LearnFrenzy Presents

C PROGRAMMING

The Complete Handbook

```c
#include <stdio.h>

int main() {
  printf("Hello, World!\n");
  return 0;
}
```

Mastering the Foundations of Computer Science

C Programming Handbook

CHAPTER 01

Beginner Friendly Learning Guide

What is Programming?

Programming is a way of communicating with computers by giving them instructions to perform specific tasks.

Just like we use languages such as Hindi or English to communicate with each other, programming languages allow us to communicate with computers and tell them what to do.

What is C?

C is one of the oldest and most powerful programming languages. It is known for its speed, efficiency, and close interaction with computer hardware.

It was developed by Dennis Ritchie at AT&T Bell Labs in 1972 and has influenced many modern programming languages.

Uses of C

01

Used to build operating systems such as Windows, Linux, and other low-level system software.

02

Commonly used for writing device drivers and programming embedded systems.

03

Suitable for microcontrollers, cameras, IoT devices, and memory-constrained systems.

04

Ideal for applications where high performance, speed, and low latency are critical.

Installation

GETTING STARTED

Just install it like a game

Install **Visual Studio Code** as your code editor and **MinGW GCC** as your C compiler. VS Code is used to write your programs, while GCC converts your C source code into machine code that your computer can execute.

C Programming Handbook

CHAPTER 01

Variables, Constants & Keywords

Variables

A variable is a named container that stores a value in memory. Variables make programs flexible because their values can change while the program is running.

Example

```
int a = 3;
float b = 4.7;
char c = 'A';
```

Rules for Naming Variables

01

The first character must be an alphabet or underscore (_).

02

Numbers are allowed, but not as the first character.

03

Only underscores are allowed as special characters.

04

Variable names are case-sensitive.

Constants

A constant is a value that does not change during program execution.

Types of Constants

01

Integer Constants: 1, 6, 7, 9

02

Real Constants: 322.1, 2.5, 7.0

03

Character Constants: 'a', '$', '@'

C Programming Handbook

CHAPTER 01

Variables, Constants & Keywords

Keywords

auto	double	int	struct
break	long	else	switch
case	return	enum	t ypedef
char	register	extern	union
const	shor t	float	unsigned
continue	signed	for	void
default	sizeof	goto	volatile
do	static	if	while

Our First C Program

```c
#include <stdio.h>

int main() {
    printf("Hello, I am learning C with Samir");
    return 0;
}
```

C Programming Handbook

CHAPTER 01

Beginner Friendly Learning Guide

Basic Structure of a C Program

All C programs must follow a basic structure. Every program starts with a **main()** function and executes instructions present inside it.

Every instruction in C is terminated with a semicolon (**;**).

01

Every program starts execution from the **main()** function.

02

Every statement must end with a semicolon.

03

C language instructions are case-sensitive.

04

Instructions execute in the same order in which they are written.

Comments

Comments are used to explain code and improve readability. They are ignored by the compiler and do not affect program execution.

A single-line comment starts with **//**.

```
// This is a single-line comment.
```

A multi-line comment starts with **/*** and ends with ***/**.

```
/*
This is a multi-line comment
*/
```

Note: Comments are ignored during compilation.

Compilation and Execution

A compiler converts C source code into machine language so that the computer can execute it.

A C program is written as plain text. The compiler performs various checks and finally converts it into an executable file.

C Programming Handbook

CHAPTER 01

Beginner Friendly Learning Guide

Library Functions

C provides many built-in library functions that perform common tasks. For example, **printf()** is used to display output on the screen.

```c
#include <stdio.h>

int main() {
    int i = 10;
    printf("This is %d\n", i);
    // %d for integers, %f for floating point values, %c for characters
    return 0;
}
```

Types of Variables

INT

Integer variables store whole numbers.

```c
int a = 3;
```

FLOAT

Real variables store decimal values.

```c
float a = 7.7;
```

CHAR

Character variables store a single character.

```c
char a = 'b';
```

Receiving Input from the User

To take input from the user and assign it to a variable, we use the **scanf()** function.

Syntax:

```c
scanf("%d", &i);
```

The **&** operator represents the address of the variable. It tells the compiler where the entered value should be stored in memory.

Practice Set

1. Write a C program to calculate the area of a rectangle:
    a. Using hard coded inputs.
    b. Using inputs supplied by the user.
2. Calculate the area of a circle and modify the same program to calculate the volume of a cylinder given its radius and height.
3. Write a program to convert Celsius (Centigrade) temperature to Fahrenheit.
4. Write a program to calculate simple interest for a set of values representing principal, number of years, and rate of interest.

C Programming Handbook

CHAPTER 02

Instructions and Operators

A C program is a set of instructions, just like a recipe that contains instructions for preparing a particular dish.

Types of Instructions

01
Type Declaration Instructions

02
Arithmetic Instructions

03
Control Instructions

Type Declaration Instructions

This is how you declare a variable in C

```
int a;
float b;
char c;
```

Other Variations

Some other variations of this declaration look like this:

```
int a;        // Declare an integer variable 'a'
float b;      // Declare a float variable 'b'
int i = 10;   // Declare 'i' and initialize with 10
int j = i;    // Declare 'j' and initialize with 'i'
int a = 2, b = 3, c = 4, d = 5; // Declare and initialize multiple variables

int j1 = a + j - i; // Valid: use previously defined variables

// Invalid: 'a' is used before declaration
// float b = a + 3;
// float a = 1.1;

// Valid: Assigning the same value to multiple variables
int a, b, c, d;
a = b = c = d = 30; // a, b, c, d all equal to 30
```

C Programming Handbook

CHAPTER 02

Instructions and Operators

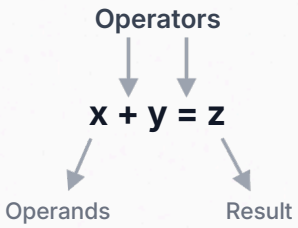
Arithmetic Instructions

Arithmetic instructions perform mathematical operations.

Here are some of the commonly used operators in C language:

Arithmetic Operators

+ Addition	- Subtraction
* Multiplication	/ Division
% Modulus	



Notes on Arithmetic Operators

01

Operands can be int/float. **+ - * /** are arithmetic operators. Result must be assigned to a variable.

```
int z = b * c; // legal
b * c = z; // illegal
```

02

% is the modular division operator.

→ returns the remainder
→ cannot be applied on float
→ sign follows the numerator

```
-5 % 2 = -1
```

03

No operator is assumed between two operands.

```
int i = ab; // invalid
int i = a * b; // valid
```

04

No exponentiation operator in C. Use **pow(x, y)** from *<math.h>* instead (more later).

C Programming Handbook

CHAPTER 02

Instructions and Operators

Type Conversion

An arithmetic operation between two operands follows these rules:

int + int → int

Both operands are int, result is int.

`5 / 2 = 2`

int + float → float

One operand is float, result is float.

`5.0 / 2 = 2.5`

float + float → float

Both are float, result is float.

`5.0 / 2.0 = 2.5`

NOTE

`2 / 5` becomes **0** because both values are `int`, so the fractional part is discarded.

NOTE: In programming, type compatibility is crucial. For `int a = 3.5;`, the float 3.5 is demoted to 3, losing the fractional part because a is an integer. Conversely, for `float a = 8;`, the integer 8 is promoted to 8.0, matching the float type and retaining precision.

Type Promotion & Demotion

When you assign a value to a variable of a different type, C either promotes or demotes it automatically.

```
int a = 3.5;   // 3.5 (float) is demoted to 3 (int) - the fractional part is lost
               // because int is not able to store floats

float a = 8;   // 8 (int) is promoted to 8.0 (float) - the precision is retained
```

Quick Quiz: `int k = 3.0 / 9;` - what is the value of k and why?

Ans: 3.0 / 9 = 0.333. But since k is an `int`, it cannot store floats and the value 0.333 is demoted to **0**.

C Programming Handbook

CHAPTER 02

Instructions and Operators

Operator Precedence in C

Have a look at the below statement: **3*x – 8*y**

Is it **(3x) – (8y)** or **3(x – 8y)**? In C language, simple mathematical rules like BODMAS no longer apply. The answer is provided by **operator precedence & associativity**.

The following table lists the operator priority in C:

Priority	Operators
1st	* / %
2nd	+ -
3rd	=

Operators of higher priority are evaluated first in the absence of parenthesis.

Operator Associativity

When operators of equal priority are present in an expression, the tie is taken care of by associativity.

Left to Right

```
x * y / z  →  (x * y) / z
```

```
x / y * z  →  (x / y) * z
```

***** and **/** follow left to right associativity.

PRO TIP

Always use parentheses in case of confusion to make the order of evaluation explicit and avoid subtle bugs.

Control Instructions

Determines the flow of control in a program. Four types of control instructions in C are:

01 Sequence Control Instructions

02 Decision Control Instructions

03 Loop Control Instructions

04 Case Control Instructions

Practice Set

1. Which of the following is invalid in C?
    a. int a = 1; int b = a;
    b. int v = 3*3;
    c. char dt = '21 dec 2020';
2. What data type will 3.0/8 – 2 return?
3. Write a program to check whether a number is divisible by 97 or not.
4. Explain step by step evaluation of `3*x/y-z+k`, where `x = 2`, `y = 3`, `z = 3`, `k = 1`.
5. 3.0 + 1 will be:
    a. Integer.
    b. Floating point number.
    c. Character.

C Programming Handbook

CHAPTER 03

Conditional Instructions

Sometimes we want to watch comedy videos on YouTube if the day is Sunday.

Sometimes we order junk food if it is our friend's birthday in the hostel.

You might want to buy an umbrella if it's raining, and you have the money.

You order the meal if dal or your favourite bhindi is listed on the menu.

All these are decisions which depend on a condition being met.

In C language too, we must be able to execute instructions on a condition(s) being met.

Decision Making Instructions in C

01
if–else statement

02
switch statement

If-Else Statement

The syntax of an if-else statement in C looks like:

```
if (condition_to_be_checked) {
    // Statements if condition is true
} else {
    // Statements if condition is false
}
```

Code Example

```
int a = 23;
if (a > 18)
{
    printf("you can drive \n");
}
```

Note that else block is not necessary but optional.

C Programming Handbook

CHAPTER 03

Conditional Instructions

Relational Operators in C

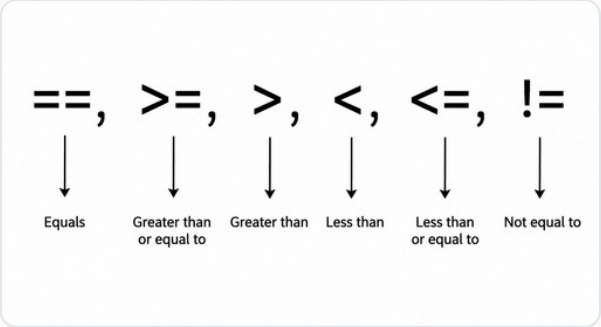
Relational operators are used to evaluate conditions (true or false) inside the if statements.

Some examples of relational operators are:

`==, >=, >, <, <=, !=`

Important note: '=' is used for assignment whereas '==' is used for equality check.

The condition can be any valid expression. In C a non-zero value is considered to be true.



Logical Operators

&&, || and ! are three logical operators in C. These are read as "AND", "OR" and "NOT".

They are used to provide logic to our C programs.

Usage of Logical Operators:

1. **&& (AND)** → is true when both the conditions are true
    a. "1 and 0" is evaluated as false.
    b. "0 and 0" is evaluated as false.
    c. "1 and 1" is evaluated as true.
2. **|| (OR)** → is true when at least one of the conditions is true. (1 or 0 → 1) (1 or 1 → 1)
3. **! (NOT)** → returns true if given false and false if given true
    a. !(3==3) → evaluates to false
    b. !(3>30) → evaluates to true.

As the number of conditions increases, the level of indentation increases. This reduces readability. Logical operators come to rescue in such cases.

Else If Clause

Instead of using multiple if statements, we can also use else if along with it thus forming an if-else if-else ladder.

C Programming Handbook

CHAPTER 03

Conditional Instructions

Code Example

A typical if - else if - else ladder looks like this:

```
if (condition1) {
    // Statements
}
else if (condition2) {
    // Statements
}
else {
    // Statements
}
```

Important Note

01
Using if-else if-else reduces indents.

02
The last "else" is optional.

03
Also there can be any number of "else if".

04
Last else is executed only if all conditions fail.

Operator Precedence

Pr ior it y	Operator
1st	!
2nd	*, /, %
3rd	+, -
4th	<>, <=, >=
5th	==, !=
6th	&&
7th	||
8th	=

Conditional Operators

A shorthand "if – else" can be written using the conditional or ternary operators

```
condition ? expression-if-true : expression-if-false // "?" and ":" are called Ternary Operators
```

C Programming Handbook

CHAPTER 03

Conditional Instructions

Switch Case Control Instruction

switch-case is used when we have to make a choice between number of alternatives for a given variable.

```
switch (integer expression)
{
case c1:
    // code;

case c2:                    // c1, c2 & c3 -> Constants
    // code;                // code -> Any valid C code.

case c3:
    // code;

default:
    // code;
}
```

The value of integer-expression is matched against c1, c2, c3... If it matches any of these cases, that case along with all subsequent "case" and "default" statements are executed.

Quick Quiz

Write a program to find grade of a student given his marks based on below:

Marks Range

90 – 100 ⇒ A
80 – 90 ⇒ B
70 – 80 ⇒ C
60 – 70 ⇒ D
50 – 60 ⇒ E
<50 ⇒ F

Some Important Notes

We can use switch-case statements even by writing cases in any order of our choice (not necessarily ascending).

char values are allowed as they can be easily evaluated to an integer.

A switch can occur within another but in practice this is rarely done.

Practice Set

1. What will be the output of this program?

```c
int a = 10;

if (a = 11)
    printf("I am 11");
else
    printf("I am not 11");
```

2. Write a program to determine whether a student has passed or failed. To pass, a student requires a total of 40% and at least 33% in each subject. Assume there are three subjects and take the marks as input from the user.

3. Calculate income tax paid by an employee to the government as per the slabs mentioned below:

Income Slab	Tax
2.5 - 5.0L	5%
5.0L - 10.0L	20%
Above 10.0L	30%

Note that there is no tax below 2.5L. Take income amount as an input from the user.

4. Write a program to find whether a year entered by the user is a leap year or not. Take year as an input from the user.

5. Write a program to determine whether a character entered by the user is lowercase or not.

6. Write a program to find greatest of four numbers entered by the user.

C Programming Handbook

CHAPTER 04

Loop Control Instruction

Why Loops

Sometimes we want our programs to execute a set of instructions repeatedly over and over again. For example: Printing 1 to 100, first 100 even numbers etc.

Hence loops make it easy for a programmer to tell computer that a given set of instructions must be executed repeatedly.

Types of Loops

Primarily there are three types of loops in C language:

01
while loop

02
do-while loop

03
for loop

We will look into these one by one:

While Loop

```
while (condition is true) {
    // Code
    // The block keeps executing as long as the condition is true
}
```

Example:

```
int i = 0;
while (i<10) {
    printf("the value of i is %d\n", i);
    i++;
}
```

> **Note:** If the condition never becomes false, the while loop keeps getting executed. Such loop is known as an **infinite loop**.

Quick Quiz: Write a program to print natural numbers from 10 to 20 when initial loop counter is initialized to 0.

The loop counter need not be int, it can be float as well.

C Programming Handbook

CHAPTER 04

Loop Control Instruction

Increment and Decrement Operators

i++

i is increased by 1

i--

i is decreased by 1

```
// Decrement i first and then print
printf("--i = %d\n",  --i);

// Print i first and then decrement
printf("i-- = %d\n", i--);
```

- +++ operator does not exist.
- i+=2 is compound assignment which translates to i = i + 2
- Similar to += operator we have other operators like -=, *=, /=, %=.

do-while Loop

The syntax of do-while loop looks like this:

```
do {
    //code;
} while (condition);
```

The do-while loop works very similar to while loop.

In simpler terms we can say:

do-while loop = while loop which executes at least once.

Quick Quiz: Write a program to print first 'n' natural number using do-while loop.

Input

```
4
```

Output

```
1  2  3  4
```

For Loop

The syntax of a typical 'for' loop looks like this:

```
for (initialize; test; increment or decrement)
{
    //code;
}
```

C Programming Handbook

CHAPTER 04

Loop Control Instruction

For Loop: How It Works

Initialize

Setting a loop counter to an initial value.

Test

Checking a condition.

Increment

Updating the loop counter.

Example:

```
for (i=0; i<3; i++){
    printf("%d\n", i);
    printf("\n");
}
//  Output:
//       0
//       1
//       2
```

Quick Quiz: Write a program to print first 'n' natural numbers using for loop.

A Case of Decrementing For Loop

```
for (i=5; i ; i--)
    printf("%d\n",i);
```

This for loop will keep on running until i becomes 0.

The loop runs in following steps:

1. 'i' is initialized to 5.
2. The condition "i" (0 or none) is tested.
3. The code is executed.
4. 'i' is decremented.
5. Condition 'i' is checked & code is executed if it's not 0.
6. And so on until 'i' becomes 0.

Quick Quiz: Write a program to print 'n' natural numbers in reverse order.

C Programming Handbook

CHAPTER 04

Loop Control Instruction

The Break Statement in C

The 'break' statement is used to exit the loop irrespective of whether the condition is true or false.

Whenever a "break" is encountered inside the loop, the control is sent outside the loop.

Let us see this with the help of an example:

```
for (i=0; i<1000; i++){
    printf("%d\n",i);
    if (i==5){
    break;
    }
}
```

Output

```
0   1   2   3   4   5
```

Note

The output of the above program will be below (and not 0 to 100).

The Continue Statement in C

The 'continue' statement is used to immediately move to the next iteration of the loop.

The control is taken to the next iteration thus skipping everything below "continue" inside the loop for that iteration.

Example:

```
#include <stdio.h>

int main() {
    int skip = 5;
    int i = 0;
    while (i < 10) {
        if (i == skip) {
            i++;
            continue; // skips the rest of the loop body for i == 5
        }
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```

C Programming Handbook

CHAPTER 04

Loop Control Instruction

Notes

01

Sometimes, the name of the variable might not indicate the behaviour of the program.

02

'break' statement completely exits the loop.

03

'continue' statement skips the particular iteration of the loop.

Practice Set

1. Write a program to print multiplication table of a given number `n`.
2. Write a program to print multiplication table of 10 in reversed order.
3. A do while loop is executed:
    a. At least once.
    b. At least twice.
    c. At most once.
4. What can be done using one type of loop can also be done using the other two types of loops – true or false?
5. Write a program to sum first ten natural numbers using `while` loop.
6. Write a program to implement program 5 using `for` and `do-while` loop.
7. Write a program to calculate the sum of the numbers occurring in the multiplication table of 8 (consider 8 × 1 to 8 × 10).
8. Write a program to calculate the factorial of a given number using a `for` loop.
9. Repeat 8 using `while` loop.
10. Write a program to check whether a given number is prime or not using loops.
11. Implement 10 using other types of loops.

C Programming Handbook

PROJECT 01

Mini Project

Number Guessing Game

> In this project, we will build a simple **Number Guessing Game** using loops, conditional statements, and a random number generator.

The program generates a random number and asks the player to guess it.

If the player's guess is greater than the actual number, the program should display:

```
Lower number please
```

Similarly, if the user's guess is smaller than the actual number, the program should display:

```
Higher number please
```

The game should continue until the player guesses the correct number.

Once the correct number is guessed, the program should display the total number of guesses taken by the player.

Hint

Use a loop along with a random number generator to repeatedly compare the user's guess with the actual number.

Example Run

```
=== NUMBER GUESSING GAME ===

I have generated a number between 1 and 100.
Can you guess it?
Enter your guess: 50
Higher number please
Enter your guess: 75
Lower number please
Enter your guess: 65
Congratulations! You guessed the number 65
in 3 attempts!
```

C Programming Handbook

CHAPTER 05

Functions and Recursion

Sometimes our program gets bigger in size and it becomes difficult for a programmer to track what each piece of code is doing.

Function is a way to break our code into chunks so that it is possible for a programmer to reuse them.

What is a Function?

A function is a block of code which performs a particular task.

A function can be reused by the programmer in a given program any number of times.

Syntax:

```c
#include <stdio.h>

// Function prototype
void display();

int main() {
int a;           // Variable declaration
display();       // Function call
return 0;        // Return statement
}

// Function definition
void display() {
    printf("hi i am display\n"); // Printing the message
}
```

Function Prototype

A function prototype informs the compiler about a function that will be defined later in the program.

The void keyword indicates that the function does not return any value.

Function Call

A function call instructs the compiler to execute the function's body when the call is made.

Note that program execution starts from the main function and follows the sequence of instructions written.

C Programming Handbook

CHAPTER 05

Functions and Recursion

Function Definition

This part contains the exact set of instructions executed during the function call.

When a function is called from main(), the main function pauses and temporarily suspends. During this time, control transfers to the called function. Once the function finishes executing, main() resumes.

Quick Quiz

Write a program with three functions:

1. Good morning function which prints "good morning".
2. Good afternoon function which prints "good afternoon".
3. Good night function which prints "good night".

main() should call all of these in order 1→2→3

Important Points

01

Execution of a C program starts from main().

02

A C program can have more than one function.

03

Every function gets called directly or indirectly from main().

Types of Functions

Library Functions

Commonly required functions grouped together in a library file on disk.

User Defined Functions

These are the functions declared and defined by the user.

Why Use Functions

01

To avoid rewriting the same logic again and again.

02

To keep track of what we are doing in a program.

03

To test and check logic independently.

C Programming Handbook

CHAPTER 05

Functions and Recursion

Passing Values to Function

We can pass values to a function and can get a value in return from a function. Have a look at the code snippet below:

```
int sum(int a, int b);
```

A function prototype in programming is a declaration of a function that specifies its name, return type, and parameters (if any) but does not include the function body.

The above prototype means that sum is a function which takes values 'a' (of type int) and 'b' (of type int) and returns a value of type int.

Function definition of sum can be:

```
int sum (int a, int b) { // a and b are parameters
    int c;
    c = a+b;
    return c;
}
// Now we can call sum (2,3); from main to get 5 in return.
// Here 2 & 3 are arguments.
int d = sum (2,3); // d becomes 5
```

Parameters vs Arguments

Parameters

The values or variable placeholders in the function definition.

Example: a & b

Arguments

The actual values passed to the function to make a call.

Example: 2 & 3

Important Notes

01

A function can return only one value at a time.

02

If the passed variable is changed inside the function, it doesn't change the value in the calling function.

Call by Value: Example

'change' is a function which pretends to change 'a' to 77. Now if we call it from main like this:

```
int change(int a) {
    a = 77;             // Misnomer
    return 0;
}
```

```
int b = 22;
change(b);              // The value of b remains 22
printf("b is %d", b); // Prints "b is 22"
```

This happens because a **copy of 'b'** is passed to the change function. The original variable b in main() is not affected.

Quick Quiz

Quick Quiz: Use the library function to calculate the area of a square with side a.

Hint

Use `pow(a, 2)` from `<math.h>` to calculate a^2.

Formula

Area of square = side × side = a^2

Recursion

A function defined in C can call itself. This is called **recursion**. A function calling itself is also called a 'recursive' function.

Example: A very good example of recursion is factorial.

Factorial(n) = 1 × 2 × 3 ... × n

Factorial(n) = 1 × 2 × 3 ... × (n-1) × n

Factorial(n) = Factorial(n-1) × n

Since we can write factorial of a number in terms of itself, we can program it using recursion.

```
int factorial(int x) {
    int f;
    if (x == 0 || x == 1) {
        return 1; //aprogram to calculate factorial using recursion
    } else {
        f = x * factorial(x - 1);
        return f;
    }
}
```

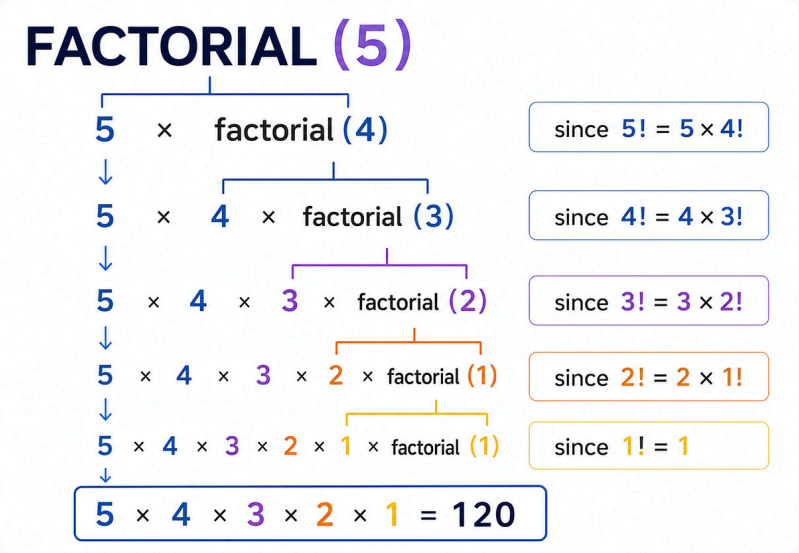
C Programming Handbook

CHAPTER 05

Functions and Recursion

Dry Run of Recursive Factorial Program

Let's trace through how `factorial(5)` is evaluated step by step:



Important Notes

01

Recursion is often a direct way to implement certain algorithms, but not always the most efficient. It is particularly suited for problems divisible into smaller subproblems like factorial computation or tree traversal.

02

The condition in a recursive function that stops further recursion is called the **base case**. It is crucial as it prevents infinite recursion and ensures the function terminates correctly.

03

Sometimes, due to an oversight by the programmer, a recursive function can continue to run indefinitely without reaching a base case, potentially causing a **stack overflow** or memory error.

C Programming Handbook

CHAPTER 05

Functions & Recursion

Practice Set

1. Write a program using function to find average of three numbers.
2. Write a function to convert Celsius temperature into Fahrenheit.
3. Write a function to calculate force of attraction on a body of mass `'m'` exerted by earth. Consider `g = 9.8m/s²`.
4. Write a program using recursion to calculate `nth` element of Fibonacci series.
5. What will the following line produce in a C program?

```c
int a = 4;
printf("%d %d %d \n", a, ++a, a++);
```

6. Write a recursive function to calculate the sum of first `'n'` natural numbers.
7. Write a program using function to print the following pattern (first `n` lines):

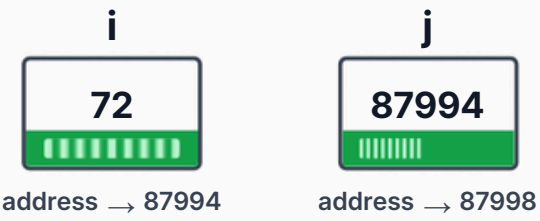
```
*
***
*****
```

C Programming Handbook

CHAPTER 06

Pointers

A pointer is a variable which stores the address of another variable.



The "Address Of" (&) Operator

The address of operator is used to obtain the address of a given variable.

If you refer to the diagrams above:

&i → 87994

Returns the memory address where variable i is stored.

&j → 87998

Returns the memory address where variable j is stored.

Format specifier for printing pointer address is **'%p'**.

The 'Value at Address' Operator (*)

The value at address or * operator is used to obtain the value present at a given memory address. It is denoted by *.

***(&i) = 72**

Dereferences the address of i to get its value: 72.

***(&j) = 87994**

Dereferences the address of j to get its value: 87994.

How to Declare a Pointer?

A pointer is declared using the following syntax:

int *j

Declare a variable j of type int-pointer.

j = &i

Store the address of i in j.

Just like pointer of type integer, we also have pointers to char, float etc.

C Programming Handbook

CHAPTER 06

Pointers

Pointer Type Declarations

```
int  *in_ptr;  //pointer to integer
char *ch_ptr;  //pointer to character
float *fl_ptr; //pointer to float
```

Although it's a good practice to use meaningful variable names, we should be very careful while reading and working on programs from fellow programmers.

A Program to Demonstrate Pointers

```
#include <stdio.h>

int main (){
    int i = 8;
    int *j;
    j = &i;
    printf("add    i=    %u\n",&i);
    printf("add     i=     %u\n",j);
    printf("add    j=    %u\n",&j);
    printf("value    i=    %d\n",i);
    printf("value i= %d\n",*(&i));
    printf("value   i=   %d\n",*j);
    return 0;
}
```

Output

Address outputs

```
add i= 87994
add i= 87994
add j= 87998
```

Value outputs

```
value i= 8
value i= 8
value i= 8
```

This program sums it all. If you understand it, you have got the idea of pointers.

Pointer to a Pointer

Just like 'j' is pointing to 'i' or storing the address of 'i', we can have another variable k which can further store the address of 'j'. What will be the type of 'k'?

```
int **k;
k = &j;
```

C Programming Handbook

Pointers

Pointer to Pointer: Memory Layout

We can even go further one level and create a variable 'l' of type int*** to store the address of 'k'. We mostly use int* and int** sometimes in real world programs.

Types of Function Call

Based on the way we pass arguments to the function, function calls are of two types:

Call by Value

Sending the **values** of arguments to the function.

Call by Reference

Sending the **addresses** of arguments to the function.

Call by Value

Here the values of the arguments are passed to the function. Consider this example:

```
int c = sum (3,4); //assume x=3 and y=4
```

If sum is defined as sum (int a, int b), the values 3 and 4 are copied to a and b. Now even if we change a and b, nothing happens to the variables x and y.

This is call by value. In C we usually make a call by value.

Call by Reference

Here the address of the variables is passed to the function as arguments.

Now since the addresses are passed to the function, the function can now modify the value of a variable in calling function using * and & operators.

Call by Reference: Example

This function is capable of swapping the values passed to it. If a = 3 and b = 4 before a call to swap(a, b), then a = 4 and b = 3 after calling swap.

```
void swap (int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

```
int main(){
    int a = 3;
    int b = 4;  // a is 3 and b is 4
    swap(&a, &b);
    return 0;   //now a is 4 and b is 3
}
```

Summary

Call by Value

A copy of the argument is passed. Changes inside the function do **not** affect the original variable.

Call by Reference

The address is passed. Changes inside the function **do** affect the original variable.

*** Operator**

Dereference operator: gets the value stored at a memory address.

& Operator

Address-of operator: gets the memory address of a variable.

Practice Set

1. Write a program to print the address of a variable. Use this address to get the value of the variable.
2. Write a program having a variable `i`. Print the address of `i`. Pass this variable to a function and print its address. Are these addresses the same? Why?
3. Write a program to change the value of a variable to ten times its current value.
4. Write a function and pass the value by reference.
5. Write a program using a function which calculates the sum and average of two numbers. Use pointers and print the values of sum and average in `main()` .
6. Write a program to print the value of a variable `i` by using a `pointer to pointer` type variable.
7. Try problem 3 using call by value and verify that it does not change the value of the variable.

C Programming Handbook

CHAPTER 07

Ar rays

An array is a collection of similar elements. Array allows a single variable to store multiple values.

Syntax

```
int   marks[90];        // integer array
char name[20];          // character array or string
float percentile[90];   // float array
```

The values can now be assigned to make array like this:

```
marks[0] = 33;
marks[1] = 12;
```

Note: It is very important to note that the array index starts with 0.

Array in Memory: Marks[90]

Total = 90 elements

Accessing Elements

Elements of an array can be accessed using:

```
scanf("%d", &marks[0]);  // input first value
printf("%d", marks[0]);  // output first value of the array
```

Quick Quiz: Write a program to accept marks of five students in an array and print them on the screen.

Initialization of an Array

There are many other ways in which an array can be initialized.

```
int cgpa[3] = {9, 8, 8}; // arrays can be initialized while declaration
float marks[] = {33, 40};
```

C Programming Handbook

CHAPTER 07

Ar rays

Arrays in Memory

Consider this array:

```
int arr[3] = {1, 2, 3}; // 1 integer = 4 bytes
```

This will reserve 4 × 3 = 12 bytes in memory (4 bytes for each integer).

Pointer Arithmetic

A pointer can be incremented to point to the next memory location of that type.

Consider this example:

```
int i = 32;
int *a = &i;  // a = 87994
a++;          //address of i or value of a = 87998

char a = 'A';
char *b = &a;  // a= 87994
b++;           // now a = 87995

float i = 1.7;
float *a = &i; // now a = 87994
a++;           // now a = 87998
```

Following operations can be performed on a pointer:

01
Addition of a number to a pointer.

02
Subtraction of a number from a pointer.

03
Subtraction of one pointer from another.

04
Comparison of two pointer variables.

Quick Quiz: Try these operations on another variable by creating pointers in a separate program. Demonstrate all the four operations.

C Programming Handbook

CHAPTER 07

Ar rays

Accessing Array Using Pointers

Consider this array:

If ptr points to index 0, ptr++ will point to index 1 & so on...

This way we can have an integer pointer pointing to first element of the array like this:

```
int *ptr = &arr[0]; // or simply: arr
ptr++;
*ptr  // will have 9 as its value
```

Passing Array to Functions

Array can be passed to the functions like this:

```
printArray(arr, n);              // function call
void printArray(int *i, int n);  // function prototype
// or
void printArray(int i[], int n);
```

Multidimensional Arrays

An array can be of 2 dimension / 3 dimension / n dimensions.

A 2 dimensional array can be defined like this:

```
int  arr[3][2]  =  {{1,  4},
{7, 9},  {11, 22}};
```

We can access the elements of this array as arr[0][0], arr[0][1] & so on ...

C Programming Handbook

CHAPTER 07

Ar rays

2-D Arrays in Memory

A 2d array like a 1d array is stored in contiguous memory blocks like this:

Quick Quiz

Create a 2-d array by taking input from the user. Write a display function to print the content of this 2-d array on the screen.

Summary

Array Declaration

```
int marks[90];
```

Declares an integer array of size 90. Index starts from 0.

Initialization

```
int a[3] = {1, 2, 3};
```

Arrays can be initialized at the time of declaration.

Pointer Access

```
int *ptr = arr;
```

Array name is a pointer to the first element. Use ptr++ to traverse.

2D Array

```
int a[3][2];
```

Stored in contiguous memory. Access via a[row][col].

Pass to Function

```
void f(int *i, int n);
```

Arrays are passed as pointers to functions.

Memory

Each int takes 4 bytes. An int[3] reserves 12 bytes in contiguous memory locations.

C Programming Handbook

CHAPTER 07

Ar rays

Practice Set

1. Create an array of 10 numbers. Verify using pointer arithmetic that `(ptr+2)` points to the third element where `ptr` is a pointer pointing to the first element of the array.
2. If `S[3]` is a 1-D array of integers then `*(S+3)` refers to the third element:
    i. True.
    ii. False.
    iii. Depends.
3. Write a program to create an array of 10 integers and store multiplication table of 5 in it.
4. Repeat problem 3 for a general input provided by the user using `scanf` .
5. Write a program containing a function which reverses the array passed to it.
6. Write a program containing functions which counts the number of positive integers in an array.
7. Create an array of size 3 × 10 containing multiplication tables of the numbers 2, 7 and 9 respectively.
8. Repeat problem 7 for a custom input given by the user.
9. Create a three-dimensional array and print the addresses of its elements in increasing order.

C Programming Handbook

CHAPTER 08

Strings

A string is a 1-D character array terminated by a null character ('\0').

A null character is used to denote the termination of a string. Characters are stored in contiguous memory locations.

Initializing Strings

Since string is an array of characters, it can be initialized as follows:

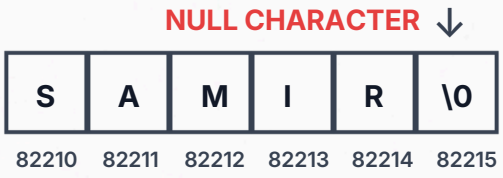
```c
char s[] = {'S', 'A', 'M', 'I', 'R', '\0'};
```

There is another shortcut for initializing string in C language:

```c
char s[] = "SAMIR"; // C adds a null character automatically.
```

Strings in Memory

A string is stored just like an array in the memory as shown below.



Contiguous Blocks in Memory

Quick Quiz: Create a string using double quotes and print its content using a loop.

C Programming Handbook

CHAPTER 08

Strings

Printing Strings

A string can be printed character by character using printf and %c.

But there is another convenient way to print strings in C:

```c
char st[] = "SAMIR";
printf("%s", st); // print the entire string.
```

Taking String Input from the User

We can use %s with scanf to take string input from the user:

```c
char st[50];
scanf ("%s", st);
```

scanf automatically adds a null character when the enter key is pressed.

Note 01

The string should be short enough to fit into the ar ray.

Note 02

scanf cannot be used to input multi-word strings with spaces.

gets() and puts()

gets() is a function which can be used to receive a multi-word string.

```c
char st[30];
gets(st); // The entered string is stored in st
```

Multiple gets() calls will be needed for multiple strings.

Likewise, puts can be used to output a string.

```c
puts(st); // Prints the string & places the cursor on the next line
```

C Programming Handbook

CHAPTER 08

Strings

Declaring a String Using Pointers

We can declare strings using pointers.

```
char *ptr = "samir";
```

This tells the compiler to store the string in memory and the assigned address is stored in a char pointer.

Note 01

Once a string is defined using `char st[] = "samir"`, it cannot be reinitialized to something else.

Note 02

A string defined using pointers **can** be reinitialized.

```
ptr = "Rohan";
```

Standard Library Functions for Strings

C provides a set of standard library functions for string manipulation. These are declared under **<string.h>** header file.

Some of the most commonly used string functions are:

strlen()

Counts the number of characters in a string, excluding the null ('\0') character.

strcpy()

Copies the content of the second string into the first string passed to it.

strcat()

Concatenates two strings. The result is stored in the first string. No space is added between them.

strcmp()

Compares two strings. Returns 0 if equal, negative or positive based on ASCII difference.

str len()

This function is used to count the number of characters in the string excluding the null ('\0') character.

```
int length = strlen(st);
```

strcpy()

This function is used to copy the content of second string into first string passed to it.

```
char source[] = "samir";
char target[30];
strcpy (target, source); // target now contains "samir"
```

Target string should have enough capacity to store the source string.

strcat()

This function is used to concatenate two strings.

```
char s1[12] = "hello";
char s2[] = "samir";
strcat(s1, s2); // s1 now contains "hellosamir" <no space in between>
```

C Programming Handbook

CHAPTER 08

Strings

strcmp()

This function is used to compare two strings. It returns 0 if the strings are equal, a negative value if the first string's mismatching character's ASCII value is less than the second string's corresponding mismatching character, and a positive value otherwise.

```
strcmp("far",  "joke");  // Negative value
strcmp("joke", "far");   // Positive value
```

Summar y

String Definition

A 1-D char array terminated by '\0'. Stored in contiguous memory blocks.

Initialization

`char s[] = "SAMIR";`

C automatically appends '\0' at the end.

Printing

Use `printf("%s", st);` to print entire string at once.

scanf vs gets

Use **scanf** for single-word. Use **gets** for multi-word strings with spaces.

Pointer Strings

`char *ptr = "samir";`

Can be reinitialized unlike array strings.

Header File

All string functions require **<string.h>** to be included.

strlen()

Returns character count excluding '\0'.

strcpy()

Copies source into target. Target must have enough size.

strcat()

Appends s2 to s1. No space is added between them.

strcmp()

Returns 0 (equal), negative (s1 < s2), or positive (s1 > s2).

Practice Set

1. Which of the following is used to appropriately read a multi-word string.
    1. gets()
    2. puts()
    3. printf()
    4. scanf()
2. Write a program to take string as an input from the user using `%c` and `%s` and confirm that the strings are equal.
3. Write your own version of `strlen` function from `<string.h>`.
4. Write a function `slice()` to slice a string. It should change the original string such that it is now the sliced string. Take `m` and `n` as the start and ending position for slice.
5. Write your own version of `strcpy` function from `<string.h>`.
6. Write a program to encrypt a string by adding 1 to the ASCII value of its characters.
7. Write a program to decrypt the string encrypted using encrypt function in problem 6.
8. Write a program to count the occurrence of a given character in a string.
9. Write a program to check whether a given character is present in a string or not.

C Programming Handbook

CHAPTER 09

Structures

Arrays & Strings

Hold **similar** data (int, float, char).

Structures

Can hold **dissimilar** data of different types together.

Creating a Structure

A C structure can be created as follows:

```
struct employee
{
    int code;    // This declares a new user defined data type!
    float salary;
    char name[10];
}; // semicolon is important
```

We can use this user defined data type as follows:

```
struct employee e1; // creating a structure variable
strcpy(e1.name, "samir");
e1.code   = 100;
e1.salary = 71.22;
```

So, a structure in C is a collection of variables of different types under a single name.

Quick Quiz

Write a program to store the details of 3 employees from user defined data. Use the structure declared above.

Why Use Structures?

We can create the data types in the employee structure separately but when the number of properties in a structure increases, it becomes difficult for us to create data variables without structures. In a nutshell:

a

Structures keep the data organized.

b

Structures make data management easy for the programmer.

Array of Structures

Just like an array of integers, an array of floats and an array of characters, we can create an array of structures.

```
struct employee facebook[100]; // an array of structures
// We can access the data using:
facebook[0].code = 100;
facebook[1].code = 101;
// And so on
```

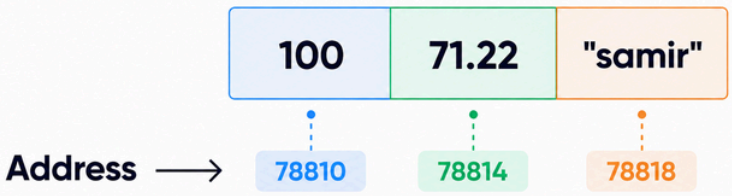
Initializing Structures

Structures can also be initialized as follows:

```
struct employee samir = {100, 71.22, "samir"};
struct employee shubh = {0}; // All elements set to 0
```

Structures in Memory

Structures are stored in contiguous memory locations. For the structure 'e1' of type struct employee, memory layout looks like this:



In an array of structures, these employee instances are stored adjacent to each other.

C Programming Handbook

CHAPTER 09

Structures

Pointer to Structures

A pointer to structures can be created as follows:

```
struct employee *ptr;
ptr = &e1;
// now we can print structure elements using:
printf("%d", (*ptr).code);
```

Arrow Operator

Instead of writing (*ptr).code, we can use arrow operator to access structure properties as follows:

```
(*ptr).code
// or
ptr->code
// here -> is known as the arrow operator.
```

(*ptr).code

Dereference pointer then access member using dot operator.

ptr->code

Shorthand arrow operator. Cleaner and preferred way to access struct members via pointer.

Passing Structure to a Function

A structure can be passed to a function just like any other data type.

```
void show(struct employee e); // function prototype
```

Pass by Value

A copy of the entire structure is passed. Changes inside the function do not affect the original.

Pass by Reference

Pass a pointer to the structure. Changes inside the function **do** affect the original struct.

typedef Keyword

We can use the 'typedef' keyword to create an alias name for data types in C.

'typedef' is more commonly used with structures.

```
struct Complex
{
    float real;
    float img; // struct complex c1,c2, for defining complex numbers
};
```

```
typedef struct Complex
{
    float real;
    float img; // ComplexNo c1,c2, for defining complex numbers
} ComplexNo;
```

Example Usage

Using the typedef alias, you can declare complex number variables more succinctly:

```
ComplexNo c1, c2;
```

Instead of having to write `struct Complex c1, c2;` every time, typedef lets you use the shorter alias `ComplexNo` directly.

Practice Set

1. Create a two-dimensional vector using structures in C.
2. Write a function `sumVector` which returns the sum of two vectors passed to it. The vectors must be two-dimensional.
3. Twenty integers are to be stored in memory. What will you prefer - Array or structure?
4. Write a program to illustrate the use of arrow operator `->` in C.
5. Write a program with a structure representing a complex number.
6. Create an array of 5 complex numbers created in Problem 5 and display them with the help of a display function. The values must be taken as an input from the user.
7. Write problem 5's structure using `typedef` keywords.
8. Create a structure representing a bank account of a customer. What fields did you use and why?
9. Write a structure capable of storing date. Write a function to compare those dates.
10. Solve problem 9 for time using `typedef` keyword.

C Programming Handbook

CHAPTER 10

FileInput / Output

File I/O

The random-access memory is volatile, and its content is lost once the program terminates. In order to persist the data forever we use files.

A file is data stored in a storage device.

A C program can talk to the file by reading content from it and writing content to it.

File Pointer

A "FILE" is a structure which needs to be created for opening the file.

A file pointer is a pointer to this structure of the file. (FILE pointer is needed for communication between the file and the program).

A FILE pointer can be created as follows:

```
FILE *ptr;
ptr = fopen("filename.ext", "mode");
```

File Opening Modes in C

C offers the programmers to select a mode for opening a file.

Following modes are primarily used in C File I/O.

- `r` -> open for reading
- `rb` -> open for reading in binary
- `w` -> open for writing // If the file exists, the contents will be overwritten
- `wb` -> open for writing in binary
- `a` -> open for append // If the file does not exist, it will be created

C Programming Handbook

CHAPTER 10

FileInput / Output

Types of Files

Primarily, there are two types of files:

1. Text files (.txt, .c)
2. Binary files (.jpg, .dat)

Reading a File

A file can be opened for reading as follows:

```
FILE *ptr;
ptr = fopen("samir.txt", "r");
int num;
```

Let us assume that "samir.txt" contains an integer we can read that integer using:

```
fscanf(ptr, "%d", &num); // fscanf is file counterpart of scanf
```

This will read an integer from file in Num variables.

> **Quick Quiz:** Modify the program above to check whether the file exists or not before opening the file.

Closing the File

It is very important to close the file after read or write. This is achieved using fclose as follows:

```
fclose(ptr);
```

This will tell the compiler that we are done working with this file and the associated resources could be freed.

Write to a File

We can write to a file in a very similar manner like we read the file:

```
FILE *fptr;
fptr = fopen("samir.txt", "w");
int num = 432;
fprintf(fptr, "%d", num);
fclose(fptr);
```

fgetc() and fputc()

fgetc and fputc are used to read and write a character from / to a file.

```
fgetc(ptr);    // used to read a character from file
fputc('c', ptr);    // used to write character 'c' to the file
```

EOF : End of File

fgetc returns EOF when all the characters from a file have been read. So, we can write a check like below to detect end of file:

```
while(1)
{
    ch = fgetc(ptr); // when all the content of a file has been read break the loop!
    if (ch == EOF)
    {
        break;
    }
    // code
}
```

Practice Set

1. Write a program to read three integers from a file.
2. Write a program to generate multiplication table of a given number in text format. Make sure that the file is readable and well formatted.
3. Write a program to read a text file character by character and write its content twice in separate file.
4. Take name and salary of two employees as input from the user and write them to a text file in the following format:
    i. Name1, 3300
    ii. Name2, 7700
5. Write a program to modify a file containing an integer to double its value.

C Programming Handbook

PROJECT 02

Mini Project

Snake, Water, Gun

> In this project, we will build a simple **Snake, Water, Gun** game using conditional statements and random number generation.

Snake, Water, Gun (or Rock, Paper, Scissors) is a game that most of us have played during our school days. It is a simple game where the player competes against the computer.

Write a C program capable of playing this game with the user.

The program should accept the user's choice and then randomly generate the computer's choice before displaying the result.

Hint

Use a random number generator to select the computer's move and conditional statements to determine the winner.

Example Run

```
=== SNAKE WATER GUN ===

Choose:
1. Snake
2. Water
3. Gun

Your choice: Snake

Computer chose: Water

Congratulations!
You Win!
```

C Programming Handbook

CHAPTER 11

Dynamic Memory Allocation

Dynamic Memory Allocation

C is a language with some fixed rules of programming. For example: Changing the size of an array is not allowed.

Dynamic memory allocation is a way to allocate memory to a data structure during the runtime. We can use DMA function available in C to allocate and free memory during runtime.

Functions for DMA in C

Following function are available in C to perform dynamic memory allocation:

1. malloc()
2. calloc()
3. free()
4. realloc()

malloc() Function

malloc stands for memory allocation. It takes number of bytes to be allocated as an input and returns a pointer of type void.

Syntax:

```
ptr = (int*)malloc(30 * sizeof(int));
```

The expression returns a null pointer if the memory cannot be allocated.

> **Quick Quiz:** Write a program to create a dynamic array of 5 floats using malloc().

calloc() Function

calloc stands for continuous allocation. It initializes each memory block with a default value of 0.

Syntax:

```
ptr = (float*)calloc(30, sizeof(float));
//allocates contiguous space in memory for 30 blocks (floats)
```

If the space is not sufficient, memory allocation fails, and a NULL pointer is returned.

> **Quick Quiz:** Write a program to create an array of size n using calloc where n is an integer entered by the user.

free() Function

We can use free() function to deallocate the memory. The memory allocated using malloc/malloc is not deallocated automatically.

Syntax:

```
free(ptr); //memory of ptr is released.
```

> **Quick Quiz:** Write a program to demonstrate the usage of free() with malloc().

realloc() Function

Sometimes the dynamically allocated memory is insufficient or more than required. realloc is used to allocate memory of new size using the previous pointer and size.

Syntax:

```
ptr = realloc(ptr, newsize);
ptr = realloc(ptr, 3 * sizeof(int));
```

C Programming Handbook

CHAPTER 11

Dynamic Memory Allocation

Practice Set

1. Write a program to dynamically create an array of size 6 capable of storing 6 integers.
2. Use the array in Problem 1 to store 6 integers entered by the user.
3. Solve Problem 1 using `calloc()`.
4. Create an array dynamically capable of storing 5 integers. Now use `realloc` so that it can now store 10 integers.
5. Create an array of multiplication table of 7 up to 10 (7 × 10 = 70). Use `realloc` so that it can store the multiplication table up to 15 (7 × 15 = 105).
6. Attempt Problem 4 using `calloc()`.