# C Programing:

**Table of Contents:**

**Operations:**

1. Ascii values
2. Binary , Decimal Conversions
3. Arithmetic
4. Assign
5. Comparison
6. Logical
7. Conditional
8. Bitwise
9. Precedence

**Looping & Statements:**

1. If else
2. Switch case
3. Loopings (for,while and do while)
4. Breaking loops
5. Knowledge of goto

**Functions:**

1. Declaration
2. Passing arguments
3. Recursive function
4. Custom function headers
5. Restricting

**Pointers:**

1. Declaration
2. Pointers Arthmetic

9. Genarating random numbers
10. Windows API (WinAPI)
11. Timer set

**Reference: (mostly show in table like format)**

1. ASCII Character codes
2. printf() format specifiers
3. scanf() format specifiers
4. Functions for character input and output
5. String functions
6. Math functions
7. Utility functions
8. Diagnostic functions
9. Argument functions
10. Date and Time Functions in C
11. All type of format specifiers for time func
12. Jump functions
13. Limit & Float constants

[NOTE]

All the source code files are located in git syllabus repository.

All the syntax and table format is located in drawio.com.

# 1.Introduction about C:

## 1.1 History of C Programming Language

### 1.1.1 Origins and Development

Developed by Dennis Ritchie at Bell Labs between 1969 and 1973.

Evolved from earlier languages such as B and BCPL.

Created for system programming and writing operating systems (notably UNIX).

### 1.1.2 History

1972: Introduction of the first version of C.

1978: Publication of "The C Programming Language" by Brian Kernighan and Dennis Ritchie (K&R C).

1989: ANSI C (C89) - Standardization by ANSI.

1990: ISO C (C90) - International standardization.

1999: C99 - Introduction of new features like inline functions, variable-length arrays.

2011: C11 - Enhanced multi-threading support, Unicode support, improved performance.

2018: C18 - Minor bug fixes and improvements to C11.

## 1.2 Why Learn C Programming ?

1.2.1 Foundational Language: C is the basis for many modern languages, making it easier to learn C++, Java, and Python.

1.2.2 Performance and Efficiency: Known for high performance and resource management.

1.2.3 Portability: Code written in C can be compiled on various platforms.

1.2.4 Versatility: Used in a wide range of applications, from operating systems to embedded systems.

1.2.5 Career Opportunities: Proficiency in C is highly valued in many tech fields, offering robust career prospects

## 1.3 Standard C Libraries

1.stdio.h (Standard Input and Output)

Provides functions for input and output operations, including reading from the keyboard, writing to the screen, and file handling (e.g., printf, scanf, fopen, fclose).

2.stdlib.h (Standard Library)

Offers functions for memory allocation, process control, conversions, and others (e.g., malloc, free, exit, atoi).

3.string.h (String Handling)

Contains functions for manipulating arrays of characters, commonly known as strings (e.g., strlen, strcpy, strcat, strcmp).

4.math.h (Mathematical Functions)

Provides functions for performing mathematical operations (e.g., sin, cos, tan, sqrt, pow).

5.time.h (Time and Date)

Includes functions for manipulating and formatting time and date information (e.g., time, ctime, difftime, strftime).

6.ctype.h (Character Type Functions)

Contains functions for testing and mapping characters (e.g., isalpha, isdigit, islower, toupper).

7.limits.h (Limits of Data Types)

Defines the sizes and ranges of basic data types (e.g., INT_MAX, CHAR_MIN, UINT_MAX).

8.float.h (Floating-Point Limits)

Specifies the characteristics of floating-point types, including their precision and range (e.g., FLT_MAX, DBL_MIN).

9.assert.h (Diagnostics)

Provides a macro for adding diagnostics to programs (e.g., assert).

10.errno.h (Error Handling)

Defines macros for reporting and retrieving error conditions through error codes (e.g., errno, EDOM, ERANGE).

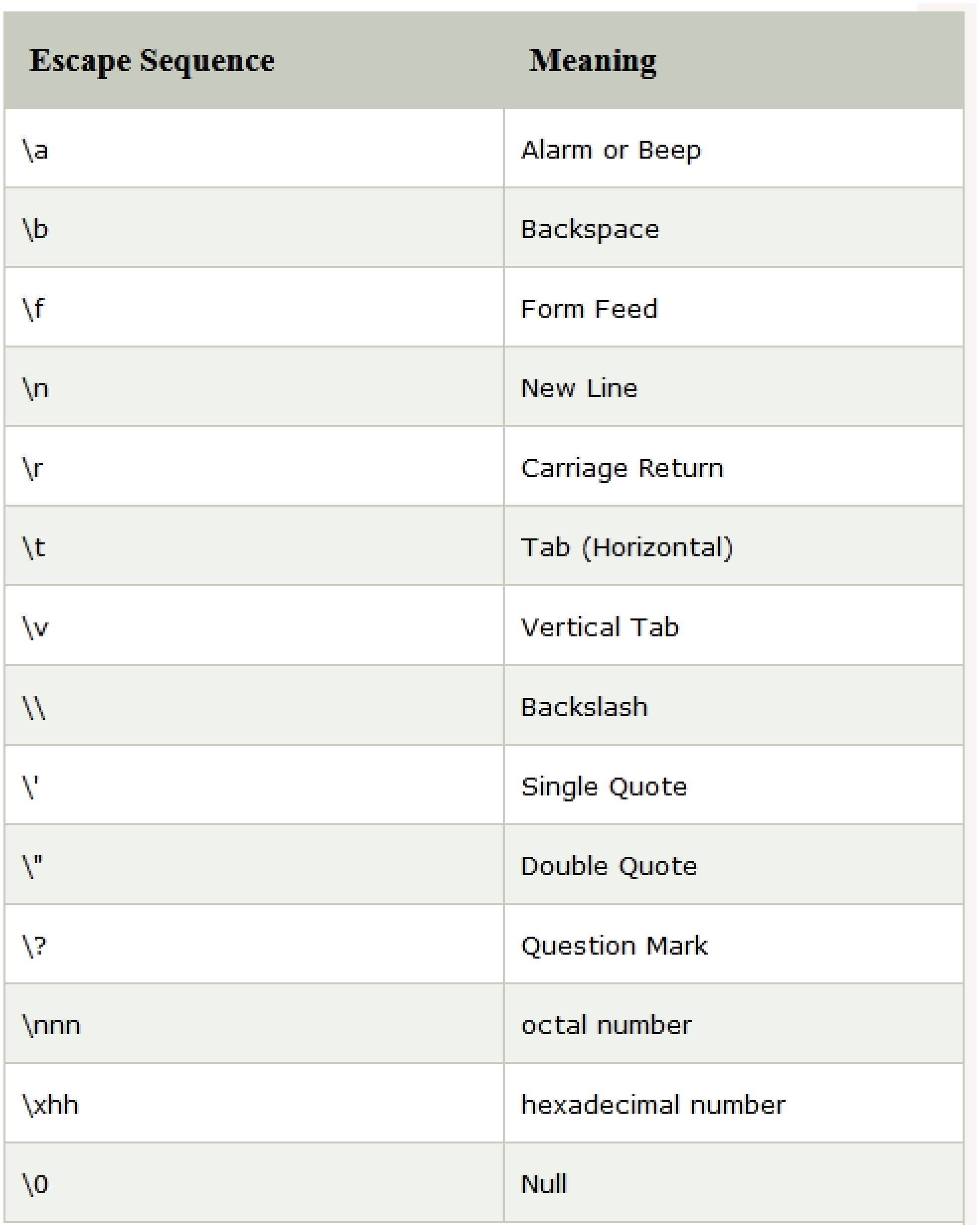
## 1.4 C Keywords

| | | | |
|---|---|---|---|
| auto | double | int | struct |
| break | else | long | switch |
| case | enum | register | typedef |
| char | extern | return | union |
| continue | for | signed | void |
| do | if | static | while |
| default | goto | sizeof | Volatile |
| const | float | short | Unsigned |

| | |
|---|---|
| auto | - Declares automatic local variables |
| break | - Exits from a loop or switch statement |
| case | - Defines a branch in a switch statement |
| char | - Declares a character variable |
| const | - Declares a constant variable |
| continue | - Skips to the next iteration of a loop |
| default | - Defines the default branch in a switch |
| do | - Starts a do-while loop |
| double | - Declares a double-precision floating point |
| else | - Defines an alternative branch in an if |
| enum | - Declares an enumerated type |
| extern | - Declares an external variable or function |
| float | - Declares a floating point variable |
| for | - Starts a for loop |
| goto | - Transfers control to a labeled statement |
| if | - Starts a conditional statement |
| int | - Declares an integer variable |
| long | - Declares a long integer variable |
| register | - Declares a register variable |
| return | - Returns from a function |
| short | - Declares a short integer variable |
| signed | - Declares a signed type variable |

- sizeof - Returns the size of a data type
- static - Declares a static variable
- struct - Declares a structure
- switch - Starts a switch statement
- typedef - Creates a new data type name
- union - Declares a union
- unsigned - Declares an unsigned type variable
- void - Declares a function with no return value
- volatile - Declares a variable that be modified in unexpected ways
- while - Starts a while loop

## 1.5 Escape sequence



| Escape Sequence | Meaning |
|---|---|
| \a | Alarm or Beep |
| \b | Backspace |
| \f | Form Feed |
| \n | New Line |
| \r | Carriage Return |
| \t | Tab (Horizontal) |
| \v | Vertical Tab |
| \\ | Backslash |
| \' | Single Quote |
| \" | Double Quote |
| \? | Question Mark |
| \nnn | octal number |
| \xhh | hexadecimal number |
| \0 | Null |

## 1.6 C compilers and editors

There are around 5+ c editors are available like VS code,GNU,Turbo c,notepad etc..

## 1.7 Folder Creation

1.7.1 Create a folder name called Enn programs in local drive. For the use of our workouts

## 1.8 Syntax

```
// Include necessary header files
#include <headerfile.h>

// Main function, entry point of the program
int main()

{

// Statements inside the main function



// Return statement indicating successful termination

 return 0;

}
```

1.8.1 Get into enn C :

example:

```
// Include necessary header files
#include <stdio.h>

// Main function, entry point of the program
int main()

{
   // Statements inside the main function
   printf("Hello, Enn World!\n");

   // Return statement indicating successful termination
   return 0;

}
```

Save the c program files with .c extensions

Output: Hello, Enn World

**<<sample program file name was hello.c it located in git repository at introduction folder for your reference>>**

## 1.9 Compliation process

**<<The picturized format is located at introduction folder in drawio for your reference>>**

## 2. variables

Variables are containers for storing data values, like numbers and characters.

### 2.1 Variables creation

Syntax:

type variableName = value; (or)  type variable_name;

#### 2.1.2 Rules to declare:

1.Must begin with a letter (uppercase or lowercase) or an underscore (_).

2.Subsequent characters may be letters, digits, or underscores.

3.Case-sensitive.

4.Cannot use reserved keywords in C as variable names.

5.Cannot contain spaces or special characters other than the underscore (_).

6.No specific maximum length, but keep names reasonably short and meaningful.

### 2.2 Data Types

**<<The picturized table format for datatypes is located  at variables folder in drawio  for your reference>>**

Example :

int num = 6;

```
char a;
```

### 2.2.2 Format Specifiers

**<<The picturized table format for Format Specifiers is located at variables folder in drawio for your reference>>**

#### 2.2.2.1 Explanation:

%d: Prints the integer value 10.

%ld: Prints the long integer value 1000000.

%f: Prints the float value 3.14159.

%c: Prints the character 'A'.

%s: Prints the string "Hello".

%p: Prints the memory address, typically in hexadecimal format (e.g., 0x0).

## 2.3 Input function

### 2.3.1 Input Function scanf() in C:

The scanf() function in C is used to read input from the standard input (keyboard by default) and store it in variables. It is typically used with format specifiers to specify the type of data being read.

Syntax:

```
scanf("format_specifiers", &variable1, &variable2, ...);
```

Example: var.c

in var.c file explained the topic programmly , you to compile the program by compiler like, gcc

```
Integer: 3
Float: 3.500000
Character: e
String: ennbook
Memory address of num1 = 0x7ffffe385efc
```

## 2.4 Scope of variables

### 2.4.1 Global Variables in C

Global variables in C are declared outside of any function and have a scope that extends throughout the entire program. They are accessible from any part of the program, including all functions, and retain their values until the program terminates.

### 2.4.2 Local Variables in C

Local variables in C are declared within a block or function and have a limited scope, existing only within the block in which they are defined. Once the block ends, local variables are destroyed, and their memory space is freed.

## 2.5 static Keyword

Definition:

Static Global Variable: When static is used with a global variable declaration, it limits the scope of the variable to the file in which it is declared. It retains its value between function calls. Static avoids the need to rename one of the variables in each occurrence within the source code.

## 2.6 extern Keyword

Definition:

Extern Global Variable: The extern keyword is used to declare a global variable that is defined in another file. It allows the variable to be accessed from multiple files within a program.

**<<sample program file name was scope1.c & scope2.c for external variable it located in git repository at introduction folder for your reference>>**

output:

```
Global Variable (main.c): 100
Static Global Variable (main.c): 50
Extern Global Variable (main.c): 200
Local Variable (func): 20
```

## 2.7 Type Casting in C

Type casting in C refers to explicitly converting a variable from one data type to another. It allows you to change the interpretation of data temporarily or permanently.

2.7.1 Usage:

1.Implicit Type Conversion:

Automatically performed by the compiler when mixing different data types in expressions, following rules of promotion (e.g., int to float).

2.Explicit Type Casting:

Manually specified by the programmer using casting operators to convert data types explicitly.

**<<The detailed syntax for Type Conversion is located at variables folder in drawio for your reference>>**

**<<sample program file name was type.c it located in git repository at variables folder for your reference>>**

output:

```
Implicit Conversion - char to int: 65
Explicit Conversion - int to char: A
Implicit Conversion - int to float: 65.000000
Explicit Conversion - float to int: 3
```

2.8 Arrays

Arrays in C are a collection of elements of the same data type stored in contiguous memory locations. They provide a convenient way to store and access multiple values of the same type under a single name.

## 2.8.1 Characteristics:

1.Fixed Size:

Arrays have a fixed size that must be specified when they are declared, and this size cannot be changed during runtime.

2.Zero-based Indexing:

Elements in an array are accessed using an index, starting from 0 for the first element up to size - 1 for the last element of an array of size size.

3.Contiguous Memory Allocation:

Elements of an array are stored in consecutive memory locations, which allows for efficient memory access.

## 2.8.2 Types of Array:

### 2.8.2.1 Single Dimensional Array

Declaration and Initialization:

```
dataType arrayName[arraySize];
```

1.dataType: Type of elements in the array (int, float, char, etc.).

2.arrayName: Name of the array.

3.arraySize: Number of elements in the array.

**<<sample program file name was array1.c it located in git repository at variables folder for your reference>>**

output:

```
First element: 1

Second element: 2

Third element: 3

Modified third element: 10
```

Explanation:

Array Declaration: int numbers[5]; declares an array numbers of integers with size 5.

Array Initialization: int numbers[5] = {1, 2, 3, 4, 5}; initializes the array numbers with values {1, 2, 3, 4, 5}.

Accessing Elements: Elements of the array are accessed using square brackets [] and the index of the element (e.g., numbers[0] accesses the first element).

Modifying Elements: Array elements can be modified using their indices (e.g., numbers[2] = 10; modifies the third element to 10).

2.8.2.2 Multi Dimensional Array

Multidimensional arrays in C are arrays that have more than one dimension, typically represented as a matrix or a table of elements. They allow storing data in multiple rows and columns, providing a structured way to organize and access data.

Declaration and Initialization:

```
dataType arrayName[rowSize][colSize];
```

1.dataType: Type of elements in the array (int, float, char, etc.).

2.arrayName: Name of the array.

3.rowSize: Number of rows in the array.

4.colSize: Number of columns in the array.

**<<sample program file name was array2.c it located in git repository at variables folder for your reference>>**

output:

```
Element at matrix[0][0]: 1

Element at matrix[0][1]: 2

Element at matrix[1][0]: 3

Element at matrix[1][1]: 4

Modified element at matrix[1][1]: 10
```

Explanation:

Array Declaration: int matrix[2][2]; declares a 2x2 array matrix of integers.

Array Initialization: int matrix[2][2] = {{1, 2}, {3, 4}}; initializes the array matrix with values { {1, 2}, {3, 4} }.

Accessing Elements: Elements of the array are accessed using two indices: row index and column index (e.g., matrix[0][0] accesses the element in the first row and first column).

Modifying Elements: Array elements can be modified using their row and column indices (e.g., matrix[1][1] = 10; modifies the element in the second row and second column to 10).

## 3 Constants in C

In C programming, constants are fixed values that do not change during the execution of a program. They are used to represent values that are known and fixed at compile-time. Constants can be of various types, such as integer constants, floating-point constants, character constants, and string literals.

### 3.1 Declaration

By Using const Keyword we have to declare the constant. Try to indicate the constant variable in uppercase, that follows trationally.

**<<The detailed syntax for Constants is located  at Constants folder in drawio  for your reference>>**

**<<sample program file name was const.c it located in git repository at Constants folder for your reference>>**

### 3.2 Enumerations (Enums) in C

Enumerations, often referred to as enums, in C are user-defined data types used to assign names to integral constants.The enum keyword provides a handy way to create a sequence of constants in concise manner.

**<<The detailed syntax for Enumerations is located  at Constants folder in drawio  for your reference>>**

**<<sample program file name was enum.c it located in git repository at Constants folder for your reference>>**

## 3.2.2 Enum Variables

Enum variables are variables declared with an enum type. They can store any of the enum constants defined within their enumeration type.

**<<The detailed syntax for Enum Variables is located  at Constants folder in drawio  for your reference>>**

**<<sample program file name was enumvar.c  it located in git repository at Constants folder for your reference>>**

Explanation:

Enum Declaration: enum Weekday { MONDAY, TUESDAY, ... }; defines an enum Weekday with constants representing days of the week.

Enum Variable: enum Weekday today = TUESDAY; declares today as a variable of type enum Weekday initialized to TUESDAY.

Switch Case: Uses today in a switch statement to print the current day based on the enum constant.

1.Enum Constants: Provide symbolic names to integral values within an enum.

2.Enum Variables: Store values from the set of enum constants defined within their enum type

## 3.3 Define directive

The #define directive in C is a preprocessor directive used to define symbolic constants and macros. It instructs the preprocessor to replace all instances of the defined identifier with the specified token sequence throughout the code before compilation.

**<<The detailed syntax for Define directive is located at Constants folder in drawio for your reference>>**

**<<sample program file name was def.c it located in git repository at Constants folder for your reference>>**

## 3.4 #ifdef, #endif Directives & Debuging

The #ifdef and #endif directives are used for conditional compilation in C. They check if a macro is defined and control the inclusion of code based on the result.

**<<The detailed syntax for Debuging is located at Constants folder in drawio for your reference>>**

**<<sample program file name was debug.c it located in git repository at Constants folder for your reference>>**

Explanation:

#ifdef DEBUG: Checks if the macro DEBUG is defined.

printf("Debugging information\n"); is included in the program because DEBUG is defined (#define DEBUG 1).

printf("Program continues...\n"); is always included regardless of the DEBUG macro.

# 4 Operations

## 4.1 Operations

## 4.2 Binary and Decimal Conversions in C

### 4.2.1 Introduction to Number Systems

Binary System:

A base-2 number system that uses two symbols, typically 0 and 1, to represent values. Each digit in a binary number is a power of 2.

Decimal System:

A base-10 number system that uses ten symbols (0-9) to represent values. Each digit in a decimal number is a power of 10.

### 4.2.2 Binary to decimal conversion:

Example:

The binary sequence (1101)2 has the decimal equivalent

(1101)2 = 1 × 23 + 1 × 22 + 0 × 21 + 1 × 20

= 8 + 4 + 0 + 1

= (13)10

### 4.2.3 Decimal to binary conversion:

To convert Decimal to Binary "Repeated Division by 2" method can be used. Any Decimal number divided by 2 will leave a remainder of 0 or 1. Repeated division by 2 will leave a sequence of 0s and 1s that become the binary equivalent of the decimal number.

Example

Convert (65)10 into its equivalent binary number

**<<The detailed example for Decimal to binary Conversion is located at Operations folder in drawio for your reference>>**

## 4.3 Operators in C

Operators in C are special symbols or keywords that are used to perform operations on variables and values. C language supports a rich set of built-in operators that can be categorized based on the type of operation they perform.

1. Arithmetic Operators

   These operators are used to perform basic arithmetic operations like addition, subtraction, multiplication, division, and modulus.

**<<The picturized table format for Arithmetic Operators is located at Operations folder in drawio for your reference>>**

**<<sample program file name was arith.c it located in git repository at Operations folder for your reference>>**

2. Relational Operators

Relational operators are used to compare two values. They return either true or false based on the comparison.

**<<The picturized table format for Relational Operators is located at Operations folder in drawio for your reference>>**

**<<sample program file name was relation.c it located in git repository at Operations folder for your reference>>**

3. Logical Operators

Logical operators are used to perform logical operations on boolean values.

**<<The picturized table format for Logical Operators is located at Operations folder in drawio for your reference>>**

**<<sample program file name was logical.c it located in git repository at Operations folder for your reference>>**

Explanation:

Logical AND (&&): A && B is true only if both A and B are true.

Logical OR (||): A || B is true if either A or B (or both) are true.

Logical NOT (!): !A is true if A is false, and !A is false if A is true.

4. Assignment Operators

Assignment operators are used to assign values to variables.

**<<The picturized table format for . Assignment Operators is located at Operations folder in drawio for your reference>>**

**<<sample program file name was assign.c it located in git repository at Operations folder for your reference>>**

5. Ternary Operator

The ternary operator is a shorthand for if-else statements.

**<<The picturized table format for Ternary Operator ors is located at Operations folder in drawio for your reference>>**

**<<sample program file name was ternary.c it located in git repository at Operations folder for your reference>>**

6. Unary Operators

Unary operators operate on a single operand.

**<<The picturized table format for Unary Operators is located at Operations folder in drawio for your reference>>**

**<<sample program file name was unary.c it located in git repository at Operations folder for your reference>>**

7. Bitwise Operators

Bitwise operators are used to perform bit-level operations on integers.Bitwise operators manipulate bits of operands at the binary level. They are used to perform operations on individual bits of integer operands.

1. Bitwise AND (&)

Definition: Performs a bitwise AND operation between corresponding bits of two operands. The result is 1 only if both bits are 1.

Example: a & b

2. Bitwise OR (|)

Definition: Performs a bitwise OR operation between corresponding bits of two operands. The result is 1 if at least one of the bits is 1.

Example: a | b

3. Bitwise XOR (^)

Definition: Performs a bitwise XOR (exclusive OR) operation between corresponding bits of two operands. The result is 1 if the bits are different.

Example: a ^ b

4. Bitwise NOT (~)

Definition: Performs a bitwise NOT operation on each bit of the operand, flipping all bits (0s become 1s and 1s become 0s).

Example: ~a

5. Left Shift (<<)

Definition: Shifts the bits of the left operand left by the number of positions specified by the right operand. Zeros are shifted into the low-order bits.

Example: a << b

6. Right Shift (>>)

Definition: Shifts the bits of the left operand right by the number of positions specified by the right operand. For unsigned types, zeros are shifted into the high-order bits. For signed types, the result depends on the implementation (typically, sign extension).

Example: a >> b

**<<The picturized table format for Bitwise Operators is located at Operations folder in drawio for your reference>>**

**<<sample program file name was bit.c it located in git repository at Operations folder for your reference>>**

8. Special Operators

Special operators include sizeof, comma, and member access operators.

**<<The picturized table format for Special Operators is located at Operations folder in drawio for your reference>>**

**<<sample program file name was special.c it located in git repository at Operations folder for your reference>>**

## 4.4 Operator Precedence in C

Operator precedence determines the order in which operators are evaluated in expressions. Operators with higher precedence are evaluated before operators with lower precedence. When operators have the same precedence, their associativity (left-to-right or right-to-left) determines the order of evaluation.

**<<The picturized table format for Operator Precedence is located at Operations folder in drawio for your reference>>**

**<<sample program file name was precedence.c it located in git repository at Operations folder for your reference>>**

# 5 Looping and Statements

## 5.1 Introduction to Control Structures

Definition and Importance:

1.Control structures are fundamental to programming as they dictate the flow of a program.

2.Decision-making is crucial for creating dynamic and responsive programs.

3.The if statement is a fundamental tool for making decisions in C programming.

4.It allows the execution of specific code blocks based on whether a condition is true or false.

5.Proper use of if statements, including understanding condition expressions, logical operators, and nested structures, is crucial for writing effective and readable C programs.

The if Statement

Basic Usage:

The if statement allows the program to execute a block of code only if a specified condition is true.

Condition Expressions:

Conditions typically involve comparison operators (==, !=, >, <, >=, <=).

**<<The detailed syntax for if Statement is located at making statements folder in drawio for your reference>>**

**<<sample program file name was if.c it located in git repository at looping statements folder for your reference>>**

Explanation:

Start: The program begins execution.

Variable Initialization: x is set to 10.

Condition Check: The if statement checks if x is greater than 5.

Condition: $x > 5 \rightarrow 10 > 5 \rightarrow$ true

True Branch: Since the condition is true, the program executes the code inside the if block.

Output: The message "x is greater than 5" is printed to the console.

End: The program finishes execution after printing the message.

5.2 if else

The if-else Statement

Basic Usage:

The if-else statement allows the program to choose between two paths of execution based on a condition.

if condition is true means if block executed orelse condition false means else block executed.

**<<The detailed syntax for if-else Statement is located  at making statements folder in drawio  for your reference>>**

**<<sample program file name was ifelse.c it located in git repository at looping statements folder for your reference>>**

Explanation:

Start: The program begins execution.

Variable Initialization: age is set to 18.

Condition Check: The if statement checks if age is greater than or equal to 18.

Condition: age >= 18 → 18 >= 18 → true

True Branch Execution: Since the condition is true, the program executes the code inside the if block.

Output: The message "You are eligible to vote." is printed to the console.

False Branch Skipped: The else block is skipped because the condition was true.

End: The program finishes execution after printing the message.

5.3 if else if

The if-else if Statement

Basic Usage:

The if-else if statement allows the program to choose between multiple paths of execution based on multiple conditions.

if condition is true means if block executed orelse checks one more condition which declarte in else if block ,is that condition true means else if block executed,like way its goes on .is any condition is satisfied means else block executed.

**<<The detailed syntax for if-else if Statement is located  at making statements folder in drawio  for your reference>>**

**<<sample program file name was ifelseif.c it located in git repository at looping statements folder for your reference>>**

Explanation:

Start: The program begins execution.

Variable Initialization: x is set to 10.

First Condition Check: if (x > 5) → 10 > 5 → true.

Executes the code block inside the if statement.

Outputs: "x is greater than 5".

Other Conditions:

The else if (x == 5) and else blocks are skipped because the first condition was true.

End: The program finishes execution after printing the message.

5.4 Nested if

Nested if-else Statements

Basic Usage:

Nested if-else statements involve placing one if-else statement inside another.

This allows for multiple levels of conditions to be checked.

**<<The detailed syntax for Nested if is located at making statements folder in drawio for your reference>>**

**<<sample program file name was nestedif.c it located in git repository at looping statements folder for your reference>>**

Start: The program begins execution.

Variable Initialization: x is set to 10.

Outer Condition Check: if (x > 5) → 10 > 5 → true.

Executes the code block inside the outer if statement.

Inner Condition Check: if (x == 10) → 10 == 10 → true.

Executes the code block inside the inner if statement.

5.5 Switch case

Switch-Case Statement:

1 Expression:

The switch-case statement begins with an expression (often an integer or a character) whose value is evaluated against various case labels.

2 Case Labels:

Each case label specifies a constant value or a range of values that the expression might match.

3 Code Blocks:

When a match is found between the expression and a case label, the corresponding block of code associated with that case is executed.

4 Default Case (Optional):

The default case is optional and provides a block of code to execute when none of the case labels match the value of the expression.

5 Break Statement:

After executing a block of code associated with a case label, the break statement is used to exit the switch-case statement. Without a break, execution will continue to the next case (known as fall-through).

**<<The detailed syntax for Switch-Case Statement: is located  at making statements folder in drawio  for your reference>>**

**<<sample program file name was switch.c it located in git repository at looping statements folder for your reference>>**

Explanation:

Expression: day is evaluated in the switch statement.

Case Labels: Each case label (1, 2, 3, 4, 5) represents a possible value of day.

Code Blocks: Depending on the value of day, the corresponding weekday name is printed.

Default Case: If day does not match any of the specified case labels (1 to 5), the default case prints "Weekend".

Break Statement: Each case block ends with a break statement to exit the switch statement after executing the corresponding block of code.

## 5.6 for loop

In programming, specifically in languages like C, C++, Java, and others, a for loop is a control flow statement that allows repeated execution of a block of statements based on a specified condition. It is particularly useful when the number of iterations (repetitions) is known beforehand or can be determined during runtime.

**<<The detailed syntax for for loop Statement is located  at making statements folder in drawio  for your reference>>**

**<<sample program file name was for.c it located in git repository at looping statements folder for your reference>>**

Start: The program begins execution.

Variable Initialization: i is initialized to 1.

For Loop Execution:

Iterates from 1 to 5.

Prints each value of i followed by a space (1 2 3 4 5 ).

Newline Output: After the loop completes, prints a newline character (\n).

End: The program returns 0, indicating successful execution.

## 5.7 Nested for loop

In programming, a nested for loop refers to the situation where one for loop is nested inside another. This construct allows for repetitive execution of a block of statements within another block of statements. Nested loops are often used to iterate over multi-dimensional arrays or perform operations that require repeated actions within other repeated actions.

Key Components of Nested For Loop:

Outer Loop:

The outer loop encloses the inner loop and controls how many times the inner loop will execute.

It typically initializes and updates a loop control variable that determines the number of iterations.

Inner Loop:

The inner loop is contained within the outer loop and executes its statements repeatedly for each iteration of the outer loop.

It has its own initialization, condition, and update statements.

<<The detailed syntax for Nested for loop is located at making statements folder in drawio for your reference>>

<<sample program file name was nestedfor.c it located in git repository at looping statements folder for your reference>>

Explanation:

Outer Loop Initialization: i is initialized to 1.

Outer Loop Condition Check: Checks if i <= 3 (true).

Inner Loop Initialization: j is initialized to 1.

Inner Loop Condition Check: Checks if j <= 3 (true).

Inner Loop Execution: Executes printf("%d ", i * j); (outputs 1 2 3 for i = 1).

Inner Loop Update: j increments to 2.

Repeat Inner Loop Execution: Outputs 2 4 6 for i = 1.

Inner Loop Update and Repeat: Outputs 3 6 9 for i = 1.

Outer Loop Update: i increments to 2.

Repeat Steps 3-9 for i = 2 and i = 3.

Complete Execution: Outputs the final result:

5.7 while loop

In programming, a while loop is a control flow statement that allows a block of code to be executed repeatedly based on a specified condition. It continues to execute the block as long as the condition remains true.

Key Components of While Loop:

Condition:

An expression that is evaluated before each iteration of the loop.

If the condition evaluates to true (true or a non-zero value), the loop body is executed.

If false (false or zero), the loop terminates.

**<<The detailed syntax for while loop is located at making statements folder in drawio for your reference>>**

**<<sample program file name was while.c it located in git repository at looping statements folder for your reference>>**

Explanation:

Initialization: i is initialized to 1.

Condition Check: Checks if i <= 5 (true).

Execute Loop Body: Prints 1 (outputs 1), increments i to 2.

Condition Check: Checks if i <= 5 (true).

Execute Loop Body: Prints 2 (outputs 2), increments i to 3.

Repeat Steps 4-5: Continues executing the loop body until i reaches 6.

Condition Check: Checks if i <= 5 (false, terminates the loop).

End: Program exits the loop and proceeds to execute printf("\n");.

Output: Outputs 1 2 3 4 5 followed by a newline (\n).

## 5.8 Do while loop

Definition of Do-While Loop:

In programming, a do-while loop is a control flow statement that executes a block of code at least once, and then repeatedly executes the block based on a specified condition. Unlike the while loop, which tests the condition before entering the loop, the do-while loop tests the condition after executing the loop body.

Key Components of Do-While Loop:

Execution of Loop Body:

The block of statements inside the loop is executed once first, regardless of the condition.

Condition:

An expression that is evaluated after each execution of the loop body.

If the condition evaluates to true (true or a non-zero value), the loop continues executing.

If false (false or zero), the loop terminates.

**<<The detailed syntax for do while loop  is located  at making statements folder in drawio for your reference>>**

**<<sample program file name was dowhile.c it located in git repository at looping statements folder for your reference>>**

Explanation:

Initialization: i is initialized to 1.

Execute Loop Body: Prints 1 (outputs 1), increments i to 2.

Condition Check: Checks if i <= 5 (true).

Repeat Steps 2-3: Continues executing the loop body until i reaches 6.

Condition Check: Checks if i <= 5 (false, terminates the loop).

End: Program exits the loop and proceeds to execute printf("\n");.

Output: Outputs 1 2 3 4 5 followed by a newline (\n).

key theoretical differences between while and do-while loops:

Execution Order:

While Loop: In a while loop, the condition is evaluated before the execution of the loop body. If the condition is initially false, the loop body may never execute.

Do-While Loop: In a do-while loop, the loop body is executed at least once before the condition is evaluated. This ensures that the loop body executes at least once, regardless of the condition.

.9 Break, continue and goto

5.9.1 Break Statement

Definition:

The break statement is a control flow statement in programming languages like C, C++, and others. It is used to immediately terminate the execution of the innermost loop (for, while, or do-while) or switch statement in which it appears. When encountered, break causes control to exit the loop or switch statement and resume execution at the next statement after the loop or switch.

**<<The detailed syntax for Break Statement  is located  at making statements folder in drawio  for your reference>>**

5.9.2 Continue Statement

Definition:

The continue statement is a control flow statement used to skip the current iteration of a loop (for, while, or do-while) and proceed to the next iteration. It causes the loop to immediately jump to the next iteration's condition check or loop body execution, bypassing any remaining code within the current iteration.

**<<The detailed syntax for Continue Statement is located  at making statements folder in drawio  for your reference>>**

## 5.9.3 Goto Statement

Definition:

The goto statement is a control flow statement in C that allows transferring control to a specified label within the same function or within the same file scope. It provides an unconditional jump to a specified statement, skipping any code in between. The use of goto is generally discouraged in modern programming practices due to its potential to create complex and hard-to-maintain code.

**<<The detailed syntax for Goto Statement is located  at making statements folder in drawio for your reference>>**

**<<sample program file name was pass.c it located in git repository at looping statements folder for your reference>>**

Explanation:

Using break statement:

Purpose: Exit the loop immediately when i equals 3.

Output: Prints numbers from 1 to 2.

Flow of Execution:

Starts the loop with i = 1.

Prints 1.

Increments i to 2, prints 2.

Encounters break;, exits the loop.

Using continue statement:

Purpose: Skip printing 3 and continue with the next iteration.

Output: Prints numbers from 1 to 5, excluding 3.

Flow of Execution:

Starts the loop with i = 1.

Prints 1.

Increments i to 2, prints 2.

Encounters continue; when i = 3, skips printing 3.

Continues with i = 4, prints 4.

Continues with i = 5, prints 5.

Using goto statement:

Purpose: Implement a loop using goto statement to jump back to the label start.

Output: Prints numbers from 1 to 5.

Flow of Execution:

Initializes j = 1.

Prints 1.

Increments j to 2, prints 2.

Continues until j is 6.

Prints newline (\n) and exits the loop.

Break: Terminates the loop immediately when a condition (i == 3) is met.

Continue: Skips the remaining code in the current iteration (i == 3) and continues with the next iteration.

Goto: Jumps to a specified label (start) unconditionally, allowing for non-sequential execution within the loop.

# 6 Functions

## 6.1 Define Function

A function is a block of code that performs a specific task. It is designed to be reusable, so once a function is written, it can be called multiple times from different parts of a program. Functions help in organizing code into manageable sections, making it more readable, maintainable, and easier to debug.

## 6.2 Why Use Functions

Modularity: Functions break down complex problems into simpler parts.

Reusability: Once a function is written, it can be reused across multiple programs.

Maintainability: Functions make it easier to update and maintain the code.

Abstraction: Functions allow for higher-level programming by hiding implementation details.

## 6.3 Function Declaration

Function declaration, also known as a function prototype, tells the compiler about the function's name, return type, and parameters. It is usually placed before the main function or in a header file.

it has two types of return type available in function,

1.void

2.int

### 6.3.1 Void type function

#### 6.3.1.1 Function Declaration

Function declaration, also known as a function prototype, tells the compiler about the function's name, return type, and parameters. It is usually placed before the main function or in a header file.

**<<The detailed syntax for Void type function Declaration is located at functions folder in drawio for your reference>>**

#### 6.3.1.2 Function Definition

A function definition provides the actual body of the function. It includes the code that specifies what the function does.

**<<The detailed syntax for Void type function Definition is located  at functions  folder in drawio  for your reference>>**

6.3.2 int type function

6.3.2.1 Function Declaration

Function declaration, also known as a function prototype, tells the compiler about the function's name, return type, and parameters. It is usually placed before the main function or in a header file.

**<<The detailed syntax for int type function Declaration is located  at functions  folder in drawio  for your reference>>**

6.3.1.2 Function Definition

A function definition provides the actual body of the function. It includes the code that specifies what the function does.

**<<The detailed syntax for int type function Definition is located  at functions  folder in drawio  for your reference>>**

**<<sample program file name was func.c it located in git repository at functions  folder for your reference>>**

Explanation:

1. Start main Function Execution:

Execution begins in the main function.

2.Call greet Function:

main calls greet().

Execution moves to the greet function.

3.Inside greet Function:

Print Hello, World!\n to the console.

Return from greet to main.

4.Call square Function:

main calls square().

Execution moves to the square function.

5.Inside square Function:

Compute 5 * 5 and store the result (25) in the variable s.

Return the value 25 from square to main.

6.Store Return Value:

Back in main, store the returned value (25) in the variable num.

7.Print Statement:

Print Hello, World! followed by the value of num (which is 25) to the console.

8.Return from main:

Return 0 from main, indicating successful completion of the program.

## 6.4 Passing Arguments

### 6.4.1 Introduction to Function Arguments

Function arguments, also known as parameters, allow you to pass data to functions. They enable functions to operate on different data without changing their code, promoting reusability and modularity.

### 6.4.2 Parameter Passing Techniques

1.Call by Value

2.Call by Reference

#### 6.4.2.1 Call by Value

In call by value, a copy of the actual parameter's value is passed to the function. Changes made to the parameter inside the function do not affect the actual parameter.

**<<The detailed syntax for Call by Value is located at functions folder in drawio for your reference>>**

**<<sample program file name was passval.c it located in git repository at introduction folder for your reference>>**

Explanation:

Start main function execution.

Initialize num with the value 5.

Call addTen with num as an argument.

Inside addTen, the parameter n is a copy of num (value 5).

Modify n to 15 (n + 10).

Print Value inside function: 15.

Return from addTen to main.

Print Value of num after function call: 5.

Return 0 from main.

### 6.4.2.2 Call by Reference

In call by reference, a reference (address) to the actual parameter is passed to the function. Changes made to the parameter inside the function affect the actual parameter.

**<<The detailed syntax for Call by Reference is located at functions folder in drawio for your reference>>**

**<<sample program file name was passref.c it located in git repository at introduction folder for your reference>>**

Explanation:

Start main function execution.

Initialize num with the value 5.

Call addTen with the address of num as an argument (&num).

Inside addTen, the parameter n is a pointer to num.

Dereference n and modify the value to 15 (*n + 10).

Print Value inside function: 15.

Return from addTen to main.

Print Value of num after function call: 15.

Return 0 from main.

## 6.4 Recursive Function

Recursion is a powerful concept in programming where a function calls itself directly or indirectly to solve a problem. It allows problems to be solved in simpler terms, often leading to elegant and compact code solutions.

### 6.4.1 Recursive Function Structure

A recursive function typically consists of two parts:

Base Case: A condition that terminates the recursion.

Recursive Case: A condition where the function calls itself with modified arguments.

**<<The detailed syntax for Recursive Function is located at functions folder in drawio for your reference>>**

**<<sample program file name was funrec.c it located in git repository at introduction folder for your reference>>**

Explanation:

Start main function execution.

Initialize num with the value 5.

Call factorial function with num as an argument.

Inside factorial function:

Check if n is 0 or 1 (base case).

Otherwise, compute n * factorial(n - 1) (recursive case).

Recursively call factorial with n - 1 until base case is reached.

Return the computed factorial value back to the caller.

Print the result in main.

## 6.5 Custom function headers

Introduction to Custom Header Files

Custom header files in C allow you to organize your code by placing function declarations, definitions, and other declarations in a separate file. This promotes modularity and reusability of code across multiple source files.

### 6.5.1 Structure of a Custom Header File

A custom header file typically contains:

Function declarations

Type definitions (structs, enums, etc.)

**<<The detailed syntax for Custom function headers is located  at functions folder in drawio for your reference>>**

**<<sample program file name was custom.c , header.h it located in git repository at introduction folder for your reference>>**

Explanation:

Start main function execution.

Include the header.h file, which contains the declaration & defination of the SquareArea function.

Define the side variable with a value of 5.0.

Call the calculateSquareArea function with side as an argument.

Inside calculateSquareArea, compute the area of the square (side * side).

Return the computed area to main.

Print the area of the square using printf.

End of the main function with return 0.

## 6.6 Function Restriction

In C programming, the static keyword can be used to restrict the visibility and accessibility of functions to the current source file. This can be useful for encapsulating functionality that should only be accessed within a specific module or file.

**<<The detailed syntax for Function Restriction is located at functions folder in drawio for your reference>>**

**<<sample program file name was static.c it located in git repository at introduction folder for your reference>>**

Explanation:

Function Declaration (static void restrictedFunction(int secretCode)):

Declares a static function named restrictedFunction that takes an int parameter secretCode.

The static keyword restricts the visibility of restrictedFunction to the current source file (main.c in this case), preventing it from being accessed from other files.

Function Definition ({ ... }):

Inside restrictedFunction, it checks if secretCode equals 1234.

If true, it prints "Access granted! Performing restricted operation..." and may perform additional restricted operations.

If false, it prints "Access denied!" indicating unsuccessful access.

Main Function (int main() { ... }):

Defines a variable code with the value 1234, which simulates a secret code for accessing restrictedFunction.

Calls restrictedFunction(code) to invoke the function and pass code as an argument.

Output:

Depending on the value of code

# 7 Pointers

## 7.1 Definition:

A pointer is a variable that stores the memory address of another variable. Pointers are used to directly access and manipulate the memory address of other variables, enabling efficient array and string manipulation, dynamic memory allocation, and implementation of data structures like linked lists and trees.

## 7.2 Why Use Pointers:

Dynamic Memory Allocation:

Pointers are essential for dynamic memory management

Low-Level Memory Access:

Pointers provide the ability to perform low-level memory manipulation

## 7.3 Define Pointer Variables

Declaration:

A pointer variable is declared using the asterisk (*) symbol before the variable name. The type of pointer determines the type of data it can point to.

**<<The detailed syntax for Pointer Variables is located at pointers folder in drawio for your reference>>**

Initialization:

A pointer can be initialized to the address of an existing variable using the address-of operator (&). The address-of operator (&) is used to obtain the memory address of a variable.

**<<The detailed syntax for Pointer Initialization: is located at pointers folder in drawio for your reference>>**

**<<sample program file name was ptr.c it located in git repository at pointers folder for your reference>>**

Explanation:

1.Start of main Function:

Execution begins at the main function.

2.Variable Declaration and Initialization:

int x = 10;

An integer variable x is declared and initialized to 10.

int *ptr = &x;

A pointer variable ptr is declared and initialized with the address of x.

3.Print the Value of the Regular Variable:

printf("Value of x: %d\n", x);

The printf function is called to print the value of x.

Output: Value of x: 10.

4.Print the Storage Address of the Regular Variable:

printf("Address of x: %p\n", (void*)&x);

The printf function is called to print the address of x.

The &x expression gets the address of x.

(void*)&x casts the address to void* for proper format.

Output: Address of x: <address> (the actual address value will vary).

5.Print the Address Contained in the Pointer Variable:

printf("Address stored in ptr: %p\n", (void*)ptr);

The printf function is called to print the address stored in ptr.

ptr contains the address of x.

(void*)ptr casts the address to void* for proper format.

Output: Address stored in ptr: <address> (should be the same as the address of x).

6.Print the Value Contained at the Address Stored in the Pointer Variable:

```
printf("Value pointed to by ptr: %d\n", *ptr);
```

The printf function is called to print the value at the address stored in ptr.

The *ptr expression dereferences the pointer to get the value stored at that address.

Since ptr points to x, *ptr yields the value of x, which is 10.

Output: Value pointed to by ptr: 10.

7.End of main Function:

The main function returns 0.

Program execution ends.

## 7.4 Pointer Arithmetic in C

Pointer arithmetic is the process of performing operations such as addition and subtraction on pointer variables. Understanding pointer arithmetic is crucial for efficient array and memory manipulation in C. Here's the content for teaching pointer arithmetic:

### Introduction to Pointer Arithmetic

Pointer arithmetic involves the following operations:

1.Incrementing and Decrementing Pointers

2.Adding and Subtracting Integers to/from Pointers

3.Subtracting Two Pointers

### 7.4.1 Incrementing a Pointer:

When you increment a pointer, it points to the next element of its type in memory.The amount by which the pointer is incremented depends on the size of the data type it points to.

### 7.4.2 Decrementing a Pointer:

When you decrement a pointer, it points to the previous element of its type in memory.

### 7.4.3 Adding an Integer to a Pointer:

Adding an integer to a pointer advances the pointer by that number of elements.

### 7.4.4 Subtracting an Integer from a Pointer:

Subtracting an integer from a pointer moves the pointer back by that number of elements.

### 7.4.5 Subtracting Two Pointers:

Subtracting two pointers gives the number of elements between the two addresses.

**<<The detailed syntax for Pointer arithmetic is located at pointers folder in drawio for your reference>>**

**<<sample program file name was arithptr.c it located in git repository at pointers folder for your reference>>**

Explanation:

1.Start of main Function:

Execution begins at the main function.

2.Array Declaration and Initialization:

int arr[5] = {1, 2, 3, 4, 5};

An array arr of 5 integers is declared and initialized with the values {1, 2, 3, 4, 5}.

3.Pointer Initialization:

int *ptr = arr;

A pointer variable ptr is declared and initialized to point to the first element of arr (i.e., arr[0]).

4.Print Initial Value Pointed by ptr:

printf("Initial value: %d\n", *ptr);

The printf function prints the value pointed to by ptr, which is 1.

Output: Initial value: 1

5.Incrementing the Pointer:

ptr++;

The pointer ptr is incremented to point to the next element in the array (i.e., arr[1]).

6.Print Value After Incrementing the Pointer:

```
printf("After incrementing: %d\n", *ptr);
```

The printf function prints the value pointed to by ptr after incrementing, which is 2.

Output: After incrementing: 2

7.Adding an Integer to the Pointer:

```
ptr += 2;
```

The pointer ptr is advanced by 2 elements in the array (i.e., now points to arr[3]).

8.Print Value After Adding 2 to the Pointer:

```
printf("After adding 2: %d\n", *ptr);
```

The printf function prints the value pointed to by ptr after adding 2, which is 4.

Output: After adding 2: 4

9.Subtracting an Integer from the Pointer:

```
ptr -= 1;
```

The pointer ptr is moved back by 1 element in the array (i.e., now points to arr[2]).

10.Print Value After Subtracting 1 from the Pointer:

```
printf("After subtracting 1: %d\n", *ptr);
```

The printf function prints the value pointed to by ptr after subtracting 1, which is 3.

Output: After subtracting 1: 3

11.Pointer Initialization for Subtraction:

int *ptr1 = &arr[4];

A pointer variable ptr1 is declared and initialized to point to the fifth element of arr (i.e., arr[4]).

int *ptr2 = arr;

A pointer variable ptr2 is declared and initialized to point to the first element of arr (i.e., arr[0]).

12.Subtracting Two Pointers:

int difference = ptr1 - ptr2;

The difference between ptr1 and ptr2 is calculated, which gives the number of elements between them in the array.

13.Print the Difference Between Two Pointers:

printf("Difference between ptr1 and ptr2: %d\n", difference);

The printf function prints the difference, which is 4.

Output: Difference between ptr1 and ptr2: 4

14.End of main Function:

The main function returns 0.

Program execution ends.

## 7.5 Passing Pointers to Functions in C

Passing pointers to functions in C allows functions to modify the original variables and work efficiently with large data structures like arrays. This method avoids making copies of large data, improving performance.

**<<The detailed syntax for Passing Pointers to Functions is located at pointers folder in drawio for your reference>>**

**<<sample program file name was passptr.c it located in git repository at pointers folder for your reference>>**

Explanation:

1.Start of main Function:

Execution begins at the main function.

2.Variable Declaration and Initialization:

int x = 10;

An integer variable x is declared and initialized to 10.

3.Print Original Value:

printf("Original value of x: %d\n", x);

The printf function prints the original value of x, which is 10.

Output: Original value of x: 10

4.Function Call with Pointer Argument:

updateValue(&x);

The updateValue function is called with the address of x passed as an argument.

5.Start of updateValue Function:

Execution moves to the updateValue function with the pointer ptr pointing to x.

6.Update Value via Pointer:

```
*ptr = 20;
```

The value at the address ptr points to is updated to 20.

Since ptr points to x, x is updated to 20.

7.Return to main Function:

Execution returns to the main function after updateValue completes.

8.Print Updated Value:

```
printf("Updated value of x: %d\n", x);
```

The printf function prints the updated value of x, which is 20.

Output: Updated value of x: 20

9.End of main Function:

The main function returns 0.

Program execution ends.

## 7.6 Creating Arrays of Pointers

Arrays of pointers in C are useful for managing collections of pointers to data, such as strings or other arrays. Each element in the array holds a pointer to a data element, providing flexibility and efficiency in memory management and data manipulation.

**<<The detailed syntax for Arrays of Pointers is located  at pointers folder in drawio  for your reference>>**

**<<sample program file name was ptrarr.c it located in git repository at pointers folder for your reference>>**

Explanation:

1.Start of Program Execution:

The program starts executing from the main function.

2.Integer Array (ptrArray) Initialization:

int *ptrArray[3]; declares an array of 3 integer pointers.

int a = 10, b = 20, c = 30; initializes three integer variables.

3.Assigning Addresses to ptrArray Elements:

ptrArray[0] = &a;, ptrArray[1] = &b;, ptrArray[2] = &c; assigns addresses of a, b, and c to elements of ptrArray.

4.Printing Integer Values and Addresses:

Enters a loop to iterate through ptrArray.

Prints each integer value and its corresponding address stored in ptrArray.

5.String Array (names) Initialization:

char *names[3] = {"Alice", "Bob", "Charlie"}; initializes an array of 3 pointers to strings.

6.Printing Strings and Their Addresses:

Enters a loop to iterate through names.

Prints each string and its corresponding address stored in names.

7.End of Execution:

return 0; indicates successful completion of main.

Program execution ends.

## 8 String Manipulation

### 8.1 Reading Strings in C

Definition:

Reading strings in C involves accepting a sequence of characters from user input, typically through functions such as scanf or gets.

#### 8.1.1 Using scanf:

scanf is a standard input function used to read formatted input.

**<<The detailed syntax for Using scanf is located  at strings folder in drawio  for your reference>>**

8.1.2 Using fgets (recommended):

fgets reads a line from the specified stream and stores it into the string.

**<<The detailed syntax for Using fgets is located  at strings folder in drawio  for your reference>>**

8.1.3 Using puts:

Definition:

puts writes a string to the standard output (usually the screen) followed by a newline character.

**<<The detailed syntax for Using puts is located  at strings folder in drawio  for your reference>>**

**<<sample program file name was read.c it located in git repository at string manipulation folder for your reference>>**

Explanation:

1.Program Start:

The main function is invoked, starting the execution of the program.

2.Variable Declaration:

A character array str of size 51 is declared to store the input strings.

3.First Input Prompt:

The program prints: \nEnter up to 50 characters with spaces:\n.

4.Reading Input with fgets:

The function fgets(str, sizeof(str), stdin) is called.

fgets reads up to 50 characters (or until a newline character is encountered) from the standard input (stdin) and stores it in the str array.

5.Output of fgets Input:

The program prints: fgets() read:.

The function puts(str) is called, which prints the string stored in str followed by a newline.

6.Second Input Prompt:define

The program prints: \nEnter up to 50 characters with spaces:\n.

7.Reading Input with scanf:

The function scanf("%s", str) is called.

scanf reads a single word (up to the first whitespace character) from the standard input (stdin) and stores it in the str array.

8.Output of scanf Input:

The program prints: scanf() read: followed by the string stored in str.

9.Program Termination:

The main function returns 0, indicating the successful completion of the program.

## 8.2 Copy the strings

### 8.2.1 Standard strcpy()

Copies the entire source string, including the null terminator, to the destination.Useful when you want to copy a complete string.

**<<The detailed syntax for Standard strcpy is located at strings folder in drawio for your reference>>**

### 8.2.2 strncpy()

Copies up to n characters from the source string to the destination.

If the source string is shorter than n characters, the remaining characters in the destination are padded with null characters ('\0').

**<<The detailed syntax for strncpy is located  at strings folder in drawio  for your reference>>**

8.2.3 strlen()

Calculates the length of a string by counting the number of characters before the null terminator ('\0').

**<<The detailed syntax for strlen is located  at strings folder in drawio  for your reference>>**

**<<sample program file name was cpy.c it located in git repository at string manipulation folder for your reference>>**

Explanation:

1.Program Start:

The main function is invoked, starting the execution of the program.

2.Variable Declarations:

source: A character array initialized with the string "Hello, World!".

3.destination:

An empty character array of size 50 to store the copied string using strcpy.

4.destination_n:

An empty character array of size 50 to store the partially copied string using strncpy.

5.length:

An integer variable to store the length of the string calculated by strlen.

6.Copying Using strcpy:

strcpy(destination, source); copies the entire source string to destination.

7.The program prints:

Copied using strcpy: Hello, World!.

8.Copying Using strncpy:

strncpy(destination_n, source, 5); copies the first 5 characters of source to destination_n.

destination_n[5] = '\0'; manually adds a null terminator to the end of the copied string in destination_n.

The program prints: Copied using strncpy: Hello.

9.Calculating Length Using strlen:

length = strlen(source); calculates the length of source, excluding the null terminator ('\0').

The program prints: Length of source string: 13.

10.Program Termination:

The main function returns 0, indicating the successful completion of the program.

## 8.3 Joining strings:

In C, there isn't a built-in function specifically called "joining string function" like there might be in other languages. However, we commonly achieve string concatenation (joining strings together) using various methods. Let's define these methods and provide a simple example program:

### 8.3.1 Methods of String Concatenation Using strcat:

Definition:

strcat appends a copy of the source string to the destination string.

**<<The detailed syntax for Using strcat is located  at strings folder in drawio  for your reference>>**

### 8.3.3 Using strncat:

appends a specified number of characters from source string to targeted string.

**<<The detailed syntax for strncat is located  at strings folder in drawio  for your reference>>**

### 8.3.4 Using strncat by places:

appends a specified number of characters from source string starting from places to target string.

**<<The detailed syntax for Using strncat is located at strings folder in drawio for your reference>>**

**<<sample program file name was join.c it located in git repository at string manipulation folder for your reference>>**

Explanation:

1.Program Start:

The main function is invoked, starting the execution of the program.

2.Variable Declarations:

s1, s2, s3, s4, s5, s6: Character arrays (char[]) used to store strings.

3.Example 1: strncat(s1, s2, 3);

s1 is initialized with "Hello" and s2 with " Ennbook".

strncat(s1, s2, 3); appends the first 3 characters of s2 (" Enn") to s1, resulting in "Hello Enn".

printf("Result after strncat: %s\n", s1); prints the modified s1: "Hello Enn".

4.Example 2: strncat(s3, (s4 + 1), 4);

s3 is initialized with "Hello" and s4 with " Ennbook".

strncat(s3, (s4 + 1), 4); appends 4 characters from s4 starting from index 1 ("Ennb") to s3, resulting in "Hello Ennbook".

printf("Result after strncat: %s\n", s3); prints the modified s3: "Hello Ennbook".

5.Example 3: strcat(s5, s6);

s5 is initialized with "hi " and s6 with "Copy me!".

strcat(s5, s6); concatenates s6 ("Copy me!") to the end of s5, resulting in s5 containing "hi Copy me!".

printf("Result after strcat: %s\n", s5); prints s5, resulting in "hi Copy me!".

6.Program Termination:

The main function returns 0, indicating the successful execution of the program.

## 8.4 Find Sub Strings

### 8.4.1 strstr()

Purpose:

Searches for the first occurrence of substring sub within the string str.

**<<The detailed syntax for strstr is located  at strings folder in drawio  for your reference>>**

### 8.4.2 Index Calculation:

Purpose:

Calculates the index (position) in str where sub starts.

**<<The detailed syntax for Index Calculation is located at strings folder in drawio for your reference>>**

8.4.3 String comparision:

n C, strcmp is used to compare two strings. It returns an integer value indicating whether the strings are equal or if one is greater or less than the other. Here's how to use strcmp in C: If strcmp returns 0, it means the two strings are equal.

**<<The detailed syntax for String comparision is located at strings folder in drawio for your reference>>**

Example:

strcmp(sub, "Time")

1.sub = "Time"

"Time" = "Time"

Result: 0

Explanation: Both strings are identical, so strcmp() returns 0.

strcmp(sub, "time")

2.sub = "Time"

"time" = "time"

Result: -1

Explanation: The comparison is case-sensitive. In dictionary order, uppercase letters come before lowercase letters. Therefore, "Time" is considered less than "time", and strcmp() returns -1.

strcmp(sub, "TIME")

3.sub = "Time"

"TIME" = "TIME"

Result: 1

Explanation: Similar to the previous case, but in this case, "TIME" is considered greater than "Time" because uppercase letters come after lowercase letters in dictionary order. Hence, strcmp() returns 1.

**<<sample program file name was camp.c it located in git repository at string manipulation folder for your reference>>**

Explanation:

1.Initialization:

Define two strings:

str: "Have A Nice Day"

sub: "Nice"

2.Using strstr():

Check if the substring "Nice" exists within the string str using strstr(str, sub).

If strstr(str, sub) returns NULL, print "Substring "Nice" Not Found".

Otherwise, if strstr(str, sub) returns a non-null pointer, print:

"Substring "Nice" Found at <address>"

Calculate and print the index of the substring using strstr(str, sub) - str.

3.Using strcmp():

Compare the string sub with different cases of "Nice" using strcmp().

Print the results of these comparisons:

"%s Versus \"Nice\": %d\n"

"%s Versus \"nice\": %d\n"

"%s Versus \"NICE\": %d\n"

4.Output:

The program outputs whether the substring "Nice" is found in str, its position if found, and the results of the comparisons between sub and "Nice", "nice", and "NICE".

## 8.5 String Validation

Character Classification and Conversion:

Purpose for ctype.h:

The ctype.h header file in C provides functions and macros for classifying and converting characters.

Headers:

Include #include <ctype.h> at the beginning of your C program to use these functions and macros.

Functions:

Character Classification Functions:

These functions (isalpha(), isdigit(), islower(), isupper(), ispunct(), etc.) determine the type of a character based on its ASCII value.

Character Conversion Functions: These functions (tolower(), toupper()) convert characters between lowercase and uppercase forms.

Usage:

Character Classification: You can use isalpha(), isdigit(), islower(), isupper(), ispunct(), etc., to check if a character belongs to a specific category (alphabetic, numeric, lowercase, uppercase, punctuation).

Character Conversion: tolower() and toupper() functions convert characters to lowercase or uppercase, respectively, if possible; otherwise, they return the character unchanged.

**<<The picturized table format for String Validation is located  at strings folder in drawio for your reference>>**

**<<sample program file name was vali.c it located in git repository at string manipulation folder for your reference>>**

Explanation:

1.Include Headers:

The program includes <stdio.h> for standard input/output operations and <ctype.h> for character handling functions.

2.Main Function:

Initialization: Defines an integer variable ch and assigns it the ASCII value of character 'A'.

3.Function Calls:

isalpha(ch): Checks if ch is an alphabetic character (a-z or A-Z). It prints 1 if true ('A' is alphabetic), otherwise 0.

islower(ch): Checks if ch is a lowercase alphabetic character (a-z). Prints 0 because 'A' is uppercase.

isupper(ch): Checks if ch is an uppercase alphabetic character (A-Z). Prints 1 because 'A' is uppercase.

isdigit(ch): Checks if ch is a digit (0-9). Prints 0 because 'A' is not a digit.

ispunct(ch): Checks if ch is a punctuation character. Prints 0 because 'A' is not a punctuation character.

tolower(ch): Converts ch to lowercase if it is an uppercase letter. Prints 'a'.

toupper(ch): Converts ch to uppercase if it is a lowercase letter (converted from previous tolower()). Prints 'A'.

4.Output:

Each printf() statement displays the result of the corresponding function call.

## 8.6 String Conversion

In C programming, string conversion refers to the process of converting between strings and other data types, such as integers or floating-point numbers. Here's a brief explanation of each function involved in string conversion:

### 8.6.1. itoa

Definition:

itoa (Integer to ASCII) converts an integer to a string representation in C.

**<<The detailed syntax for itoa is located  at strings folder in drawio  for your reference>>**

### 8.6.2. atoi

Definition:

atoi (ASCII to Integer) converts a string representation of an integer to an actual integer value.

**<<The detailed syntax for atoi is located  at strings folder in drawio  for your reference>>**

### 8.6.3. sprintf

Definition:

sprintf (String printf) formats and stores a series of characters and values in a string buffer.

**<<The detailed syntax for sprintf is located  at strings folder in drawio  for your reference>>**

**<<sample program file name was conv.c it located in git repository at string manipulation folder for your reference>>**

Explanation:

1.Include Headers:

The program includes <stdio.h> for standard input/output operations and <stdlib.h> for itoa and atoi functions.

2.Main Function:

itoa Example:

int num1 = 123;

char buffer1[20];

itoa(num1, buffer1, 10);

Converts the integer 123 to a string using itoa with base 10.

printf("Integer to string using itoa: %s\n", buffer1);

Prints the result: "Integer to string using itoa: 123".

atoi Example:

char str[] = "456";

int num2 = atoi(str);

Converts the string "456" to an integer using atoi.

printf("String to integer using atoi: %d\n", num2);

Prints the result: "String to integer using atoi: 456".

sprintf Example:

char buffer3[50];

int num3 = 789;

sprintf(buffer3, "The number is: %d", num3);

Formats the integer 789 into a string "The number is: 789" using sprintf.

printf("Formatted string using sprintf: %s\n", buffer3);

Prints the result: "Formatted string using sprintf: The number is: 789".

3.Output:

Each printf statement displays the result of the corresponding operation.

# 9 Structures

In C programming, a struct (short for structure) is a user-defined data type that allows you to combine different data types into a single unit. Structures are particularly useful for grouping related data together. Here's how to define and use structures in C, including the syntax and a simple example program.

## 9.1 Declaration of Structure

To define a structure, you use the struct keyword followed by the structure name and the structure members enclosed in curly braces.

**<<The detailed syntax for Structure is located at Structures folder in drawio for your reference>>**

## 9.2 Variable declaration

Once a structure is declared, you can declare variables of that structure type. There are two main ways to do this:

### 9.2.1 Separate Declaration:

Declare the structure first and then declare variables of that structure type.

**<<The detailed syntax for Separate Declaration: is located at Structures folder in drawio for your reference>>**

**<<sample program file name was struct.c it located in git repository at Structures folder for your reference>>**

Explanation:

1.Include Header File:

#include <stdio.h>: Include the standard input-output header file.

2.Define Structure:

struct Student: Define a structure named Student with members name, age, gpa, a, and b.

3.coords: Declare a variable coords of type struct Student.

Enter main Function:

Execution starts from the main function.

4.Declare student1 Variable:

struct Student student1;: Declare a variable student1 of type struct Student.

5.Input Student's Name:

printf("Enter student's name: ");: Print the prompt for the student's name.

fgets(student1.name, 50, stdin);: Read up to 49 characters from the standard input and store them in student1.name.

6.Input Student's Age:

printf("Enter student's age: ");: Print the prompt for the student's age.

scanf("%d", &student1.age);: Read an integer from the standard input and store it in student1.age.

7.Input Student's GPA:

printf("Enter student's GPA: ");: Print the prompt for the student's GPA.

scanf("%f", &student1.gpa);: Read a floating-point number from the standard input and store it in student1.gpa.

8.Display Student Information:

printf("\nStudent Information:\n");: Print the header for the student information.

printf("Name: %s", student1.name);: Print the student's name.

printf("Age: %d\n", student1.age);: Print the student's age.

printf("GPA: %.2f\n\n", student1.gpa);: Print the student's GPA to two decimal places.

9.Assign and Print Coordinates:

coords.a = 10;: Assign the value 10 to coords.a.

coords.b = 20;: Assign the value 20 to coords.b.

printf("The coordinate a: %d\n", coords.a);: Print the value of coords.a.

printf("The coordinate b: %d", coords.b);: Print the value of coords.b.

10.End of main Function:

return 0;: The main function returns 0, indicating successful execution, and the program terminates.

## 9.3 Define typedef structures

In C, typedef is used to create alias names for data types, which can make code more readable and easier to manage. When used with structures, typedef can simplify the syntax for declaring variables of that structure type.

In C programming, the dot operator (.) is used to access members of a structure . It allows you to refer to a specific variable or field with the syntax of outer-structure.inner-structure.member.

**<<The detailed syntax for Define typedef structures is located at Structures folder in drawio for your reference>>**

**<<sample program file name was typedef.c it located in git repository at Structures folder for your reference>>**

Explanation:

1.Structure Definitions:

typedef struct { int x; int y; } Point;

typedef struct { Point center; float radius; } Circle;

typedef struct { Point topLeft; Point bottomRight; } Rectangle;

2.Main Function Execution:

3.Circle Initialization:

Circle myCircle; // Declares a variable myCircle of type Circle.

myCircle.center.x = 10; // Sets the x-coordinate of the circle's center.

myCircle.center.y = 15; // Sets the y-coordinate of the circle's center.

myCircle.radius = 5.5; // Sets the radius of the circle.

4.Rectangle Initialization:

Rectangle myRect; // Declares a variable myRect of type Rectangle.

myRect.topLeft.x = 3; // Sets the x-coordinate of the top-left corner of the rectangle.

myRect.topLeft.y = 8; // Sets the y-coordinate of the top-left corner of the rectangle.

myRect.bottomRight.x = 15; // Sets the x-coordinate of the bottom-right corner of the rectangle.

myRect.bottomRight.y = 20; // Sets the y-coordinate of the bottom-right corner of the rectangle.

5.Printing Circle Details:

printf("Circle Center: (%d, %d)\n", myCircle.center.x, myCircle.center.y); // Prints the coordinates of the circle's center.

printf("Circle Radius: %.2f\n", myCircle.radius); // Prints the radius of the circle.

6.Printing Rectangle Details:

printf("Rectangle Top-Left: (%d, %d)\n", myRect.topLeft.x, myRect.topLeft.y); // Prints the coordinates of the top-left corner of the rectangle.

printf("Rectangle Bottom-Right: (%d, %d)\n", myRect.bottomRight.x, myRect.bottomRight.y); // Prints the coordinates of the bottom-right corner of the rectangle.

7.End of Program:

return 0; // Indicates successful execution of the program.

## 9.4 Using Pointers in structures

In C programming, structures (struct) and pointers (*) are fundamental concepts for organizing and manipulating data efficiently. Let's delve into the concepts used in your example without focusing on the specific code implementation.

### 9.4.1 Structures (struct)

Definition:

A structure (struct) is a composite data type that allows you to group together variables of different types under a single name.

Declaration:

After defining a structure, you can declare variables of that structure type, just like primitive data types.

**<<The detailed syntax for Using Pointers in structures is located at Structures folder in drawio for your reference>>**

**<<The picturized table format difference for pointer and array in structuresis located at variables folder in drawio for your reference>>**

**<<sample program file name was strptr.c it located in git repository at Structures folder for your reference>>**

Explanation:

1.Structure Definitions and Initializations:

Two structures are defined using typedef:

ArrType (holding an array of characters) and PtrType (holding a pointer to a character).

Variables arr and ptr of these respective types are initialized with initial values.

2.Printing Initial Values:

printf("\nArray string is a %s", arr.str); prints the initial value of arr.str, which is "Bad ".

printf("\nPointer string is a %s", ptr.str); prints the initial value of ptr.str, which is "Good ".

3.Modifying arr.str:

Individual characters of arr.str are modified to spell "Idea" (arr.str[0] = 'I';, arr.str[1] = 'd';, etc.).

4.Printing Modified arr.str:

printf("%s\n", arr.str); prints the modified arr.str, which now contains "Idea".

5.Modifying ptr.str:

ptr.str = "Idea"; assigns a new string "Idea" to ptr.str.

6.Printing Modified ptr.str:

printf("%s\n", ptr.str); prints the modified ptr.str, which now also contains "Idea".

7.Program Completion:

The main() function completes execution, returning 0 to indicate successful program completion.

## 9.5 Pointing to structures

Pointing to structures in C involves using pointers to reference and manipulate data stored in structures. This capability is particularly useful for dynamically allocating memory, passing structures between functions efficiently, and accessing nested structures or arrays within structures.

The arrow operator (->) in C is used primarily for accessing members of a structure through a pointer to that structure or union. Here are two key points about the arrow operator:

**<<The picturized table format differance for -> and .(dot)is located at variables folder in drawio for your reference>>**

**<<The detailed syntax for Pointing to structures is located at Structures folder in drawio for your reference>>**

**<<sample program file name was mem.c it located in git repository at Structures folder for your reference>>**

Explanation:

1.Main Function Start:

The program starts with including necessary headers (stdio.h) and defines a structure Point with members x and y.

main() function begins.

2.Structure Variable Declaration and Initialization:

A structure variable myPoint of type Point is declared and initialized with values 10 for x and 20 for y.

3.Pointer Declaration and Initialization:

A pointer ptrPoint to a Point structure is declared and initialized to point to myPoint using the address-of operator (&).

4.Printing Original Values:

The original values of myPoint (x = 10 and y = 20) are printed using printf.

5.Modifying Structure Members via Pointer:

Structure members x and y are modified indirectly through ptrPoint using the arrow operator (->).

ptrPoint->x is set to 15.

ptrPoint->y is set to 25.

6.Printing Modified Values:

The modified values of myPoint (x = 15 and y = 25) are printed using printf.

7.Program Completion:

main() function returns 0, indicating successful program execution.

## 9.6 Passing structures to functions

Passing structures to functions in C can be done in two primary ways: by value and by reference (using pointers). Here's a brief overview, along with the syntax and a simple example program for each method:

### 9.6.1 Passing Structures by Value

When you pass a structure by value, a copy of the structure is made. Any changes to the structure inside the function do not affect the original structure.

**<<The detailed syntax for Passing structures to functionsis located  at Structures folder in drawio  for your reference>>**

**<<sample program file name was structvalue.c it located in git repository at Structures folder for your reference>>**

Explanation:

1.Include Standard I/O Library:

The program starts by including the standard input-output library stdio.h.

2.Define Structure:

A structure named Point is defined with two integer members: x and y.

3.Define Function:

A function printPoint is defined that takes a Point structure by value and prints its x and y members.

4.Main Function Begins:

The main() function begins execution.

5.Declare and Initialize Structure Variable:

A Point structure variable myPoint is declared and initialized with x set to 10 and y set to 20.

6.Call Function with Structure by Value:

The printPoint function is called, passing myPoint as an argument. This creates a copy of myPoint inside the printPoint function.

7.Function Execution (printPoint):

Inside the printPoint function, the x and y members of the passed Point structure are printed. Since the structure is passed by value, the original myPoint remains unchanged.

8.Main Function Resumes:

Control returns to the main() function after the printPoint function completes execution.

9.End of Program:

The main() function returns 0, indicating successful completion of the program.

## 9.6.2 Passing Structures by Reference

When you pass a structure by reference, you pass a pointer to the structure. This way, any changes to the structure inside the function will affect the original structure.

**<<The detailed syntax for Passing Structures by Reference is located  at Structures folder in drawio  for your reference>>**

**<<sample program file name was structref.c it located in git repository at Structures folder for your reference>>**

Explanation:

1.Include Standard I/O Library:

The program starts by including the standard input-output library stdio.h.

2.Define Structure:

A structure named Point is defined with two integer members: x and y.

3.Define Function:

A function modifyPoint is defined that takes a pointer to a Point structure and modifies its x and y members.

4.Main Function Begins:

The main() function begins execution.

5.Declare and Initialize Structure Variable:

A Point structure variable myPoint is declared and initialized with x set to 10 and y set to 20.

6.Print Original Values:

The original values of myPoint (x = 10 and y = 20) are printed using printf.

7.Call Function with Structure by Reference:

The modifyPoint function is called, passing the address of myPoint (&myPoint) as an argument.

8.Function Execution (modifyPoint):

Inside the modifyPoint function, the x and y members of the Point structure

9.pointed to by p are modified:

p->x is set to 15.

p->y is set to 25.

10.Main Function Resumes:

Control returns to the main() function after the modifyPoint function completes execution.

11.Print Modified Values:

The modified values of myPoint (x = 15 and y = 25) are printed using printf.

12.End of Program:

The main() function returns 0, indicating successful completion of the program.

## 9.7 Union

Defining a structure (struct) and a union (union) in C allows you to group different types of variables under a single name. However, they are used in different scenarios depending on whether you need to store multiple data items at the same time or just one data item at a time.

A union in C is a user-defined data type that allows storing different data types in the same memory location. Unions provide an efficient way of using the same memory location for multiple purposes.

**<<The detailed syntax for Union is located  at Structures folder in drawio  for your reference>>**

**<<sample program file name was union.c it located in git repository at Structures folder for your reference>>**

Explanation:

1.Include Header:

The program starts by including the standard input-output header file #include <stdio.h>.

2.Define DataUnion:

The typedef union DataUnion is defined to hold three types of data: an integer (intValue), a float (floatValue), and a char (charValue).

3.Define DataStruct:

The typedef struct DataStruct is defined to hold the same three types of data: an integer (intValue), a float (floatValue), and a char (charValue).

4.Main Function:

The main function begins execution.

5.Declare Variables:

Two variables, myUnion of type DataUnion and myStruct of type DataStruct, are declared.

6.Assign Values to Union:

The integer value 42 is assigned to myUnion.intValue.

7.Assign Values to Struct:

The integer value 42 is assigned to myStruct.intValue.

The float value 3.14f is assigned to myStruct.floatValue.

The char value 'A' is assigned to myStruct.charValue.

8.Print Union Memory Locations:

The memory locations of the union members (intValue, floatValue, and charValue) are printed. These addresses will be the same, showing that all members of a union share the same memory space.

9.Print Struct Memory Locations:

The memory locations of the struct members (intValue, floatValue, and charValue) are printed. These addresses will be different, showing that each member of a structure has its own separate memory space.

10.Return Statement:

The main function ends and returns 0, indicating successful execution.

## 9.8 Understanding of memory allocation

### 9.8.1 malloc

Allocates a block of memory of a specified size. The memory is uninitialized.

**<<The detailed syntax for malloc is located at Structures folder in drawio for your reference>>**

### 9.8.2 calloc

Allocates a block of memory for an array of elements and initializes all bytes to zero.

**<<The detailed syntax for calloc is located at Structures folder in drawio for your reference>>**

### 9.8.3 free

Deallocates the memory previously allocated by malloc or calloc.

**<<The detailed syntax for free is located at Structures folder in drawio for your reference>>**

### 9.8.4 Size_t & %zu

Definition:

size_t is an unsigned integer type used to represent sizes and counts in memory. It is commonly used for array indexing and loop counting.

**<<The detailed syntax for Size_t & %zu is located at Structures folder in drawio for your reference>>**

When using printf or similar functions to output a size_t value, %zu should be used to avoid type mismatch issues and ensure proper formatting.

**<<sample program file name was alloc.c it located in git repository at Structures folder for your reference>>**

Explanation:

1.Include Header Files:

#include <stdio.h>: Includes the standard I/O library for input and output functions.

#include <stdlib.h>: Includes the standard library for dynamic memory allocation functions (malloc, calloc, free).

2.Main Function Begins:

int main() {: Starts the definition of the main function.

3.Declare Variables:

int num = 10;: Initializes an integer variable num with the value 10.

int *ptr;: Declares a pointer ptr to an integer.

4.Use sizeof to Determine Size:

printf("Size of int: %zu bytes\n", sizeof(int));: Prints the size of an int in bytes using the %zu format specifier to correctly display size_t values.

5.Allocate Memory Using malloc:

ptr = (int*) malloc(sizeof(int));: Allocates memory for one int and assigns the pointer to ptr.

if (ptr == NULL) {: Checks if memory allocation was successful.

printf("Memory allocation failed using malloc\n");: Prints an error message if allocation failed.

return 1;: Exits the program with an error code 1.

6.Assign and Print Value Using malloc:

*ptr = num;: Assigns the value of num to the allocated memory.

printf("Value at allocated memory using malloc: %d\n", *ptr);: Prints the value stored in the allocated memory.

7.Free Allocated Memory:

free(ptr);: Frees the memory allocated by malloc.

ptr = NULL;: Sets the pointer to NULL to avoid a dangling pointer.

8.Allocate Memory Using calloc:

int *arr = (int*) calloc(5, sizeof(int));: Allocates memory for an array of 5 integers, initializing all elements to 0, and assigns the pointer to arr.

if (arr == NULL) {: Checks if memory allocation was successful.

printf("Memory allocation failed using calloc\n");: Prints an error message if allocation failed.

return 1;: Exits the program with an error code 1.

9.Print Initial Values from calloc:

printf("Values in allocated array using calloc:\n");: Prints a header for the values.

for (int i = 0; i < 5; i++) {: Loops through the array indices.

printf("arr[%d] = %d\n", i, arr[i]);: Prints the initial values (which should be 0).

10.Assign Values to the Array:

for (int i = 0; i < 5; i++) {: Loops through the array indices.

arr[i] = i * 10;: Assigns new values to the array elements.

11.Print Updated Values from calloc:

printf("Updated values in allocated array using calloc:\n");: Prints a header for the updated values.

for (int i = 0; i < 5; i++) {: Loops through the array indices.

printf("arr[%d] = %d\n", i, arr[i]);: Prints the updated values.

12.Free Allocated Memory:

free(arr);: Frees the memory allocated by calloc.

13.End of Main Function:

return 0;: Exits the program with a success code 0.

# 10 File creation

## File

A file is a container in a computer system used to store data, such as text, images, or binary data. Files are typically organized into directories or folders on a storage device like a hard drive, SSD, or USB drive. Each file has a name and an extension that often indicates its format (e.g., .txt, .jpg, .exe).

Key Characteristics of Files:

Name: Identifies the file. It usually includes a file extension that indicates the file type.

Size: The amount of data stored in the file.

Type: Defines the format or content of the file (e.g., text, image, video).

Location: Path where the file is stored on the storage device.

Permissions: Determines who can read, write, or execute the file.

## File Modes

File modes specify how a file should be accessed and manipulated by a program. They dictate whether a file is opened for reading, writing, or appending data, and what happens if the file does not exist or already exists. Different modes can affect the file's contents and attributes.

**<<The picturized format of file modes is located at files folder in drawio for your reference>>**

## 10.1 File pointers

In C programming, a file pointer is used to manage and manipulate files. It provides a way for a program to access files and perform operations such as reading from, writing to, or closing files.

## 10.2 fopen and fclose

fopen:

A function used to open a file. It returns a file pointer that is used to reference the file in subsequent operations.

**<<The detailed syntax for fopen is located  at files folder in drawio  for your reference>>**

fclose:

A function used to close an open file. It takes the file pointer as an argument and releases the resources associated with it.

**<<The detailed syntax for fclose is located  at files folder in drawio  for your reference>>**

## 10.3 fscanf and fprintf

1.fscanf

Definition: fscanf is used to read formatted input from a file. It reads data from a file and stores it in variables according to specified format strings.

**<<The detailed syntax for fscanf is located  at files folder in drawio  for your reference>>**

2.fprintf

Definition: fprintf is used to write formatted output to a file. It writes data to a file in a specified format, similar to how printf writes to the console.

**<<The detailed syntax for fprintf is located at files folder in drawio for your reference>>**

3.Why Use fscanf and fprintf

Formatted Input/Output: Both functions allow for formatted data input and output. This makes it easier to handle data in a structured way.

Convenience: They handle various data types and formats, making it simple to read and write complex data structures.

10.4 fseek

The fseek function in C is used to reposition the file pointer within a file. This function allows you to move the file pointer to a specific location, enabling random access within the file.

SEEK_SET:

The beginning of the file. offset is the absolute position from the start of the file.

SEEK_CUR:

The current position of the file pointer. offset is added to or subtracted from this position.

SEEK_END:

The end of the file. offset is relative to the end of the file. A positive offset moves the pointer past the end, while a negative offset moves it before the end.

**<<The detailed syntax for fseek is located at files folder in drawio for your reference>>**

Use of fseek:

1.Repositioning for Reading/Writing:

After writing data to a file, the file pointer is at the end of the file. To read the data that was just written, you need to move the file pointer back to the beginning.

2.Random Access:

fseek allows you to move the file pointer to any position within the file, enabling you to read or write data at specific locations without having to process the file sequentially.

## 10.5 File Modes

### 10.5.1 r (Read Mode):

Description: Opens a file for reading only. The file must exist; otherwise, the function returns NULL.

**<<The detailed syntax for Read Mode is located  at files folder in drawio  for your reference>>**

**<<sample program file name was read.c it located in git repository at files folder for your reference>>**

Explanation:

1.Include Header File:

Includes the standard I/O library for file operations.

2.Define Main Function:

The entry point of the program.

3.Declare Variables:

Declare file as a FILE pointer.

Declare buffer as a character array.

4.Open File for Reading:

Attempt to open example.txt in read mode.

Check if the file was successfully opened.

Print an error message and exit if the file cannot be opened.

5.Read and Print Content:

Read lines from the file into buffer.

Print each line until the end of the file is reached.

6.Close the File:

Close the file and release resources.

7.Terminate Program:

End the program and return 0 to indicate successful execution.

## 10.5.2 Write Mode (w)

This program opens a file for writing, writes some text to it, and then closes the file.

**<<The detailed syntax for Write Mode is located  at files folder in drawio  for your reference>>**

**<<sample program file name was write.c it located in git repository at files folder for your reference>>**

Explanation:

1.Include Header File:

Includes the standard I/O library for file operations.

2.Define Main Function:

The entry point of the program.

3.Declare File Pointer:

Declare file as a FILE pointer.

4.Open File for Writing:

Attempt to open example.txt in write mode ("w").

Check if the file was successfully opened.

Print an error message and exit if the file cannot be opened.

5.Write to the File:

Write the string "Hello, World!\n" to the file using fprintf.

6.Close the File:

Close the file and release resources.

7.Terminate Program:

End the program and return 0 to indicate successful execution.

## 10.5.3 Append Mode (a)

It opens a file for appending and adds some text to the end of the file.

**<<The detailed syntax for Append Mode is located at files folder in drawio for your reference>>**

**<<sample program file name was append.c it located in git repository at files folder for your reference>>**

Explanation:

1.Include Header File:

Includes the standard I/O library for file operations.

2.Define Main Function:

The entry point of the program.

3.Declare File Pointer:

Declare file as a FILE pointer.

4.Open File for Appending:

Attempt to open example.txt in append mode ("a").

Check if the file was successfully opened.

Print an error message and exit if the file cannot be opened.

5.Append to the File:

Write the string "Appending this line.\n" to the end of the file using fprintf.

6.Close the File:

Close the file and release resources.

7.Terminate Program:

End the program and return 0 to indicate successful execution.

## 10.5.4 Read/Write Mode (r+):

Opens a file for both reading and writing. The file must exist; otherwise, the function returns NULL.

**<<The detailed syntax for Read/Write Mode is located at files folder in drawio for your reference>>**

**<<sample program file name was rw.c it located in git repository at files folder for your reference>>**

Explanation:

1.Include Header File:

Includes the standard I/O library for file operations.

2.Define Main Function:

The entry point of the program.

3.Declare Variables:

Declare file as a FILE pointer.

Declare buffer as a character array to hold data read from the file.

4.Open File for Reading and Writing:

Attempt to open example.txt in read/write mode ("r+").

Check if the file was successfully opened.

Print an error message and exit if the file cannot be opened.

5.Read and Print Current Content:

Use fgets to read lines from the file into buffer.

Print each line read from the file until the end of the file is reached.

6.Move File Pointer to Beginning:

Use fseek(file, 0, SEEK_SET) to move the file pointer back to the beginning of the file.

7.Write New Content to the File:

Use fprintf to write the string "This is new content.\n" to the file.

8.Close the File:

Close the file and release resources.

9.Terminate Program:

End the program and return 0 to indicate successful execution.

10.5.5 Write/Read Mode (w+):

Opens a file for both writing and reading. If the file exists, it is truncated. If it does not exist, a new file is created.

**<<The detailed syntax for Write/Read Mode is located at files folder in drawio for your reference>>**

**<<sample program file name was wr.c it located in git repository at files folder for your reference>>**

Explanation:

1.Include Header File:

Includes the standard I/O library for file operations.

2.Define Main Function:

The entry point of the program.

3.Declare File Pointer:

Declare file as a FILE pointer.

4.Open File for Writing and Reading:

Attempt to open example.txt in write/read mode ("w+").

If the file is opened successfully, file points to the file stream.

If the file cannot be opened (e.g., due to permissions issues or file system problems), fopen returns NULL, and an error message is printed using perror. The program then returns 1 to indicate an error.

5.Write New Content to the File:

Use fprintf(file, "New content written to file.\n"); to write the string "New content written to file.\n" to the file.

6.Move the File Pointer to the Beginning:

Use fseek(file, 0, SEEK_SET); to reposition the file pointer to the beginning of the file, so the newly written content can be read back.

7.Read and Print the Content:

Use fgets(buffer, sizeof(buffer), file) to read lines from the file into buffer.

Print each line read from the file until the end of the file is reached.

8.Close the File:

Use fclose(file); to close the file and release system resources.

9.Terminate Program:

End the program and return 0 to indicate successful execution.

10.5.6 Append/Read Mode (a+):

Opens a file for both reading and appending. Data is added to the end of the file. If the file does not exist, a new file is created.

**<<The detailed syntax for Append/Read Mode is located at files folder in drawio for your reference>>**

**<<sample program file name was ar.c it located in git repository at files folder for your reference>>**

Explanation:

1.Include Header File:

Includes the standard I/O library for file operations.

2.Define Main Function:

The entry point of the program.

3.Declare File Pointer:

Declare file as a FILE pointer.

4.Open File for Appending and Reading:

Attempt to open example.txt in append/read mode ("a+").

If the file is successfully opened, file points to the file stream.

If the file cannot be opened (e.g., due to permission issues or file system problems), fopen returns NULL, and an error message is printed using perror. The program then returns 1 to indicate an error.

5.Append New Content to the File:

Use fprintf(file, "Appending more content.\n"); to append the string "Appending more content.\n" to the end of the file.

6.Move the File Pointer to the Beginning for Reading:

Use fseek(file, 0, SEEK_SET); to reposition the file pointer to the beginning of the file. This is necessary because, after opening the file in append mode, the file pointer is positioned at the end of the file.

7.Read and Print the Content:

Use fgets(buffer, sizeof(buffer), file) to read lines from the file into buffer.

Print each line read from the file until the end of the file is reached.

8.Close the File:

Use fclose(file); to close the file and release system resources.

9.Terminate Program:

End the program and return 0 to indicate successful execution.

## 10.6 Standard input and output:

Standard input and output in C are essential for interacting with users and external systems, allowing programs to receive data from the user (standard input) and display results (standard output). These mechanisms provide a consistent and platform-independent way to handle input and output operations. They enable dynamic data exchange, making programs flexible and versatile.

1.Input Functions:

### 10.6.1 fgetc:

Reads a single character from a specified file stream.Ideal for character-by-character file reading.

**<<The detailed syntax for fgetc is located  at files folder in drawio  for your reference>>**

10.6.2 fgets:

Reads a string from a specified file stream, up to a newline or specified length.Useful for reading lines of text from a file.

**<<The detailed syntax for fgets is located  at files folder in drawio  for your reference>>**

10.6.3 fread:

Reads binary data from a file into an array.Efficient for reading multiple elements at once.

**<<The detailed syntax for fread is located  at files folder in drawio  for your reference>>**

10.6.4 fscanf:

Reads formatted input from a file.Parses data according to a specified format.Returns the number of input items successfully matched and assigned.

**<<The detailed syntax for fscanf is located  at files folder in drawio  for your reference>>**

2.Output Functions:

10.6.5 fputc:

Writes a single character to a specified file stream.Useful for character-by-character file manipulation.

**<<The detailed syntax for fputc is located  at files folder in drawio  for your reference>>**

10.6.6 fputs:

Writes a null-terminated string to a specified file stream.Simplifies the process of writing strings to files.

**<<The detailed syntax for fputs is located  at files folder in drawio  for your reference>>**

10.6.7 fwrite:

Writes binary data from an array to a file.Efficient for writing multiple elements at once.

**<<The detailed syntax for fwrite is located  at files folder in drawio  for your reference>>**

10.6.8 fprintf:

Writes formatted output to a file.Formats data according to a specified format.

**<<The detailed syntax for fprintf is located  at files folder in drawio  for your reference>>**

**<<sample program file name was std.c it located in git repository at files folder for your reference>>**

Explanation:

1.Opening File for Writing:

The file example.txt is opened for writing (w mode).

If the file cannot be opened, an error message is printed and the program terminates with an exit code of 1.

2.Writing to the File:

fputc('A', file); writes the character 'A' to the file.

fputs("Hello, World!\n", file); writes the string "Hello, World!\n" to the file.

fprintf(file, "Number: %d\n", 42); writes the formatted string "Number: 42\n" to the file.

The file is then closed.

3.Opening File for Reading and Writing:

The file example.txt is opened for reading and writing (r+ mode).

If the file cannot be opened, an error message is printed and the program terminates with an exit code of 1.

4.Reading from the File:

int c = fgetc(file); reads the first character from the file and stores it in c. It prints "Read character: A".

5.Resetting File Pointer and Reading a String:

fseek(file, 0, SEEK_SET); moves the file pointer to the beginning of the file.

fgets(buffer, sizeof(buffer), file); reads a string from the file into buffer and prints "Read string: AHello, World!\n".

6.Resetting File Pointer and Reading Data:

fseek(file, 0, SEEK_SET); moves the file pointer to the beginning of the file.

fread(buffer, sizeof(char), 13, file); reads 13 characters from the file into buffer.

The buffer is null-terminated and prints "Read data: AHello, World!".

7.Appending Data to the File:

fseek(file, 0, SEEK_END); moves the file pointer to the end of the file.

fwrite(data, sizeof(char), strlen(data), file); writes the string "Appending data" to the file.

The file is closed.

8.Reopening File for Reading:

The file example.txt is opened for reading (r mode).

If the file cannot be opened, an error message is printed and the program terminates with an exit code of 1.

9.Reading Formatted Input:

fscanf(file, "%s", buffer); reads the first word from the file into buffer and prints "Read formatted input: AHello,".

10.Closing the File:

The file is closed.

## 10.7 C Errors

Predefined Error Codes in errno.h:

The errno.h header file contains definitions for a variety of error codes that can be set by system calls and library functions when an error occurs. Each error code has a unique integer value and a corresponding symbolic name.

### 10.7.1 perror

Purpose:

The perror function prints a descriptive error message to the standard error stream

Usage:

It is used to provide a human-readable message explaining why a certain error occurred.The message printed by perror includes the string passed to it, followed by a colon, a space, and the textual representation.

**<<The detailed syntax for perror is located at files folder in drawio for your reference>>**

### 10.7.2 strerror

Purpose:

The strerror function returns a pointer to the textual representation of the error code passed to it.

Usage:

It is used to convert an error number (errno) into a human-readable string.This is particularly useful for logging or displaying error messages programmatically without using perror.

**<<The detailed syntax for strerror is located  at files folder in drawio  for your reference>>**

**<<sample program file name was error.c it located in git repository at files folder for your reference>>**

Explanation:

1.Include Headers:

The program includes necessary header files: stdio.h for standard I/O functions, errno.h for error handling, and string.h for string functions.

2.Declare Variables:

A FILE pointer file is declared to handle the file operations.

An integer i is declared for use in the loop later.

3.Attempt to Open a File:

The fopen function is called to open a file named "no.file" in read mode ("r").

The result is assigned to the file pointer.

4.Check File Opening Result:

If the file is successfully opened (file != NULL), it prints "File opened".

If the file is not successfully opened (file == NULL), it prints an error message using perror("Error"). This prints the string "Error" followed by a description of the error based on the current errno value (likely "No such file or directory" in this case).

5.Loop to Print Error Messages:

A for loop runs from 0 to 19.

For each value of i, it prints the error number (i) and its corresponding error message using strerror(i).

strerror(i) returns a string that describes the error code i.

6.End of Program:

The program ends and returns 0 to indicate successful execution.

## 10.8 C Date and Time

The time.h header file in C provides various functions and types to manipulate and format time and date values. It includes functions for retrieving the current time, converting between different time representations, and formatting time and date strings.

1.time_t:

Standard Data Type:

time_t is a standard type defined by the C standard library, not a user-defined type. It is designed to be portable across different systems, providing a consistent way to represent and manipulate time.

Type:

time_t is an arithmetic data type, typically defined as a long or an int on many systems, but the exact implementation can vary. It represents the number of seconds elapsed since the Unix epoch (January 1, 1970, at midnight UTC).

**<<The picturized format for Key Functions in time.h is located  at files  folder in drawio for your reference>>**

**<<The picturized format for tm structureis located  at files  folder in drawio  for your reference>>**

Explanation:

1.Include Header Files:

#include <stdio.h>: Includes the Standard I/O library.

#include <time.h>: Includes the time library for time functions.

2.Function main:

Variable Declarations:

time_t rawtime;: Variable to store the raw time in seconds.

struct tm *timeinfo;: Pointer to a tm structure for storing local and UTC time information.

char buffer[80];: Character array to hold the formatted time string.

3.Get the Current Time:

time(&rawtime);: Fetches the current calendar time (seconds since the epoch) and stores it in rawtime.

4.Convert to Local Time:

timeinfo = localtime(&rawtime);: Converts rawtime to a tm structure representing the local time.

5.Print Local Time Using asctime:

printf("Current local time: %s", asctime(timeinfo));: Converts the tm structure to a string and prints the local time in a readable format.

6.Format Time Using strftime:

strftime(buffer, sizeof(buffer), "Formatted date and time: %Y-%m-%d %H:%M:%S", timeinfo);: Formats the local time into a custom string format and stores it in buffer.

printf("%s\n", buffer);: Prints the formatted date and time.

7.Get UTC Time:

timeinfo = gmtime(&rawtime);: Converts rawtime to a tm structure representing Coordinated Universal Time (UTC).

8.Print UTC Time Using asctime:

printf("Current UTC time: %s", asctime(timeinfo));: Converts the UTC tm structure to a string and prints the UTC time in a readable format.

9.Return Statement:

return 0;: Ends the main function and returns 0 to indicate successful completion.

## 10.9 Genarating random numbers

The stdlib.h library in C provides a range of functions for performing general utility operations such as memory allocation, process control, conversions, and others. Two important functions in this library are rand and srand.

1.rand Function

Purpose: The rand function generates pseudo-random numbers.

Range: It returns an integer value between 0 and RAND_MAX (a constant defined in <stdlib.h>).

Deterministic Sequence: Without proper seeding, rand produces the same sequence of numbers each time the program is run.

**<<The detailed syntax for rand is located  at files folder in drawio  for your reference>>**

2.srand Function

Purpose: The srand function sets the starting point for producing a series of pseudo-random integers by rand.

Seeding: By providing a different seed value (often the current time), you ensure that rand generates a different sequence of numbers on each run.

Initialization: If srand is not called before rand, the generator behaves as if srand(1) was called, leading to a predictable sequence of numbers.

**<<The detailed syntax for srand is located at files folder in drawio for your reference>>**

**<<sample program file name was rand.c it located in git repository at files folder for your reference>>**

Explanation:

1.Include Header Files:

The program starts by including the necessary header files: <stdio.h>, <stdlib.h>, and <time.h>.

2.Function main:

The main function is defined and begins execution.

3.Variable Declaration:

int i; is declared for use in the loop that generates random numbers.

4.Seed the Random Number Generator:

srand(time(NULL)); seeds the random number generator with the current time. This ensures that the random numbers generated will be different each time the program runs.

5.Print Header:

printf("Three-digit random numbers:\n"); prints a header message to the console.

6.Generate and Print Three-Digit Random Numbers:

A for loop runs 5 times to generate and print 5 three-digit random numbers.

7.Inside the Loop:

Step 1: Generate a raw random number using rand().

int raw_random_number = rand();

Step 2: Apply the modulus operation to limit the range to 0-899.

int limited_random_number = raw_random_number % 900;

Step 3: Shift the range to 100-999.

int random_number = limited_random_number + 100;

Print the final three-digit random number.

printf("%d\n", random_number);

8.Loop Iterations:

Steps 7.1 to 7.4 are repeated for each iteration of the loop (5 times in total).

9.End of Program:

After the loop completes, the program reaches the return 0; statement, signaling successful completion of the program.

## 10.10 Windows API (WinAPI)

Definition:

The Windows API (WinAPI) is a set of Microsoft Windows operating system interfaces that allows applications to interact with the operating system and other software. It provides a wide range of functions for performing tasks such as managing windows, handling user input, accessing files, and more.

WinAPI is used in native Windows applications written in languages like C or C++ to create robust, efficient applications that can interact directly with the Windows operating system.

### 10.10.1 windows.h:

Definition: windows.h is a header file that includes declarations for all the functions, constants, and data types provided by the Windows API. It provides access to various Windows functions for managing windows, handling messages, working with files, and more.

Purpose: It allows developers to interact with the Windows operating system by including necessary declarations for WinAPI functions and structures.

### 10.10.2 WINAPI:

Definition: WINAPI is a macro used to specify the calling convention for functions in the Windows API.

### 10.10.3 WinMain:

Definition: WinMain is the entry point for a Windows application. It is the function that the Windows operating system calls to start a Windows-based application.

Purpose: It is analogous to the main function in console applications but is specifically used for Windows GUI applications.

### 10.10.4 HINSTANCE hInstance:

Definition: HINSTANCE is a handle to an instance of the application. It uniquely identifies the application's executable file and provides context for resource management.

### 10.10.5 HINSTANCE hPrevInstance:

Definition: HINSTANCE hPrevInstance is a handle to the previous instance of the application. In modern versions of Windows, this parameter is always NULL because multiple instances of a Windows application do not require this handle.

### 10.10.6 LPSTR lpCmdLine:

Definition: LPSTR is a pointer to a string that contains the command-line arguments passed to the application.

Purpose: It allows the application to receive and process command-line arguments when it starts.

### 10.10.7 int nCmdShow:

Definition: nCmdShow is an integer that specifies how the application window should be shown (e.g., minimized, maximized, or normal).

Purpose: It provides instructions on how to display the application's main window based on the user's or system's preferences.

### 10.10.8 MB_OK | MB_ICONINFORMATION:

Definition: These are flags used in the MessageBox function to specify the type of buttons and icons to display in the message box.

MB_OK adds an OK button to the message box.

MB_ICONINFORMATION displays an information icon.

Purpose: To customize the appearance and functionality of the message box, making it suitable for displaying information messages to the user.

**<<The detailed syntax for Windows API is located  at files  folder in drawio  for your reference>>**

**<<sample program file name was win.c it located in git repository at files folder for your reference>>**

Explanation:

1.Include the Windows Header File:

#include <windows.h>: This line includes the Windows API header file, which contains declarations for all the functions and data types used in Windows programming.

2.Entry Point for the Application:

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow):

This is the entry point for a Windows application. It is similar to the main function in console applications.

3.Parameters:

hInstance: Handle to the current instance of the application.

hPrevInstance: Handle to the previous instance of the application (always NULL in modern Windows).

lpCmdLine: Pointer to a null-terminated string specifying the command line for the application.

nCmdShow: Specifies how the window is to be shown (ignored in this code).

4.Display a Message Box:

MessageBox(NULL, "Hello, World!", "Message Box Title", MB_OK | MB_ICONINFORMATION);:

This function call creates and displays a message box.

5.Parameters:

NULL: No owner window.

"Hello, World!": The text message to be displayed in the message box.

"Message Box Title": The title of the message box.

MB_OK | MB_ICONINFORMATION: Flags that specify the contents and behavior of the message box. MB_OK creates an OK button, and MB_ICONINFORMATION displays an information icon.

The function displays the message box and waits for the user to click the OK button.

6.Return from the Application:

return 0;: The application returns 0, indicating that it has completed successfully.

## 10.11 Timer set

The clock() function is part of the C standard library, defined in the <time.h> header. It provides a way to measure the processor time consumed by a program. Here are the details:

### 10.11.1 clock() Function

Purpose: The clock() function returns the number of clock ticks that have elapsed since the program started execution.

### 10.11.2 clock_t Data Type

Definition: clock_t is a typedef defined in <time.h>. It is an arithmetic type capable of representing times.

Purpose: This type is used to store clock tick counts returned by clock()

**<<The detailed syntax for clock_t is located  at files  folder in drawio  for your reference>>**

## 10.11.2 CLOCKS_PER_SEC

Definition: CLOCKS_PER_SEC is a macro defined in <time.h>.

Value: It represents the number of clock ticks per second. The value is implementation-defined but is typically 1,000,000 in most systems, which corresponds to microsecond granularity.

Purpose: This constant allows the conversion of clock tick counts into seconds.

A "clock tick" is the smallest measurable unit of time in a system, corresponding to one increment of the system's clock counter. It is used to measure time intervals, typically by the system's hardware timer.

In C programming, CLOCKS_PER_SEC is a constant that defines the number of clock ticks in one second, allowing conversion between clock ticks and real time. For instance, if CLOCKS_PER_SEC is 1,000,000, then 1,000,000 clock ticks represent one second.

**<<sample program file name was timer.c it located in git repository at files folder for your reference>>**

Explanation:

1.Initialization:

The program includes necessary headers <stdio.h> for input/output functions and <time.h> for time-related functions.

2.Entering main() Function:

Time Tracking:

The variable start is defined to store the starting time of the countdown.

time(NULL) is called to get the current time in seconds since the Unix epoch (January 1, 1970). This value is stored in start.

Message Output: The program outputs "Starting countdown...\n\n" to indicate the beginning of the countdown process.

3.Countdown Loop:

A for loop is used to count down from 10 to 0.

Inside the Loop:

The current countdown value i is printed, formatted as " - i".

The wait(1) function is called to create a 1-second delay before the next iteration of the loop.

4.wait() Function:

Purpose:

To create a delay for a specified number of seconds (1 second in this case).

Implementation:

clock() returns the number of clock ticks elapsed since the program started.

CLOCKS_PER_SEC is a constant representing the number of clock ticks per second.

end_wait is calculated by adding the current clock ticks (clock()) to the number of ticks corresponding to the desired delay (seconds * CLOCKS_PER_SEC).

The while loop runs until the current clock ticks (clock()) exceed end_wait, effectively creating a busy-wait delay.

5.End of Countdown:

After the loop finishes, the program gets the current time again using time(NULL) and stores it in the variable end.

6.Runtime Calculation:

The total runtime is calculated using difftime(end, start), which gives the difference in seconds between the start and end times.

The result is printed, showing the total runtime of the countdown.

7.Program Termination:

The main() function returns 0, indicating successful execution.

Reference:

1. ASCII Character Codes

Definition:

ASCII (American Standard Code for Information Interchange) is a character encoding standard used to represent text in computers and other devices that use text. It encodes 128 specified characters into seven-bit integers.

Character Set:

ASCII includes 33 non-printing control characters (such as carriage return, line feed, and null character) and 95 printable characters, which include letters (both uppercase and lowercase), digits, punctuation marks, and special symbols.

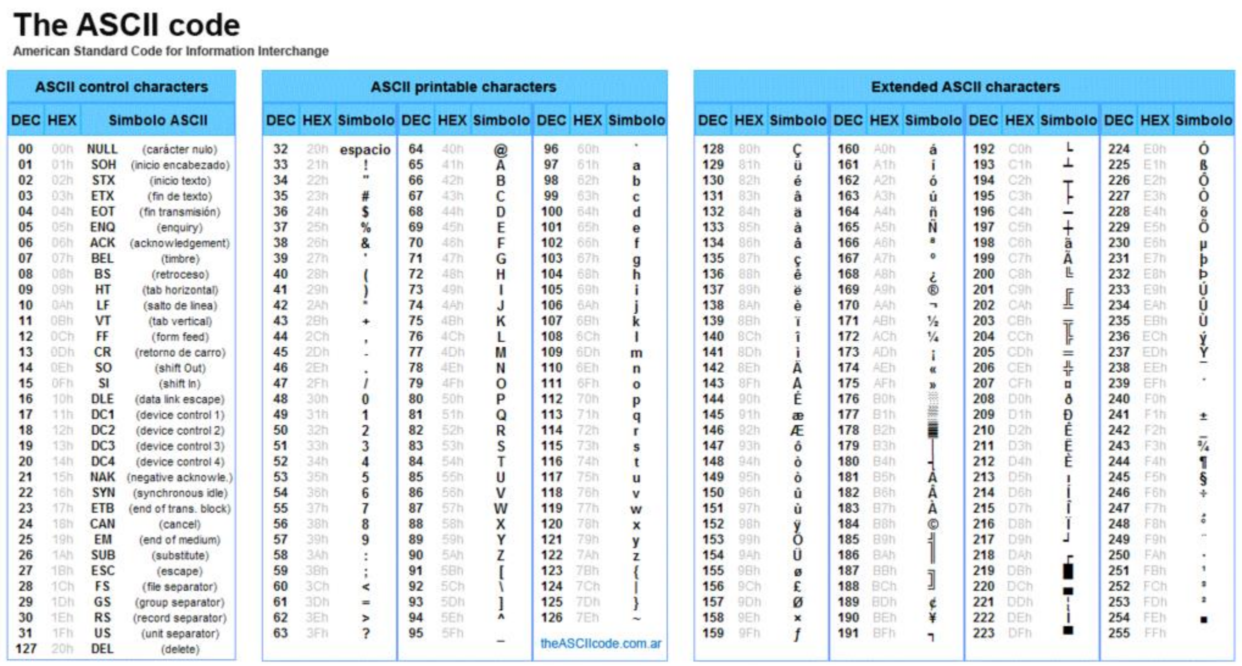
Usage and Compatibility: A

SCII serves as the basis for most modern character encoding systems and ensures compatibility across different devices and platforms by providing a standardized way to represent text. It is especially significant in early computing and data transmission standards.

1.Uppercase Letters as 65 to 90

2.Lowercase Letters as 97 to 122

3.Digits as 48 to 57



**The ASCII code**

American Standard Code for Information Interchange

**ASCII control characters**

| DEC | HEX | Simbolo ASCII | |
|---|---|---|---|
| 00 | 00h | NULL | (carácter nulo) |
| 01 | 01h | SOH | (inicio encabezado) |
| 02 | 02h | STX | (inicio texto) |
| 03 | 03h | ETX | (fin de texto) |
| 04 | 04h | EOT | (fin transmisión) |
| 05 | 05h | ENQ | (enquiry) |
| 06 | 06h | ACK | (acknowledgement) |
| 07 | 07h | BEL | (timbre) |
| 08 | 08h | BS | (retroceso) |
| 09 | 09h | HT | (tab horizontal) |
| 10 | 0Ah | LF | (salto de linea) |
| 11 | 0Bh | VT | (tab vertical) |
| 12 | 0Ch | FF | (form feed) |
| 13 | 0Dh | CR | (retorno de carro) |
| 14 | 0Eh | SO | (shift Out) |
| 15 | 0Fh | SI | (shift In) |
| 16 | 10h | DLE | (data link escape) |
| 17 | 11h | DC1 | (device control 1) |
| 18 | 12h | DC2 | (device control 2) |
| 19 | 13h | DC3 | (device control 3) |
| 20 | 14h | DC4 | (device control 4) |
| 21 | 15h | NAK | (negative acknowle.) |
| 22 | 16h | SYN | (synchronous idle) |
| 23 | 17h | ETB | (end of trans. block) |
| 24 | 18h | CAN | (cancel) |
| 25 | 19h | EM | (end of medium) |
| 26 | 1Ah | SUB | (substitute) |
| 27 | 1Bh | ESC | (escape) |
| 28 | 1Ch | FS | (file separator) |
| 29 | 1Dh | GS | (group separator) |
| 30 | 1Eh | RS | (record separator) |
| 31 | 1Fh | US | (unit separator) |
| 127 | 20h | DEL | (delete) |

**ASCII printable characters**

| DEC | HEX | Simbolo | DEC | HEX | Simbolo | DEC | HEX | Simbolo |
|---|---|---|---|---|---|---|---|---|
| 32 | 20h | espacio | 64 | 40h | @ | 96 | 60h | ` |
| 33 | 21h | ! | 65 | 41h | A | 97 | 61h | a |
| 34 | 22h | " | 66 | 42h | B | 98 | 62h | b |
| 35 | 23h | # | 67 | 43h | C | 99 | 63h | c |
| 36 | 24h | $ | 68 | 44h | D | 100 | 64h | d |
| 37 | 25h | % | 69 | 45h | E | 101 | 65h | e |
| 38 | 26h | & | 70 | 46h | F | 102 | 66h | f |
| 39 | 27h | ' | 71 | 47h | G | 103 | 67h | g |
| 40 | 28h | ( | 72 | 48h | H | 104 | 68h | h |
| 41 | 29h | ) | 73 | 49h | I | 105 | 69h | i |
| 42 | 2Ah | * | 74 | 4Ah | J | 106 | 6Ah | j |
| 43 | 2Bh | + | 75 | 4Bh | K | 107 | 6Bh | k |
| 44 | 2Ch | , | 76 | 4Ch | L | 108 | 6Ch | l |
| 45 | 2Dh | - | 77 | 4Dh | M | 109 | 6Dh | m |
| 46 | 2Eh | . | 78 | 4Eh | N | 110 | 6Eh | n |
| 47 | 2Fh | / | 79 | 4Fh | O | 111 | 6Fh | o |
| 48 | 30h | 0 | 80 | 50h | P | 112 | 70h | p |
| 49 | 31h | 1 | 81 | 51h | Q | 113 | 71h | q |
| 50 | 32h | 2 | 82 | 52h | R | 114 | 72h | r |
| 51 | 33h | 3 | 83 | 53h | S | 115 | 73h | s |
| 52 | 34h | 4 | 84 | 54h | T | 116 | 74h | t |
| 53 | 35h | 5 | 85 | 55h | U | 117 | 75h | u |
| 54 | 36h | 6 | 86 | 56h | V | 118 | 76h | v |
| 55 | 37h | 7 | 87 | 57h | W | 119 | 77h | w |
| 56 | 38h | 8 | 88 | 58h | X | 120 | 78h | x |
| 57 | 39h | 9 | 89 | 59h | Y | 121 | 79h | y |
| 58 | 3Ah | : | 90 | 5Ah | Z | 122 | 7Ah | z |
| 59 | 3Bh | ; | 91 | 5Bh | [ | 123 | 7Bh | { |
| 60 | 3Ch | < | 92 | 5Ch | \ | 124 | 7Ch | \| |
| 61 | 3Dh | = | 93 | 5Dh | ] | 125 | 7Dh | } |
| 62 | 3Eh | > | 94 | 5Eh | ^ | 126 | 7Eh | ~ |
| 63 | 3Fh | ? | 95 | 5Fh | _ | | | |

theASCIIcode.com.ar

**Extended ASCII characters**

| DEC | HEX | Simbolo | DEC | HEX | Simbolo | DEC | HEX | Simbolo | DEC | HEX | Simbolo |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | 80h | Ç | 160 | A0h | á | 192 | C0h | └ | 224 | E0h | Ó |
| 129 | 81h | ü | 161 | A1h | í | 193 | C1h | ┴ | 225 | E1h | ß |
| 130 | 82h | é | 162 | A2h | ó | 194 | C2h | ┬ | 226 | E2h | Ô |
| 131 | 83h | â | 163 | A3h | ú | 195 | C3h | ├ | 227 | E3h | Ò |
| 132 | 84h | ä | 164 | A4h | ñ | 196 | C4h | ─ | 228 | E4h | õ |
| 133 | 85h | à | 165 | A5h | Ñ | 197 | C5h | ┼ | 229 | E5h | Õ |
| 134 | 86h | å | 166 | A6h | ª | 198 | C6h | ã | 230 | E6h | µ |
| 135 | 87h | ç | 167 | A7h | º | 199 | C7h | Ã | 231 | E7h | þ |
| 136 | 88h | ê | 168 | A8h | ¿ | 200 | C8h | ╚ | 232 | E8h | Þ |
| 137 | 89h | ë | 169 | A9h | ® | 201 | C9h | ╔ | 233 | E9h | Ú |
| 138 | 8Ah | è | 170 | AAh | ¬ | 202 | CAh | ╩ | 234 | EAh | Û |
| 139 | 8Bh | ï | 171 | ABh | ½ | 203 | CBh | ╦ | 235 | EBh | Ù |
| 140 | 8Ch | î | 172 | ACh | ¼ | 204 | CCh | ╠ | 236 | ECh | ý |
| 141 | 8Dh | ì | 173 | ADh | ¡ | 205 | CDh | ═ | 237 | EDh | Ý |
| 142 | 8Eh | Ä | 174 | AEh | « | 206 | CEh | ╬ | 238 | EEh | ¯ |
| 143 | 8Fh | Å | 175 | AFh | » | 207 | CFh | ¤ | 239 | EFh | ´ |
| 144 | 90h | É | 176 | B0h | ░ | 208 | D0h | ð | 240 | F0h | |
| 145 | 91h | æ | 177 | B1h | ▒ | 209 | D1h | Ð | 241 | F1h | ± |
| 146 | 92h | Æ | 178 | B2h | ▓ | 210 | D2h | Ê | 242 | F2h | ‗ |
| 147 | 93h | ô | 179 | B3h | │ | 211 | D3h | Ë | 243 | F3h | ¾ |
| 148 | 94h | ö | 180 | B4h | ┤ | 212 | D4h | È | 244 | F4h | ¶ |
| 149 | 95h | ò | 181 | B5h | Á | 213 | D5h | ı | 245 | F5h | § |
| 150 | 96h | û | 182 | B6h | Â | 214 | D6h | Í | 246 | F6h | ÷ |
| 151 | 97h | ù | 183 | B7h | À | 215 | D7h | Î | 247 | F7h | ¸ |
| 152 | 98h | ÿ | 184 | B8h | © | 216 | D8h | Ï | 248 | F8h | ° |
| 153 | 99h | Ö | 185 | B9h | ╣ | 217 | D9h | ┘ | 249 | F9h | ¨ |
| 154 | 9Ah | Ü | 186 | BAh | ║ | 218 | DAh | ┌ | 250 | FAh | · |
| 155 | 9Bh | ø | 187 | BBh | ╗ | 219 | DBh | █ | 251 | FBh | ¹ |
| 156 | 9Ch | £ | 188 | BCh | ╝ | 220 | DCh | ▄ | 252 | FCh | ³ |
| 157 | 9Dh | Ø | 189 | BDh | ¢ | 221 | DDh | ¦ | 253 | FDh | ² |
| 158 | 9Eh | × | 190 | BEh | ¥ | 222 | DEh | Ì | 254 | FEh | ■ |
| 159 | 9Fh | ƒ | 191 | BFh | ┐ | 223 | DFh | ▀ | 255 | FFh | |

2. In C programming, the printf() function is used to output formatted text to the standard output (usually the screen). The format specifiers in printf() dictate how the subsequent arguments are formatted.

**<<The picturized table format for printf() format specifiers is located at reference folder in drawio for your reference>>**

3. In C programming, the scanf() function is used for input, reading formatted data from the standard input (usually the keyboard). It uses format specifiers to interpret the input data and store the values in the provided variable addresses.

These specifiers are used in the format string of scanf() to parse the input and store it into the corresponding variables. It's crucial to provide the correct variable addresses for each specifier, using the & operator except for strings (arrays).

**<<The picturized table format for scanf() format specifiers is located at reference folder in drawio for your reference>>**

4.Functions for character input and output

In C programming, several functions are available for handling character input and output. These functions are part of the C standard library, typically included via the <stdio.h> header. Below are the key functions used for character input and output.

**<<The picturized table format for i/o functions is located  at reference folder in drawio  for your reference>>**

5.String functions

Here's a table format summarizing common string handling functions in C, including their syntax and descriptions. These functions are part of the C Standard Library, typically found in the <string.h> header file.

**<<The picturized table format for String functions is located  at reference folder in drawio for your reference>>**

6.Math functions

in C, typically included in the <math.h> header file. These functions cover a range of mathematical operations and are widely used in scientific and engineering applications.

**<<The picturized table format for Math functions is located  at reference folder in drawio for your reference>>**

7.Utility functions

Utility functions in C, often included in the <stdlib.h> and other standard library headers, provide essential operations for tasks like memory allocation, process control, searching, sorting, and converting data types.

**<<The picturized table format for Utility functions is located  at reference folder in drawio for your reference>>**

8. Diagnostic functions

The assert in C is a diagnostic tool used to test assumptions made by the program and to identify logical errors during development. When the condition evaluated by assert is false, it generates an error message and terminates the program. This is particularly useful for catching bugs early in the development cycle by ensuring that certain conditions hold true.

**<<The detailed syntax for Diagnostic functions is located at Structures folder in drawio for your reference>>**

How assert Works:

When the expression in the assert statement evaluates to false:

The program prints an error message including the expression, the source file name, and the line number where the assert failed.

The program terminates by calling abort().

When the expression is true, the assert macro has no effect, and the program continues execution normally.

9. Argument Functions

The <stdarg.h> header in C provides a set of macros for handling functions with a variable number of arguments.The macros defined in <stdarg.h> allow functions to accept and process a variable number of arguments. This is achieved through the va_list type and the macros va_start, va_arg, and va_end, which manage the argument list's state and allow access to each argument.

**<<The picturized table format for Argument functions is located at reference folder in drawio for your reference>>**

## 10.Date and Time Functions in C

The <time.h> header provides functionality for manipulating and formatting date and time in C. Below is a comprehensive overview of the time-related functions, variables, and format specifiers used with strftime().

**<<The picturized table format for Time and Clock Functions ns is located at reference folder in drawio for your reference>>**

**<<The picturized table format for Time and Clock Variables table is located at reference folder in drawio for your reference>>**

## 11. All type of format specifiers for time functions

strftime() uses format specifiers to format the date and time. Below is a table of common format specifiers:

**<<The picturized table format for Format Specifiers is located at reference folder in drawio for your reference>>**

## 12. Jump functions

The setjmp.h header file provides the definitions for the setjmp and longjmp functions, which facilitate non-local jumps in C. These functions are primarily used for error handling and recovery in complex programs.

**<<The picturized table format for Jump functions  is located  at reference folder in drawio for your reference>>**

13. Limit & float Contants

In C programming, various standard libraries define constants that specify limits for fundamental data types and system-specific properties. These limits are defined in headers like <limits.h>, <float.h>, and <stdint.h>.

**<<The picturized table format for Limit & float Contants table is located  at reference folder in drawio  for your reference>>**

www.ennbook.com