C Programming Notes by CodeWithHarry

What is Programming?
Computer Programming is a medium for us to communicate with computers. Just like we use ‘Hindi’ or ‘English’ to communicate with each other, programming is a way for us to deliver our instructions to the computer.

What is C?
C is a programming language.
C is one of the oldest and finest programming languages.
C was developed by Dennis Ritchie at AT&T's Bell Labs, USA in 1972.

Uses of C
C language is used to program a wide variety of systems. Some of the uses of C are as follows:

1. Major parts of Windows, Linux, and other operating systems are written in C.

2. C is used to write driver programs for devices like Tablets, printers, etc.

3. C language is used to program embedded systems where programs need to run faster in limited memory (Microwave, Cameras, etc.)

4. C is used to develop games, an area where latency is very important, i.e., computer has to react quickly on user input.
Chapter 1: Variables, Constants & Keywords

Variables

A variable is a container which stores a 'value'. In kitchen, we have containers storing rice, dal, sugar etc. Similar to that variables in C stores value of a constant. Example:

```
A = 3;  // a is assigned "3"
b = 4.7;  // b is assigned "4.7"
c = 'A';  // c is assigned 'A'
```

Rules for naming variables in C

1. First character must be an alphabet or underscore (_).
2. No commas, blanks allowed.
3. No special symbol other than _ allowed.
4. Variable names are case sensitive.

We must create meaningful variable names in our programs. This enhances readability of our programs.

Constants

An entity whose value doesn't change is called as a constant.

A variable is an entity whose value can be changed.
Types of Constants

Primarily, there are three types of constants:

1. Integer Constant → -1, 6, 7, 9
2. Real Constant → -322.1, 2.5, 7.0
3. Character Constant → 'a', '$', '@' (Must be enclosed within single inverted commas)

Keywords

These are reserved words, whose meaning is already known to the compiler. There are 32 keywords available in C:

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

Our First C Program

#include <stdio.h>

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

File: first.c
Basic Structure of a C Program

All C programs have to follow a basic structure. A C program starts with a main function and executes instructions present inside it. Each instruction is terminated with a semicolon (;).

There are some rules which are applicable to all the C programs:

1. Every program’s execution starts from main() function.
2. All the statements are terminated with a semicolon.
3. Instructions are case-sensitive.
4. Instructions are executed in the same order in which they are written.

Comments

Comments are used to clarify something about the program in plain language. It is a way for us to add notes to our program. There are two types of comments in C:

1. Single line comment: // This is a comment
2. Multi-line comment: /* This is a multi-line comment */

Comments in a C program are not executed and are ignored.
Compilation and Execution

```
C compiler
```

A compiler is a computer program which converts a C program into machine language so that it can be easily understood by the computer.

A C program is written in plain text. This plain text is combination of instructions in a particular sequence. The compiler performs some basic checks and finally converts the program into an executable.

Library Functions

The C language has a lot of valuable library functions which is used to carry out certain tasks. For instance `printf` function is used to print values on the screen.

```
printf("This is \%d", i);
```

- `%d` for integers
- `%f` for real values
- `%c` for characters
Types of Variables

1. Integer variable → int a = 3;  
   Wrong as 7.7 is real
2. Real variable → int a = 7.7; float a = 7.7;
3. Character Variable → char a = 'B';

Receiving input from the User

In order to take input from the user and assign it to a variable, we use `scanf` function.

Syntax for using `scanf`:

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

This & is important!

& is the "address of" operator and it means that the supplied value should be copied to the address which is indicated by variable i.
Chapter 1 - Practice Set

Q1. Write a C program to calculate area of a rectangle:
(a) Using hard coded inputs
(b) Using inputs supplied by the User

Q2. Calculate the area of a circle and modify the same program to calculate the volume of a cylinder given its radius and height.

Q3. Write a program to convert Celsius (Centigrade degrees temperature to Fahrenheit)

Q4. Write a program to calculate simple interest for a set of values representing principal, no of years and rate of interest.
Chapter 2: Instructions and Operators

A C program is a set of instructions. Just like a recipe - which contains instructions to prepare a particular dish.

Types of Instructions

1. Type declaration Instruction
2. Arithmetic Instruction
3. Control Instruction

Type declaration Instruction

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

Other Variations:

```c
int i=10; int j=i; int a=2
int y=a+j-1;

float b = a+3; float a=1.1 => ERROR! as we are trying to use a before defining it.
```

```c
int a, b, c, d;
a = b = c = d = 30; => Value of a, b, c & d will be 30 each.
```
Arithmetic Instructions

```
int i = (3 * 2) + 1
```

Operators:
- `int` is an integer
- `+` is addition
- `-` is subtraction
- `*` is multiplication
- `/` is division
- `%` is modular division

Operands can be `int`, `float`, etc.
```
+ - * / are arithmetic operators
```

```
int b = 2, c = 3;
int z; z = b * c;  // legal
int z; b * c = z;  // illegal (not allowed)
```

% — Modular division operator
% — Returns the remainder
. — Cannot be applied on float
/. — Sign is same as of numerator (-5/.2 = -1)

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

Note:
- No operator is assumed to be present
  - int i = a b → Invalid
  - int i = a * b → valid

27. There is no operator to perform exponentiation in C.
However, we can use `pow(x, y)` from `<math.h>` (More later)
Type Conversion

An Arithmetic operation between

Int and Int → Int
Int and float → Float
Float and float → Float

\[ \frac{5}{2} \rightarrow 2 \quad 5.0/2 \rightarrow 2.5 \]

2/5 → 0 \[ 2.0/5 \rightarrow 0.4 \]

\{ Important!! \}

Note:

\[ \text{Int } a = 3.5; \ \text{In this case } 3.5 \text{ (float) will be}\]
\[ \text{demoted to } 3 \text{ (int) because } a \text{ is not able to store floats.} \]

\[ \text{float } a = 8; \ \text{a will store } 8.0 \]
\[ 8 \rightarrow 8.0 \text{ (promotion to float)} \]

Quick Quiz:

8. \[ \text{int } k = 3.0/9 \quad \text{Value of } k? \text{ and why?} \]

5. \[ 3.0/9 = 0.333 \quad \text{but since } k \text{ is an int, it cannot store floats & value } \]
\[ 0.33 \text{ is demoted to 0.} \]
Operator precedence in C

$3 * x - 8 * y$ is $(3x) - (8y)$ or $3(x - 8y)$?

In C language, simple mathematical rules like BODMAS, no longer applies.

The answer to the above question is provided by operator precedence & associativity.

Operator precedence $\div$ The following table lists the operator priority in C

<table>
<thead>
<tr>
<th>Priority</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1^{st}$</td>
<td>$+$, $-$, $%$</td>
</tr>
<tr>
<td>$2^{nd}$</td>
<td>$*$, $/$</td>
</tr>
<tr>
<td>$3^{rd}$</td>
<td>$\div$</td>
</tr>
</tbody>
</table>

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

Operator Associativity $\div$ When operators of equal priority are present in an expression, the tie is taken care of by associativity.

$x * y / z \Rightarrow (x * y) / z$

$x / y * z \Rightarrow (x / y) * z$

$\times$, $/$ follows left to right associativity
Control Instructions
Determine the flow of control in a program.

Four types of control instructions in C are:

1. Sequence Control Instruction
2. Decision Control Instruction
3. Loop Control Instruction
4. Case Control Instruction
Chapter 2 – Practice Set

Q1 Which of the following is invalid in C?
(i) int a; b=a;
(ii) int v = 3 & 3;
(iii) char dt = '21 Dec 2020';

Q2 What data type will `3.0/8 - 2` return?

Q3 Write a program to check whether a number is divisible by 97 or not.

Q4 Explain step by step evaluation of `3 + (y - z + k)`  
   where `x = 2`, `y = 3`, `z = 3`, `k = 1`.

Q5 `3.0 + 1` will be:
(a) Integer
(b) Floating Point Number
(c) Character
Chapter 3 - 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 favorite bhaendi 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

→ If - else statement
→ Switch statement

If - else statement

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

```
if (condition to be checked) {
    Statements - if - condition - true;
} else {
    Statements - if - condition - false;
}
```
Code example:

```c
int a = 23;
if (a > 18) {
    printf(“You can drive”);
}
```

Note that else block is not necessary but optional.

Relational operators in C

Relational operators are used to evaluate conditions (true or false) inside the if statements. Some examples of relational operators are:

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

equals, greater than or equal to, not equal to

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:

(i) \( \& \& \rightarrow \text{AND} \rightarrow \text{is true when both the conditions are true} \)

"1 and 0" is evaluated as false.
"0 and 0" is evaluated as false.
"1 and 1" is evaluated as true.

(ii) \( \lor \rightarrow \text{OR} \rightarrow \text{is true when at least one of the conditions is true} \)

\((1 \lor 0 \rightarrow 1)(1 \lor 1 \rightarrow 1)\)

(iii) \( ! \rightarrow \text{returns true if given false and false if given true} \)

\( ! (3 == 3) \rightarrow \text{evaluates to false} \)
\( ! (3 > 30) \rightarrow \text{evaluates to true} \)

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 if thus forming an if-else if-else ladder.

if \( \& \& \)

// Statements:

else if \( \& \& \)

else \( \& \& \)

else \( \& \& \)
Using `if - else if - else` reduces indents
The last "else" is optional
Also there can be any number of "else if"
Last else is executed only if all conditions fail.

Operator precedence

<table>
<thead>
<tr>
<th>Priority</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>!</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt;</td>
<td>*, /, %</td>
</tr>
<tr>
<td>3&lt;sup&gt;rd&lt;/sup&gt;</td>
<td>+, -</td>
</tr>
<tr>
<td>4&lt;sup&gt;th&lt;/sup&gt;</td>
<td>&lt;, &gt;, &lt;=, &gt;=</td>
</tr>
<tr>
<td>5&lt;sup&gt;th&lt;/sup&gt;</td>
<td>==, !=</td>
</tr>
<tr>
<td>6&lt;sup&gt;th&lt;/sup&gt;</td>
<td>&amp; &amp;</td>
</tr>
<tr>
<td>7&lt;sup&gt;th&lt;/sup&gt;</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>8&lt;sup&gt;th&lt;/sup&gt;</td>
<td>=</td>
</tr>
</tbody>
</table>

Conditional Operators
A short hand "if - else" can be written using the conditional or ternary operators

```
Condition ? expression-if-true : expression-if-false
```

Ternary operators
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 \rightarrow Constants
    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:
  \rightarrow 90-100 \rightarrow A
  \rightarrow \leq 70 \rightarrow F.
  \rightarrow 80-90 \rightarrow B
  \rightarrow 70-80 \rightarrow C
  \rightarrow 60-70 \rightarrow D
Important Notes

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

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

3. A switch can occur within another but in practice this is rarely done.
Chapter 3 - 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 find out whether a student is pass or fail; if it requires total 40% and at least 33% in each subject to pass. Assume 3 subjects and take marks as an input from the user.

3. Calculate income tax paid by an employee to the government as per the slabs mentioned below:
   
<table>
<thead>
<tr>
<th>Income Slab</th>
<th>Tax</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.51 - 50 L</td>
<td>5%</td>
</tr>
<tr>
<td>5.01 - 10.0 L</td>
<td>20%</td>
</tr>
<tr>
<td>Above 10.0 L</td>
<td>30%</td>
</tr>
</tbody>
</table>
   
   Note that there is no tax below 2.51. 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 the greatest of four numbers entered by the user.
Chapter 4 - Loop Control Instruction

Why Loops
Sometimes we want our programs to execute few set of instructions over and over again, for ex: 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:

1. While loop
2. do-while loop
3. 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.
An example:

```c
int i = 0

while (i <= 10)
    printf("The value of i is %d", i); i++;
```

Note: If the condition never becomes false, the while loop keeps getting executed. Such a 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.

Increment and decrement operators

```c
i++  \rightarrow i \text{ is increased by 1}
```

```c
i--  \rightarrow i \text{ is decreased by 1}
```

```c
printf("--i = %d", --i);
```

This first decrements i and then prints it

```c
printf(" i-- = %d", i--);
```

This first prints i and then decrements it
* `++` operator does not exist ⇒ Important

* `+=` is compound assignment operator just like `-=`, `*=`, `/=`, `%=` ⇒ Also Important

**do-while Loop**

The syntax of `do-while` loop looks like this:

```
do 
  // Code
  // Code
  \# while (condition)
```

`do-while` loop works very similar to `while` loop:

while ⇒ checks the condition & then executes the code

`do-while` ⇒ executes the code & then checks the condition

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

→ Quick Quiz: Write a program to print first n natural numbers using `do-while` loop

Input : 4

Output : 1

2

3

4
for Loop
The syntax of for loop looks like this:

```c
for (initialize; test; increment) {
	// Code;
}
```

Initialize → Setting a loop counter to an initial value
Test → Checking a condition
Increment → Updating the loop counter

An example:

```c
for (i = 0; i < 3; i++) {
    printf("%d", 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 > 0; 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 non 0) is tested
3. The code is executed
4. i is decremented
5. Condition i is checked & code is executed if its not 0.
6. & So on until i is non 0

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

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
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.

Let us look at an example

```c
int skip = 5;
int i = 0;

while (i < 10) 
if (i == skip)
    continue;
else
    printf ("%d", i);
```

output ⇒ 5

and not 0 ... 9
Notes:

1. Sometimes, the name of the variable might not indicate the behaviour of the program.
2. `break` statement completely exits the loop.
3. `continue` statement skips the particular iteration of the loop.
Chapter 4 - 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:
   1. at least once
   2. at least twice
   3. 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 8x1 to 8x10).

8. Write a program to calculate the factorial of a given number using a for loop.
9. Repeat 8 using while loops.

10. Write a program to check whether a given number is prime or not using loops.

11. Implement 10 using other types of loops.
Project 1: Number guessing Game

Problem: This is going to be fun! We will write a program that generates a random number and asks the player to guess it. If the player’s guess is higher than the actual number, the program displays "Lower number please". Similarly, if the user’s guess is too low, the program prints "Higher number please". When the user guesses the correct number, the program displays the number of guesses the player used to arrive at the number.

Hint: Use loops
Use a random number generator
Chapter 5 - Functions and Recursion

Sometimes our program gets bigger in size and its not possible for a programmer to track which piece of code is doing what 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 many number of times.

Example and Syntax of a Function

```c
#include <stdio.h>

void display(); => Function prototype

int main() {
    int a;
    display(); => Function call
    return 0;
}

void display() => Function definition
    printf("Hi I am display");
    return 0;
```
Function prototype
Function prototype is a way to tell the compiler about the function we are going to define in the program. Here void indicates that the function returns nothing.

Function call
Function call is a way to tell the compiler to execute the function body at the time the call is made. Note that the program execution starts from the main function in the sequence the instructions are written.

Function definition
This part contains the exact set of instructions which are executed during the function call. When a function is called from main(), the main function falls asleep and gets temporarily suspended. During this time, the control goes to the function being called. When the function body is done 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

→ Execution of a C program starts from main().
→ A C program can have more than one function.
→ Every function gets called directly or indirectly from main().
→ There are two types of functions in C: let's talk about them.

Types of Functions

1. Library functions → Commonly required functions grouped together in a library file on disk.
2. User-defined functions → These are the functions declared and defined by the user.

Why use functions?

1. To avoid rewriting the same logic again and again.
2. To keep track of what we are doing in a program.
3. To test and check logic independently.
Passing values to functions
We can pass values to a function and get a value in return from a function.

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

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:

```c
int sum (int a, int b)
{
    int c;
    c = a + b;
    return c;
}
```

Now we can call `sum (2, 3);` from main to get 5 in return.

```c
int d = sum (2, 3); => d becomes 5
```

Note:

1. Parameters are the values or variable placeholders in the function definition. Ex. `a` & `b`.

2. Arguments are the actual values passed to the function to make a call. Ex. `2 & 3`.

⇒ Here 2 & 3 are arguments.
A function can return only one value at a time.

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

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

`change` is a function which changes `a` to 77. No if we call it from main like this

```c
int b = 22
change (b);
printf("b is %d", b);
```

=> The value of b remains 22

=> prints "b is 22"

This happens because a copy of `b` is passed to the `change` function.

Quick Quiz ⇒ Use the library functions to calculate the area of a square with side `a`.
Recursion
A function defined in C can call itself. This is called recursion.
A function calling itself is also called 'recursive' function.

Example of Recursion
A very good example of recursion is factorial:

\[
\text{factorial}(n) = 1 \times 2 \times 3 \ldots \times n
\]

\[
\text{factorial}(n) = 1 \times 2 \times 3 \ldots (n-1) \times n
\]

\[
\text{factorial}(n) = \text{factorial}(n-1) \times n
\]

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

```c
int factorial (int x) {
    int f;
    if (x == 0 || x == 1)
        return 1;  // A program to calculate factorial using recursion
    else
        f = x * factorial(x-1);
    return f;
}
```
How does it work?

factorial (5)

5 \times \text{factorial}(4)

5 \times 4 \times \text{factorial}(3)

5 \times 4 \times 3 \times \text{factorial}(2)

5 \times 4 \times 3 \times 2 \times \text{factorial}(1)

5 \times 4 \times 3 \times 2 \times 1

Important Notes:

1. Recursion is sometimes the most direct way to code a given algorithm.
2. The condition which doesn't call the function any further in a recursive function is called as the base condition.
3. Sometimes, due to a mistake made by the programmer, a recursive function can keep running without returning, resulting in a memory error.
Chapter 5 - Practice Set

1. Write a program using functions 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 \((g = 9.8 \text{ m/s}^2)\).

4. Write a program using recursion to calculate \(n^{th}\) element of Fibonacci series.

5. What will the following line produce in a C program:

\[
\text{printf("%d %d %d \n", a, ++a, a++);}\]

6. Write a recursive function to calculate the sum of first \(n\) natural numbers.

7. Write a program using functions to print the following pattern (first \(n\) lines):

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

....

Scanned with CamScanner
Chapter 6 - Pointers

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

```
72  87994
```

Address: 87994  Address: 87998

j is a pointer
j points to i

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
&j  =>  87998
```

Format specifier for printing pointer address is ‘%lu’

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
*(&j) = 87994
```
How to declare a Pointer?

A pointer is declared using the following syntax:

```c
int * j;    => declare a variable j of type int pointer
j = &i      => store address of i in j
```

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

```c
int * ch_ptr;  => Pointer to integer
char * ch_ptr; => Pointer to character
float * ch_ptr; => Pointer to float
```

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

A Program to demonstrate pointers

```c
#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("Value i = %d\n", i);
    printf("Value i = %d\n", *(j));
    printf("Value i = %d\n", *(j));
    return 0;
}
```
Output:
Add i = 87994
Add i = 87994
Add j = 87998
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?

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

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>87994</td>
<td>87998</td>
</tr>
<tr>
<td>87994</td>
<td>87998</td>
<td>88004</td>
</tr>
</tbody>
</table>

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

Call by value → Sending the values of arguments
Call by reference → Sending the address of arguments

Call by value
Here the value of the arguments are passed to the function. Consider this example:

```c
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. Example:
Void Swap(int *x, int *y)

int temp;
 temp = *x;
 *x = *y;
 *y = temp;

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

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
Chapter 6 - Practice Set

1. Write a program to print the address of a variable. Use this address to get the value of this 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 same? Why?

3. Write a program to change the value of a variable to ten times of its current value. Write a function and pass the value by reference.

4. 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()

5. Write a program to print the value of a variable i by using "pointer to pointer" type of variable.

6. Try problem 3 using call by value and verify that it doesn't change the value of the said variable.
Chapter 7 - Arrays

An array is a collection of similar elements.

One variable ⇒ capable of storing multiple values

Syntax

The syntax of declaring an array looks like this:

```c
int marks[90];  ⇒ Integer array
char name[20];  ⇒ Character array or String
float percentile[90];  ⇒ Float array
```

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

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

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

```
Marks → 7 6 2 3 91 3 88 89
0 1 2 3 4 5 ... 88 89
```

Total = 90 elements
Accessing elements
Elements of an array can be accessed using:

`scanf("%d", &marks[0]);` \(\Rightarrow\) Input first value

`printf("%d", marks[0]);` \(\Rightarrow\) Output first value of the array

Quick Quiz \(\rightarrow\) Write a program to accept marks of five students in an array and print them to the screen.

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

```c
int cepa[3] = {9, 8, 8}; \(\Rightarrow\) Arrays can be initialized while declaration.
float marks[] = {33, 40};
```

Arrays in memory
Consider this array:

```c
int arr[3] = {1, 2, 3}; \(\Rightarrow\) 1 integer = 4 bytes
```

This will reserve \(4 \times 3 = 12\) bytes in memory.

4 bytes for each integer.

```
1 2 3
62302 62306 62310 \(\Rightarrow\) arr in memory
```
Pointer Arithmetic
A pointer can be incremented to point to the next memory location of that type.

Consider this example

```c
int i = 32;
int *a = &i; => a = 87994    address => 87994
a++;  => Now a = 87998
```

```c
char a = 'A';
char *b = &a; => b = 87994
b++;  => Now b = 87995
```

```c
float i = 1.7;
float *a = &i; => Address of i or a = 87994
a++;  => Now a = 87998
```

Following operations can be performed on a pointers:

1. Addition of a number to a pointer
2. Subtraction of a number from a pointer
3. Subtraction of one pointer from another
4. 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.
Accessing Arrays using pointers

Consider this array

\[
\begin{array}{cccc}
7 & 9 & 2 & 8 & 1 \\
\end{array}
\]

index \ 0 \ 1 \ 2 \ 3 \ 4

\[\text{ptr}\]

If \(\text{ptr}\) points to index \(0\), \(\text{ptr}++\) will point to index \(1\) & so on...

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

\[
\text{int}^{*} \text{ptr} = \& \text{arr}[0]; \quad \text{or simply arr} \\
\text{ptr}++; \\
* \text{ptr} \Rightarrow \text{will have 9 as its value}
\]

Passing arrays to functions

Arrays can be passed to the functions like this

\[
\text{printArray(arr, n);} \quad \Rightarrow \text{function call}
\]

\[
\text{void printArray(int* i, int n);} \quad \Rightarrow \text{function prototype}
\]

or

\[
\text{void printArray(int i[7], int n);} 
\]
Multidimensional Arrays

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

A 2 dimensional array can be defined as:

```
int arr[3][2] = 
2 3
2 4
2 5
```

We can access the elements of this array as:

```
arr[0][0], arr[0][1] & so on...
```

```
Value=1
Value=4
```

2-D arrays in Memory

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

```
arr[0][0] arr[0][1] ... 
1 4 7 9 11 22
8 2 2 8 2 8
```

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.
Chapter 7 - 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 a function which counts the number of positive integers in an array.

7 = Create an array of size 3x10 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 address of its elements in increasing order.
Chapter 8 - Strings

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

null character is used to denote string termination characters are stored in contiguous memory location

Initializing Strings

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

```
char s[7] = {H, A, R, R, Y, \0} ;
```

There is another shortcut for initializing strings in C language:

```
char s[7] = "HARRY" ; => In this case C adds a null character automatically.
```

Strings In Memory

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

```
H A R R Y \0
82210 82211 82212 82213 82214 82215
```

Contiguous blocks in memory
Quick Quiz → Create a string using " " and print its content using a loop.

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[7] = "HARRY";
printf("%s", st);  // prints 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 the null character when the enter key is pressed.

Note:
1. The string should be short enough to fit into the array.
2. 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.

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.

puts(st); => prints the string places the cursor on the next line

Declaring a string using pointers
We can declare strings using pointers

Char * ptr = "Harry";

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

Note:
1. Once a string is defined using char st[1] = "Harry", it cannot be reinitialized to something else.
2. 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.

Some of the most commonly used string functions are:

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

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

These functions are declared under `<string.h>` header file.

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

```
char source[10] = "Harry";
char target[30];
strcpy(target, source); \Rightarrow target now contains "Harry"
```

Target string should have enough capacity to store the source string.
`Strcat()`
This function is used to concatenate two strings.

```c
char s1[10] = "Hello";
char s2[10] = "Harry";

Strcat(s1, s2);  \Rightarrow s1 now contains "Hello Harry";
```
Chapter 8 - Practice Set

1. Which of the following is used to appropriately read a multi-word string
   (a) gets()
   (b) puts()
   (c) printf()
   (d) scanf()

2. Write a program to take string as an input from the user using %c and %s. 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 strupr 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.
Chapter 9 - Structures

Arrays and strings => Similar data (int, float, char)

Structures can hold => dissimilar data

Syntax for creating Structures

A C structure can be created as follows:

```c
struct employee E
    int code;
    float salary;
    char name[10];
3;
```

=> Semicolon is important

We can use this user defined data type as follows:

```c
struct employee e1;
    This declares a new user defined data type!

    strcpy (e1.name, "Harry");
    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.

```c
struct employee facebook[100];  // An array of structures
```

We can access the data using:
```c
google [0]. code = 100;
google [1]. code = 101;
... & 50 on
```

Initializing Structures
Structures can also be initialized as follows:
```c
struct employee harry = { 100, 7122, "Harry" };
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:

```
100    71 22    "Harry"
```

Address → 78810 78814 78818

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

Pointer to structures

A pointer to structure can be created as follows:

```c
struct employee *ptr;
ptr = &e1;
```

Now we can print structure elements using:

```c
printf("%d", *(ptr).*code);
```

Arrow operator

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

```c
*(ptr).*code  OR  ptr->code
```

Here `->` is known as the `arrow operator`.
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

Quick Quiz: Complete this show function to display the content of employee.

**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.

```c
struct complex {
    float real;
    float imag;
};
```

```c
typedef struct complex C;
```

```c
C C1, C2;
```

for defining complex numbers

```c
typedef struct complex C;
```

```c
C C1, C2;
```

for defining complex numbers
Chapter 9 - 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 \( \Rightarrow \) 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 keyword.

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 data. Write a function to compare those dates.

10. Solve problem 9 for time using typedef keyword.
Chapter 10 - 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.

```
C Program  ↓ write ↓ FILE
\      ^ read ^
```

Programmer

FILE pointer
The "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:

```c
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 → If the file does not exist, fopen returns NULL

"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

Types of Files

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:

```c
FILE *ptr;
ptr = fopen("Harry.txt", "r");
```
Let us assume that "Harry.txt" contains an integer. We can read that integer using:

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

This will read an integer from file in `num` variable.

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:

```c
fclose (ptr);
```

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

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

```c
FILE *fptr;
fptr = fopen ("Harry.txt", "w");
```
```c
int num = 432;
fprintf(fp, "%d", num);
fclose(fp);
fgets() and fprintf()

fgets and fprintf are used to read and write a character from/to a file.

fgets(ptr) => used to read a character from file

putc('c', ptr); => used to write character 'c' to the file

EOF : End of File

fgets 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 = fgets(ptr);
    if (ch == EOF) {
        break;
    }
}
```

// code

```
Chapter 10 - 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 a 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:

   name1, 3300
   name2, 7700

5. Write a program to modify a file containing an integer to double its value.

   \[
   \begin{array}{c}
   \text{prev. file} \\
   \text{new file}
   \end{array}
   \]
Project 2: Snake, Water, Gun

Snake, water, gun or Rock paper scissors is a game most of us have played during school time. (I sometimes play it even now 😊)

Write a C program capable of playing this game with you.

Your program should be able to print the result after you choose Snake/Water or Gun.
Chapter 11 - 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

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

Functions for DMA in C

Following functions 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));
```

- Casting void pointer to int
- Space for 30 ints
- Returns size of 1 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().

The **malloc** function stands for continuous allocation. It initializes each memory block with a default value of 0.

**Syntax:**

```c
ptr = (float*) malloc(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 malloc, where n is an integer entered by the user.

**free() function**

We can use `free()` function to de-allocate 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);
potr = realloc(ptr, 3 * sizeof(int));
```

- `ptr` now points to this new block of memory capable of storing 3 integers.
Chapter 11 - 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 (7x10 = 70). Use `realloc` to make it store 15 numbers (from 7x1 to 7x15).