers, data structures***, and ***dynamic memory allocation***.

```
#include <stdio.h>
```

```
int main() {
   int num = 42;      // An integer variable
   int *ptr;          // Declares a pointer to an integer
  ptr = #        // Assigns the address of 'num' to the pointer
   // Accessing the value of 'num' using the pointer
   printf("Value of num: %d\n", *ptr);
     return 0;
}
```

**Output:**

Value of num: 42

**Structure:**

A structure is a derived data type that enables the creation of composite data types by allowing the grouping of many data types under a single name. It gives you the ability to create your own unique data structures by fusing together variables of various sorts.

1. A structure's members or fields are used to refer to each variable within it.
2. Any data type, including different structures, can be a member of a structure.
3. A structure's members can be accessed by using the dot (.) operator.

```
#include <stdio.h>
#include <string.h>
// Define a structure representing a person
struct Person {
   char name[50];
   int age;
   float height;
};

int main() {
   // Declare a variable of type struct Person
   struct Person person1;

   // Assign values to the structure members
   strcpy(person1.name, "John Doe");
   person1.age = 30;
   person1.height = 1.8;

   // Accessing the structure members
   printf("Name: %s\n", person1.name);
   printf("Age: %d\n", person1.age);
```

```
    printf("Height: %.2f\n", person1.height);

    return 0;
}
```

**Output:**

Name: John Doe
Age: 30
Height: 1.80

## Union:

A derived data type called a ***union*** enables you to store various data types in the same memory address. In contrast to structures, where each member has a separate memory space, members of a union all share a single memory space. A value can only be held by one member of a union at any given moment. When you need to represent many data types interchangeably, unions come in handy. Like structures, you can access the members of a union by using the ***dot (.)*** operator.

```
#include <stdio.h>
// Define a union representing a numeric value
union NumericValue {
    int intValue;
    float floatValue;
    char stringValue[20];
};
int main() {
    // Declare a variable of type union NumericValue
    union NumericValue value;
    // Assign a value to the union
    value.intValue = 42;
    // Accessing the union members
    printf("Integer Value: %d\n", value.intValue);
    // Assigning a different value to the union
    value.floatValue = 3.14;
    // Accessing the union members
    printf("Float Value: %.2f\n", value.floatValue);
      return 0;
}
```

**Output:**

Integer Value: 42
Float Value: 3.14

## Enumeration Data Type

A set of named constants or ***enumerators*** that represent a collection of connected values can be defined in C using the ***enumeration data type (enum). Enumerations*** give you the means to give names that make sense to a group of integral values, which makes your code easier to read and maintain.

```
#include <stdio.h>
  // Define an enumeration for days of the week
enum DaysOfWeek {
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday,
    Sunday
};

int main() {
    // Declare a variable of type enum DaysOfWeek
    enum DaysOfWeek today;

    // Assign a value from the enumeration
    today = Wednesday;

    // Accessing the enumeration value
    printf("Today is %d\n", today);

    return 0;
}
```

**Output:**

```
Today is 2
```

## Void Data Type

The ***void data type*** in the C language is used to denote the lack of a particular type. ***Function return types, function parameters***, and ***pointers*** are three situations where it is frequently utilized.

## Function Return Type:

A ***void return type*** function does not produce a value. A ***void function*** executes a task or action and ends rather than returning a value.

## Variable and Memory Location

### Variable

A variable is a name given to a location in memory that can store data. The data stored in a variable can be changed at any time.

### Memory location

A memory location is a physical location in the computer's memory where data can be stored. The memory location is identified by its address, which is a unique number.

## Typecasting

Converting one datatype into another is known as type casting or, type-conversion. For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'. You can convert the values from one type to another explicitly using the cast operator as follows –

(type_name) expression

Typecasting allows us to convert one data type into other. In C language, we use cast operator for typecasting which is denoted by (type).

**Syntax:**

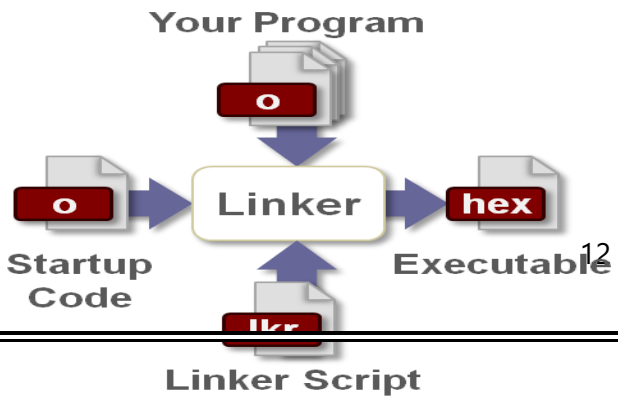
(type)value;

**Example**

```
float f=(float) 9/4;
printf("f : %f\n", f );//Output: 2.250000
```

## Run Time Environment

Runtime environment (RTE) is an environment in which a computer program executes. It provides the program with the resources it needs to



run, such as memory, storage, and input/output.
In C, the RTE is provided by the C compiler. The compiler generates code that initializes the program's memory, loads its data, and executes its instructions.
**When this program is compiled and run, the RTE will perform the following steps:**

1. Initialize the program's memory.
2. Load the program's data into memory.
3. Execute the program's instructions.

**Static and Dynamic Memory Allocation**
In this allocated memory remains from start to end of the program. In this allocated memory can be released at any time during the program. Example: This static memory allocation is generally used for array. Example: This dynamic memory allocation is generally used for linked list.

**C Storage Class**
Every variable in C programming has two properties: type and storage class.
Type refers to the data type of a variable. And, storage class determines the scope, visibility and lifetime of a variable.
There are 4 types of storage class:

1. automatic
2. external
3. static
4. register

**Local Variable**
The variables declared inside a block are **automatic or local** variables. The local variables exist only inside the block in which it is declared.

**Global Variable**
Variables that are declared outside of all functions are known as **external or global** variables. They are accessible from any function inside the program.

**Register Variable**
The register keyword is used to declare register variables. Register variables were supposed to be faster than local variables.

However, modern compilers are very good at code optimization, and there is a rare chance that using register variables will make your program faster.

Unless you are working on embedded systems where you know how to optimize code for the given application, there is no use of register variables.

### Static Variable

A static variable is declared by using the static keyword. For example;
static int i;
The value of a static variable persists until the end of the program.

### Syntax and Logical Errors In C Compilation

Here are some examples of syntax and logical errors in C compilation:

**Syntax errors**

- Missing semicolon (;)
- Missing parenthesis ({})
- Assigning value to a variable without declaring it
- Typographical errors (e.g., spelling mistakes)
- Using reserved words for variable or function names
- Unbalanced braces

**Logical errors**

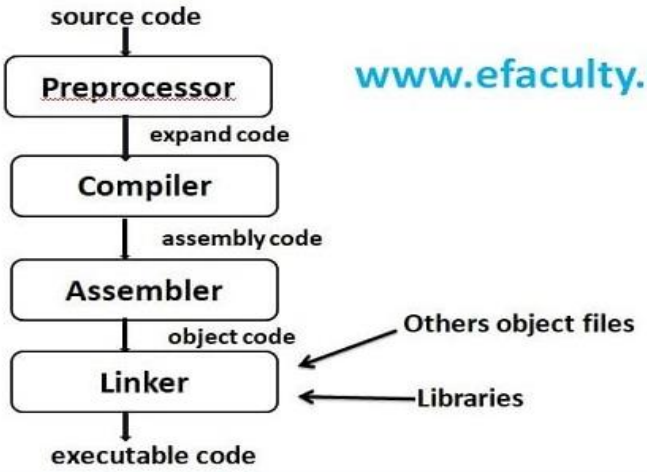
- Dividing by zero
- Trying to access an array index that is out of bounds
- Using the wrong operator
- Infinite loops
- Off-by-one errors

These are just a few examples of the many types of errors that can occur in C compilation. It is important to be familiar with these errors so that you can avoid them and write correct and efficient code.

### Object and Executable Code

Object code is an intermediate product of the compilation process. It is a machine-readable format that can be executed by a computer, but оно is not yet a complete program. An executable file is a final product that can be run directly by a computer. It contains all of the code and resources that are needed to execute the program.

To create an executable file from object



code, you need to use a linker. The linker takes all of the object files and combines them into a single file. It also resolves any unresolved symbols, which are references to functions or variables that are not defined in the object files.