HAPPY LEARNING.....

LET US C (17th EDITION)

SUMMARY

OF

ALL

CHAPTERS

The only guide you need for C.

CHAPTER 1: GETTING STARTED

Notes:

- Constants = Literals -> Cannot change
  Variables = Identifiers -> May change
- Types of variables and constants : 1) Primary 2) Secondary
- 3 types in Primary : 1) Integer 2) Real (float) 3) Character
- Ranges :
  1) 2-byte integer : -32768 to +32767
  2) 4-byte integer : -2147483648 to +247483648
  3) floats : -3.4 x 10^38 to +3.4 x 10^38
- In a char constant , both quotes must slant to the left, like 'A'
- Variable has two meanings :
  1) It's an entity whose value can change
  2) It's a name given to location memory
- Variable names are case-sensitive and must begin with an alphabet or underscore
- printf() is a function that can print multiple constants and variables
- Some format specifiers in printf(), scanf() : int - %i, float - %f, char - %c
- Use /*.....*/ or // for a comment in a C program
- & is 'address of' operator and must be used before a variable in scanf

CHAPTER 2: C INSTRUCTIONS

Notes:

- Every compiler is targeted towards a particular OS + Microprocessor combination. This combination is known as a platform. A compiler created for one platform does not work with other platform.

- / gives quotient, % gives remainder. While taking %, sign of remainder is same as sign of numerator. % doesn't work with floats.

- C offers three types of instruction :
  1) Type declaration 2) Arithmetic 3) Control

- Declaration and assignment can be combined. Ex. : int a=5

- 3 types of arithmetic instruction :
  1) Integer mode 2) Real mode 3) Mixed mode

- Rules for arithmetic instruction :
  - If one is float , result is a float
  - Result is int only if both operands are ints

- a = pow(2, 5) ; would store 2^5 in a. Remember to #include<math.h>

- Every operator has priority and Associativity, priority is * / %, + -, = . Priority can be changed using ().

- Associativity is either L to R or R to L. + , - , * , / , % has L to R, = has R to L associativity.

- Format string of printf() can contain :
  1) Format specifier - %c, %d, %f, etc.
  2) Escape sequences - \n, \t, etc. and any other character

- Format string of scanf() can contain only format specifiers

- 4 types of control instructions :
  1) Sequence 2) Decision 3) Repetition 4) Case

CHAPTER 3: DECISION CONTROL INSTRUCTION

Notes:

- Three ways for taking decisions in a program :
  1) Using if-else statement
  2) Using conditional operators
  3) Using the switch statement

- The default scope of if and else statement is only the next statement. So to execute multiple statements they must be written in a pair of braces.

- Condition is built using relation operators <, >, <=, >=, ==, !=

- An if need not always be associated with an else. However, an else must always be associated with an if.

- An if-else statement can be nested inside another if-else statement.

- a=b is assignment, a==b is comparison.

- In if(a==b==c), result of a==b is compared with c.

- If a condition is true it is replaced by 1, if it is false it is replaced by 0.

- Any non-zero number is true, 0 is false.

- ; is a null statement. It doesn't do anything on execution.

CHAPTER 4: MORE COMPLEX DECISION MAKING

Notes:

- Logical operators are &&, || and !. Useful in checking ranges and solving yes/no problem

- One more form of decision control instruction is :

```
if(condition 1)
    statement 1;
else if(condition 2)
    statement 2;
else if(condition 3)
    statement 3;
else
    statement 4;                    else works if all 3 ifs fail
```

- Unary operator needs only 1 operand. Ex. ! , sizeof()

- Binary operator needs 2 operands. Ex. +, -, *, /, %, <, >, <=, >=, ==, !=, &&, ||

- sizeof() is an operator. It gives number of bytes occupied by an entity. Ex. a = sizeof(int)

- !(a<=b) is same as (a>b). !(a>=b) is same as (a<b)

- a = !b does not change value of b.
  a = !a means, set a to 0 if a is 1 and set it to 1 if it is is 0.

- General form of ternary operators :
  condition ? statement1 : statement2

- ? : can have only 1 statement each

- ? : can be nested

- ? : always go together. : is not optional

- Always parenthesize assignment operation if used with ? :

CHAPTER 5: LOOP CONTROL INSTRUCTION

Notes:

- Repetition control instruction is implemented using :
  1) while loop
  2) for loop
  3) do-while loop

- General form of while :

```
i = 1;   /* initialization of loop counter */
while (i <= 10)   /* testing of loop counter */
{
     statement 1;
     statement 2;
     i++;  /* incrementation of loop counter */
}
```

- i++ increments value of i by 1
  i-- decrements value of i by 1
  there are no **, // and %% operators

- The expressions i = i+1, i++, ++i are all same

- j = ++i; first increments i, then assigns the incremented value to j

- j = i++; first assigns current value of i to j, then increments i

- Compound assignment operators : +=, -=, *=, /= and %=
  Ex. i += 5; which is same as i=i+5;

- Running sum and products are implemented using following :

```
S = 0;
P = 1;
while(condition)
{
     /*  calculate term  */
     S = S + term;
     P = P * term;
}
```

CHAPTER 6: MORE COMPLEX REPETITIONS

Notes:

- Usual usage :
  while loop - to repeat something unknown no. of times
  for loop - to repeat something fixed no. of times
  do-while loop - to repeat something at least once

- Equivalent forms :

```
i = 1;                  for(i=0;i<=10;i++)      i=1;
while (i <= 10)         {                       do
{                           statement 1;        {
    statement 1;            statement 2;            statement 1;
    statement 2;        }                           statement 2;
    i++;                                            i++;
}                                                }while(i<=10);
```

- for(; ;) is an infinite loop. While() results into an error

- Multiple initializations, conditions and incrementations in a for loop are acceptable. Ex.

```
for(i=1, j=2; i<=10 && j<=24; i++, j+=3)
{
    statement 1;
    statement 2;
}
```

- break - terminates the execution of the loop
  continue - goes for next iteration of loop abandoning rest of instructions

- Usual usage of break and continue :

```
while (condition 1)             while(condition 1)
{                               {
    if(condition 2)                 if(condition 2)
        break;                          continue;
    statement 1;                    statement 1;
    statement 2;                    statement 2;
}                                }
```

CHAPTER 7: CASE CONTROL INSTRUCTION

Notes:

- One more form of decision making can be done using switch - case - default. This is used when we are to check whether a variable or an expression has one of the several possible values

- General form :

```
switch (expression){      ->  use constant or variable expression

    case constant expression:   -> use only constant exp.
        statement
    case constant expression:
        statement
    default:
        statement

}
```

- If a case fails control jumps to the next case. Break takes the control out of the switch. Continue doesn't rake the control to the beginning of the switch

- Order in which cases are written does not matter. Default case is optional.

- Cases in a switch must always be unique
  Switch can be used with int, long int, char
  switch cannot be used with float, double
  switch works faster than a series of ifs

- goto keyword can take the control from any place to any other place within the function. It should be used only when we wish to break out of the innermost loop in a nested loop system

- exit() - function - terminates program execution

- #include<stdlib.h> for exit() to work

CHAPTER 8: FUNCTIONS

Notes:

- Functions are a group of instructions achieving some goal

- Why create functions :
  1) Better complexity management - Easy to design and debug
  2) Provide reuse mechanism - Avoids rewriting same code

- Types of functions :
  1) Library - printf(), scanf(), pow()
  2) User-defined - main()
  Rules for building both are same

- Three things should be done while creating a function :
  1) Function definition
  2) Function call
  3) Function prototype declaration

- General form :
  return-type function-name(type arg1, type arg2, type arg3){
  statement1; statement2;
  return (variable/constant/expression); -> only 1 value
  }

- C program is a collection of one or more functions
- If it contains more than 1 functions, then one must be main()
- Execution of any C program begins with main()
- Function names in a program must be unique
- Any function can call any other function
- Functions can be defined in any order

- More the function calls slower the execution

- If values are passed to a function, then function must collect it while defining it

- Arguments passed to a function are called actual arguments
- Arguments received by function are called formal arguments
- Actual & Formal arguments must match in number,order,type
- Actual arguments can be constants/ variables/ expressions
- Formal arguments must be variables

MORE.....

Notes:

- Nested calls are legal. Ex. : a=sin(cos(b));
- Call within an expression is legal. Ex. : a=sin(b) + cos(c);
- The error "Unresolved internal" usually means there is a mistake in the function name spelling
- return (s); - Returns control & value
  return ; - Returns only control
- If value is returned from function, we can choose to ignore it
- To ensure no value is returned from a function, use void as the return-type in function definition and its prototype declaration
- A function by default returns an integer value. If we do not specifically return an integer value then a garbage integer value is returned
- A function can return a non-integer value. The type of value must be suitably mentioned in the function definition and its prototype declaration as in :

  ```
  float area (float r) ;     /* function prototype declaration */

  float area (float r){     /* function definition */

  }
  ```

CHAPTER 9: POINTERS

Notes:

- 3 ways to call a function :
  1) Call by value - values are passed to the called function
  2) Call by reference - addresses are passed to called function
  3)Mixed call - values and addresses are passed

- Pointers are variables which hold addresses of other variables

- Address, Reference, Memory location, Cell number are same

- & - address of operator, * - value at address or indirection operator

- & can be used only with variables, * can be used with variable, constant or expression

- variable is same as *&variable

- Example of pointer usage :
  int i=10; int *j; int **k;
  j = &i; k = &j;
  printf("%d %d %d", i, *j, **k);

  here j is an integer pointer. k is a pointer to an integer pointer

- &a always gives base address of a, no matter variable is of how many bytes

- Using an integer ptr - use * to reach integer
  Using a pointer to an integer pointer - use ** to reach integer

- Call by value - change doesn't affect actual arguments
  Call by reference - actual arguments can be changed

  Examples of call types :
  1) swapv(a, b); - call by value
  2) swapr(&a, &b); - call by reference
  3) sumprod(a, b, c, &s, &p); - mixed call

CHAPTER 10: RECURSION

Notes:

- A function that calls itself is called a recursive function
- Any function including main() can become recursive
- Recursive call always leads to an infinite loop. So a provision must be made to get outside this infinite loop
- The provision is done by making the recursive call either in the if block or in the else block
- If recursive call is made in the if block, else block should contain the end condition logic, vice-versa
- Fresh set of variables are born during each function call - normal call and recursive call
- Variables die when control returns from the function
- Recursive function is an alternative for loop in which logic are expressible in the form of themselves
- Recursive calls are slower than an equivalent while/ for/ do-while loop
- Understanding how a recursive function is working becomes easy if you make several copies of the same function on paper and then perform a dry run of the program

CHAPTER 11: DATA TYPES REVISITED

Notes:

- Types :
  Integer - short, long, signed, unsigned, int
  Char - signed, unsigned
  Real - float, double, long double

- Sizes of data types may vary from one compiler to another. For example, int is two bytes in Turbo C, 4 bytes in Visual Studio Code

- For all compilers : sizeof(short) <= sizeof(int) <= sizeof(long)

- In signed, left-most bit is +ve/-ve. In unsigned, all bits contribute to value

- Negative integer are stored as 2's complement (negation and adding 1)

- Number without a decimal point is by default an in. Use suitable suffix to change it :
  365 - int, 365u - unsigned int, 365L/365l - long int,
  365 lu/365ul - long unsigned

- Number with a decimal point is by default a double. Use suitable suffix to change it :
  3.14 - double, 3.14f - float, 3.14L: - long double

- Two things are needed to completely define a variable :
  1) Type of variable 2) Storage class of variable

- Type signifies what type of value can be stored in the variable

- Storage class signifies 4 things :
  1) Storage - where the variable is stored
  2) Default value - what value would it holds if not initialized
  3) Scope - where the variable would be available
  4) Life - how long would the variable be available

MORE.....

Notes:

- Automatic storage class :
  1) Storage - memory 2) Default value - garbage
  3) Scope - local to the block ({})
  4) Life - till control is in the block in which variable is defined

- Register storage class :
  1) Storage - CPU register 2) Default value - garbage
  3) Scope - local to the block
  4) Life - till control is in the block in which variable is defined

- Static storage class :
  1) Storage - memory 2) Default value - 0
  3) Scope - local to the block
  4) Life - till execution of program doesn't end

- Extern storage class :
  1) Storage - memory 2) Default value - 0
  3) Scope - global
  4) Life - till execution of program doesn't end

- Definition of variable reserves space, just declaration doesn't. Redeclaration of variable is ok, redefinition not

- int i ; ->definition , extern int i ; -> declaration

- Local variable gets a priority over global variable of same name

- Out of locals of same name, most local variable gets a priority

- Usage :
  Register - for frequently used variable
  Static - if variable is to live across functions
  External - if variable is required by all functions
  Automatic - all other cases

CHAPTER 12: THE C PREPROCESSOR

Notes:

- Preprocessor expands the source code as per the preprocessor directives used in it

- 4 types of preprocessor directives :
  1) Macro expansion 2) File inclusion
  3) Conditional compilation 4) Miscellaneous directives

- #inlude"stdio.h" - searches the file in file in include path+current path

- #include<stdio.h> - searches the file in just included path

- Macros - every template is replaced by its expansion. Macro have global effect

- #define PLANK 6.634E - 34 -simple macro
  #define AREA(x) PI*x*x -macro with argument

- Macros can take multiple arguments :-
  #define CALC(a,b,c,d) (a+b*c/3.14)

- Macro can be split over multiple lines. Put a \ at the end of each line, except last line

- Macros are faster and functions occupy less space

- Be aware of side-effects of macros with arguments
  #define SQUARE(y) y*y
  would expand z=SQUARE(3+1) into z=3+1*3+1

- Conditional compilation - compiles the code only if the condition is true. Implemented using #ifdef, #else, #endif, #ifndef, #if

- Miscellaneous directives :
  #undef - undefines a macro that has already been defined
  #pragma inline - used for compilation of program that uses assembly language statements

CHAPTER 13: ARRAYS

Notes:

- Array is a variable capable of holding >1 value at a time
- Two basic properties of an array :
  1) Similarity - All array elements are similar to one another
  2) Adjacency - All array elements are stored in adjacent memory
- 2 ways to declare an array :
  ```
  int arr[10] ;          /* mentioning size is compulsory */
  int num[] = {23,34,54,22,33} ;   /* size is optional */
  ```
- Array elements are always counted from 0 onwards. So arr[9] is 10th element
- Array have storage classes. Default auto
- Array elements can be scanned OR calculated :
  ```
  scanf("%d %d %d", &arr[7], &arr[8], &arr[9]);
  arr[5] = 3 + 7 % 2 ;
  ```
- Arithmetic on array elements is allowed :
  ```
  arr[6] = arr[1] + arr[3]/16 ;
  ```
- Caution : Bounds checking of an array is programmer's responsibility. Ex. even if arr[5] is declared i.e. max size is 5 but it will give no error while receiving values after this bound and values at another locations may get overwritten
- Typical way to process an array element by element :
  ```
  int arr[10] ;
  for(i=0; i<=9; i++){
      /* process */
  }
  ```
- To obtain address of 0th element of array use :
  ```
  int arr[10] ;  int *p ;
  p = arr;          /* method 1 */
  p = &arr[0];   /*method 2 */
  ```

MORE.....

Notes:

- Sorting = arranging array elements in ascending/descending order

- Bubble sort = compare adjacent elements repeatedly

- Selection sort = compare 0th element with all others, 1st with other, etc.

- On incrementing a pointer it always points to the next location of its type
  On incrementing a float pointer it points to the next float which is 4 bytes away
  Similarly for int(4) and char(1)

- Only legal pointer operations :
  pointer + number -> pointer
  pointer - number -> pointer
  pointer - pointer -> pointer
  pointer == pointer

- 5 ways to access array elements using pointer :
  - Set up a pointer holding base address of the array :
    int arr[10], *p ;
    p = arr ;
  - In a for loop use of the five expressions
    1) *p ; p++ ; OR
    2) *(p+i) OR
    3) *(i+p) OR
    4) p[i] OR
    5) i[p]

- To pass array to function we must always pass two things :
  1) Base address of the array 2) Size of the array

- Array can neither grow nor shrink in size during execution of the program.

- We cannot declare array using int arr[n] and then receiving the value of n during execution

MORE.....

Notes:

- We can make the array size flexible by changing the value of MAX suitable :
  #define MAX 20
  int arr[MAX];

- To create a variable sized array, use the following :
  int *p;
  p = (int *) malloc(n * size of(int)) ;

  Then to access all elements we can use p[i]

CHAPTER 14: MULTIDIMENSIONAL ARRAY

Notes:

- 2-D array is a collection of several 1-D arrays
- If 2-D arrays is initialized at the same place where it is declared, then mentioning the column dimension is optional
- A 2-D array is laid out linearly in memory in row-major fashion i.e. row after row
- Given a 2-D array int a[4][5] ;
  a[2][3] == *a[2] + 3 == *(*(a+2)+3)
- int *p[4] ; - p is an array of 4 integers. Size of p is 16 bytes
- int (*p)[4] ; - p is a pointer to an array of 4 integers. Size of p is 4 bytes
- Application of 2-D array in games :
  chess, ludo, snakes and ladder, brainvita, any other board game
- 3-D array is a collection of several 2-D array. Size of 3-D array is sum of sizes of all its elements
- Following expressions are referring to the element in the 1st row, 3rd column of the 2nd 2-D array :
  a[2][1][3]
  * (a[2][1] + 3)
  * (*(a[2] + 1) + 3)
  * (*(*(a + 2)+ 1) + 3)
- For a 3-D array :
  a, *a, **a, will give address
  ***a will give the integer at a a[0[0][0]

CHAPTER 15: STRINGS

Notes:

- Strings are character arrays ending with '\0'. '\0' is called string terminator

- Other arrays do not end with '\0'. ASCII value of '0' = 48, and ASCII value of '\0' = 0

- Ways to output strings :
  ```
  char name[] = "Sanjay";
  printf("%s\n", name);
  puts(name);
  ```

- Ways to input strings :
  ```
  char name[30];
  scanf("%s", name);
  gets(name);
  ```

- To receive multiword strings :
  ```
  1) scanf("%[^\n]s", name);   /* ^means from beginning, \n
                                  means up to end */
  2) gets(name)
  ```

- Prefer scanf() for receiving name of city, gets() for receiving name and surname

- 3 = integer, 3.0 = double, '3' = character, "3" = string ending with '\0'

- Standard way of processing a string :
  ```
  char str[] = "Blah blah blah"; char *p;
  p = str;
  while(*p != '\0'){
      /* process current character given by *p */
      p++;
  }
  ```

- #include<string.h> for prototypes of library string functions mentioned in next page

MORE.....

Notes:

- Useful string functions :

```
int l = strlen(str);  /* returns length of string */

strcpy(target, source); /* copies source string to target */

strcat(target, source); /* appends source at the end of target*/

                                /* return 0 if strings are not equal,
int l = strcmp(str1, str2);     difference of ascii values if they
                                 are unequal */

strupr(str); /* converts string str to uppercase */

strlwr(str); /* converts string str to lowercase */

toupper(ch); /* converts character ch to uppercase */

tolower(str); /* converts character ch to lowercase */
```

- char p[] = "Nagpur" ;
  p is a constant pointer to string
  p cannot be changed
  Nagpur can be changed

- char *p = "Nagpur";
  p is a pointer to a constant string
  p can be changed
  Nagpur cannot change

CHAPTER 16: HANDLING MULTIPLE STRINGS

Notes:

- 2 ways to handle multiple related strings :
  1) Using 2-D array of strings
  2) Using array of pointer to strings

- Pros and cons of using 2-D of strings :
  Pros :
  Easy to process using 2 for loops and expression str[i][j]
  cons :
  Leads to wastage of precious memory space
  Leads to tedious processing of array elements

- Pros and cons of using array of pointers to strings :
  Pros :
  Easy to process
  Saves space
  Cons:
  Cannot change strings. Their relative positions in the array can be changed

  Cannot receive strings from keyboard easily. Can be done by allocating space for each string using malloc() and then assigning the addresses returned by malloc() to the array elements

CHAPTER 17: STRUCTURES

Notes:

- Structure is a collection of dissimilar(usually) elements stored in adjacent locations
  Structure is also known as - User-defined data type/ Secondary data type/ Aggregate data type/ Derived data type

- Terminology :
  struct employee {char name; int age; float salary;};
  struct employee e1, e2, e[10];

  struct - keyword employee - structure name/tag
  name, age, salary - structure elements/ structure members
  e1, e2 - structure variables e[] - array of structures

- Structure elements are stored in adjacent memory locations

- Size of structure variable = sum of sizes of structure elements

- 2 ways to copy structure elements :
  struct emp e1 = {"Amit", 23, 4000.50};
  struct emp e2, e3;
  1) e2.name = e1.name; e2.age = e1.age; e2.salary = e1.salary;
  2) e3 = e1;

- Structure can be nested :
  struct address {char city[20]; long int pin;};
  struct emp {char n[20]; int age; struct address a; float s;};
  struct emp e;

  To access city and pin we should use e.a.city and e.a.pin

- To access structure elements using structure variables, use "." operator as in
  struct emp e;
  printf("%s %d %f", e.name, e.age, e.salary);

MORE.....

Notes:

- To access structure elements using structure pointer, use -> operator as in
  struct emp e;
  struct emp *p;
  p = &e;
  printf("%s %d %f", p->name, p->age, p->salary);

- Uses of structures :
  Database Management
  Displaying Characters
  Printing on printer
  Mouse Programming
  Graphics Programming
  All Disk Operations

CHAPTER 18: CONSOLE INPUT/OUTPUT

Notes:

- IO(input output) in C is always done using functions, not using keywords

- All IO functions can be divided into 2 broad categories :
  1) Console IO functions : a) Formatted b) Unformatted
  2) Disk IO functions

- The formatted console IO functions can force the user to receive the input in a fixed format and display the output in a fixed format

- All formatted console IO is done using printf() and scanf()

- Examples of formatting :
  %20s - right align a string in 20 columns
  %-10d - left align an integer in 10 columns
  %12.4f - right align a float in 12 columns with 4 places beyond decimal places

- Escape sequences :
  \n - positions cursor on next line
  \r - positions cursor at beginning of same line
  When we hit enter \r is generated and is converted into \r\n combination
  \t - positions cursor at beginning of next print zone. 1 print zone = 8 columns
  \', \", \\ - produces ', ", \ in the output

- scanf() can contain format specifier like %10.2f, but it is too restrictive, hence used rarely

- Unformatted console IO functions :
  char - getchar() - waits for enter
  int/float - no functions
  string - gets(), puts()

CHAPTER 19: FILE INPUT/OUTPUT

Notes:

- File I/O functions :
  a) High level :
  1) Text mode - (i) formatted (ii) unformatted
  2) Binary mode
  b) Low level

- High level text mode formatted file I/O functions :
  fprintf(), fscanf()
  High level text mode unformatted file I/O functions :
  char - fgetc(), fputc()
  int, float - no functions
  string - fgets(), fputs()

- I/O is always done using a buffer of suitable size. High level file I/O functions manage buffer themselves while using low level file I/O functions we have to manage the buffer

- Functions to open and close a file :
  High level - fopen(), fclose()
  Low level - open(), close()

- FILE *fp = fopen("temp.dat", "r");
  FILE is a structure declared in stdio.h
  fopen() - Creates buffer, Creates structure
  - Returns address of structure and assigns to fp

- ch = fgetc(fp); - Reads char, shifts pointer to next char
  Returns ASCII value of character read
  Returns EOF if no character is left

- To read a file character by character till we do not reach the end : while((ch=fgetc(fp)) != EOF)

- To read a file line by line till we reach the end :
  char str[80]
  while(fgets(str, 79, fp) != NULL)

- EOF and NULL are macros defined in stdio.h
  #define EOF -1
  #define NULL 0

MORE.....

- fopen() :
  To open file for reading in text mode - "rt" or "r"
  To open file for writing in text mode - "wt" or "w"
  To open file for reading in binary mode - "rb"
  To open file for writing in binary mode - "wb"

- Difference :
  fs = fopen(s, "r"); - Returns NULL if file is not in disk

  ft = fopen(t, "w"); - Returns address of FILE structure, if present. Creates new file if is absent Overwrites file, if present

  fclose(fs); - Vacates the buffer
  fclose(ft); - Writes buffer to disk, vacates buffer

- To read/write record to a file in text mode :
  struct emp e = {"Amit", 19, 4500.50};
  fprintf(fp, "%s %d %f", e.name, e.age, e.sal);
  while(fscanf(fp, "%s %d %f", e.name, e.age, e.sal) != EOF)

- To move the pointer in a file :
  fseek(fp, 512L, SEEK_SET);
  moves the pointer 512 bytes from the beginning of file.

  other macros :
  SEEK_END - from end of file
  SEEK_CUR - from current position of pointer

- To read/write buffer of 521 characters using low level file I/O functions :
  int in, out; char buffer[512];
  out=open("trial.dat", O_WRONLY | O_BINARY | O_CREAT);
  in = open("sample.dat", O_RDONLY | O_BINARY);
  write(out, buffer, 512);
  n = read(in, buffer, 512); /* n- no. of bytes read successfully */

- ch = fgetc(fp); - Reads char, shifts pointer to next char
  Returns ASCII value of character read
  Returns EOF if no character is left
  Include 3 files:
  #include<fcntl.h> #include<sys\types.h>
  #include<sys\stat.h>

CHAPTER 20: MORE ISSUES IN INPUT/OUTPUT

Notes:

- C>, $ are called command prompts in Windows and Linux respectively

- Command-line arguments are arguments provided to main() from command-line

- Command-line args are collected in main() in variables argc and argv
  argc - count of arguments
  argv - vector (array) of arguments
  Any variable names other than argc, argv are ok

- char *argv[] is an array of pointers to strings. So all arguments are received as strings and their addresses are stored in argv[]

- Errors in reading/ writing from/ to a file can be detected using ferror() and reported using perror() :

```
ch = fgetc(fp);
if(ferror())
    perror("ERROR while reading");
```

- Most OSs predefine pointers for three standard files :
  stdin - standard input device (keyboard)
  stdout - standard output device (monitor)
  stderr - standard error device (monitor)

- To use and give up these predefined pointers, we need not use fopen() and fclose()

- The statement ch = fgetc(stdin) would read a character from the keyboard

- If a program uses stdin then < at cmd input can be rediracted to be recieved from a file. Ex. C> calc < file

- If a program uses stdout and stderr then using > at cmd output and and error messages can be redirected to a file.
  Ex C> calc > file
  The operators < and > are called redirection operators

CHAPTER 21: OPERATIONS ON BITS

Notes:

- Bit = Binary Digit = Basic unit of information. A bit can take a value 0 or 1

- Units :
  4 bits = Nibble, 8 bits = Byte, 16 bits = word, 32 bits = Dword

- 4 Numbering systems :
  1) Binary - 0, 1
  2) Octal - 01.....7
  3) Decimal - 01.....9
  4) Hexadecimal - 0....9,A.....F

- PC/Laptop understand only Binary numbering system, C/C++ understand Octal, Decimal, Hexadecimal numbering systems

- Always try to convert Binary into Hexadecimal instead of Decimal, as while converting to Hex a nibble can be replaced by its equivalent Hex digit

- Bitwise Operations :
  Set a bit to a value 0/1 -> Write operation
  Check whether bit 6 is 1(on) or 0(off) -> Read operation

- Bitwise operators purpose :
  ~ - Converts 0s to 1s and 1s to 0s
  << >> - Shift out desired number of bits from left or right
  & - Check whether a bit is on/off. Put off a particular bit
  | - Put on a particular bit
  ^ - Toggle a bit

- <<=, >>=, &, |=, ^= - Bitwise compound assignment operators. a = a<<5; is same as a<<=5; . Except '~' all other bitwise operators are binary operators

- Remember :
  Anything ANDed with 0 is 0
  Anything ORed with 1 is 1
  1 XORed with 1 is 0

- printf("%#x", var); prints the hexadecimal output preceded by 0x

CHAPTER 22: MISCELLANEOUS FEATURES

Notes:

- We can write programs without using miscellaneous features like union, enum, etc. But that is not advisable

- Often we are required to handle an ordered listing of items. Example colors like red, green, blue or marital status like married, unmarried or divorced. Instead of handling these as integers, enums are a better way

- Usage of enums :
  enum color {red, green, blue}
  enum color windowcolor, buttoncolor;
  windowcolor = green; buttoncolor = blue;
  printf("%d %d", windowcolor, buttoncolor);

- A typedef declaration can be used to redefine the name of an existing data type as in
  typedef unsigned long int ULI;
  ULI var1, var2;

- Usually, uppercase letters are used to make it clear that we are dealing with a renamed datatype

- typecasting can be used to forcibly convert the value of an expression to a particular data type

- Multiple items of information can be stored in a byte using bit fields
  struct employee{
  unsigned gender : 1 ; unsigned mar_stat : 2 ;
  };
  The number after colon(:) indicates the number of bits to allot for the field

- void *p(); - Prototype of a function p() that receives nothing and return a void *

MORE.....

Notes:

- void (*p)(); - p is pointer to a function that receives nothing and returns nothing
- float * (*p)(int, float); - Pointer to a function that receives int & float and returns a float *
- Usage of function pointer :

```
void (*p)();
P = display;    /* stores address of display function in p */
(*p)();         /* first way to call display() */
P();            /* one more way to call display() */
```

- We can write a function that receives a variables number of arguments using macros va_list, va_start, va_arg
- Size of a structure is sum of sizes of its elements. Elements are accesses using
- Size of union variable is size of biggest element of the union. Elements are accessed using '.'
- Utility of union - Permits access to same memory locations in multiple ways
- Usage :

```
union a{
      int i; char ch[4];
};
union a z;
z.i = 512;
printf("%d %d %d %d %d", z.i, z.ch[0], z.ch[1], z.ch[2], z.ch[3]);
```

- If a number is ABCD then in little endian architecture it is stored as DCBA
- Little Endian - Low byte is stored first. Big Endian - High byte is stored first. Endianness is a matter of convenience. So both are good