Pointer

What is Pointer?

- A normal variable is used to store value.
- A pointer is a variable that store address / reference of another variable.
- Pointer is derived data type in C language.
- A pointer contains the memory address of that variable as their value. Pointers are also called address variables because they contain the addresses of other variables.

Declaration & Initialization of Pointer

Syntax

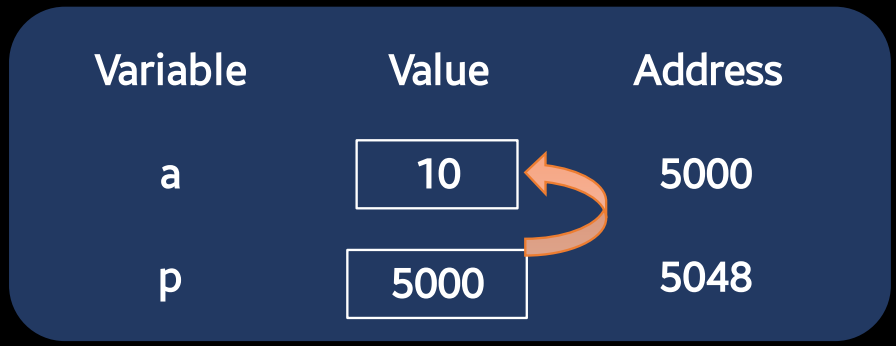
```
1  datatype *ptr_variablename;
```

Exampl
e

```
1  void main()
2  {
3          int a=10, *p; // assign memory address of a to pointer
4          variable p
5          p = &a;
6          printf("%d %d %d", a, *p, p);
7  }
```

Output

```
10 10 5000
```



- p is integer pointer variable
- & is address of or referencing operator which returns memory address of variable.
- * is indirection or dereferencing operator which returns value stored at that memory address.
- & operator is the inverse of * operator
- x = a is same as x = *(&a)

Why use Pointer?

- C uses pointers to create dynamic data structures, data structures built up from blocks of memory allocated from the heap at run-time. Example linked list, tree, etc.
- C uses pointers to handle variable parameters passed to functions.
- Pointers in C provide an alternative way to access information stored in arrays.
- Pointer use in system level programming where memory addresses are useful. For example shared memory used by multiple threads.
- Pointers are used for file handling.
- This is the reason why C is versatile.

Pointer to Pointer – Double Pointer

- Pointer holds the address of another variable of same type.
- When a pointer holds the address of another pointer then such type of pointer is known as pointer-to-pointer or double pointer.
- The first pointer contains the address of the second pointer, which points to the location that contains the actual value.

Syntax

```
datatype **ptr_variablename;
```

Example

```
int **ptr;
```

Write a program to print variable, address of pointer variable and pointer to pointer variable.

Program

```c
#include <stdio.h>
int main () {
        int var;
        int *ptr;
        int **pptr;
        var = 3000;
        ptr = &var; // address of var
        pptr = &ptr; // address of ptr using address of operator &
        printf("Value of var = %d\n", var );
        printf("Value available at *ptr = %d\n", *ptr );
    printf("Value available at **pptr = %d\n", **pptr);
    return 0;
}
```

Output

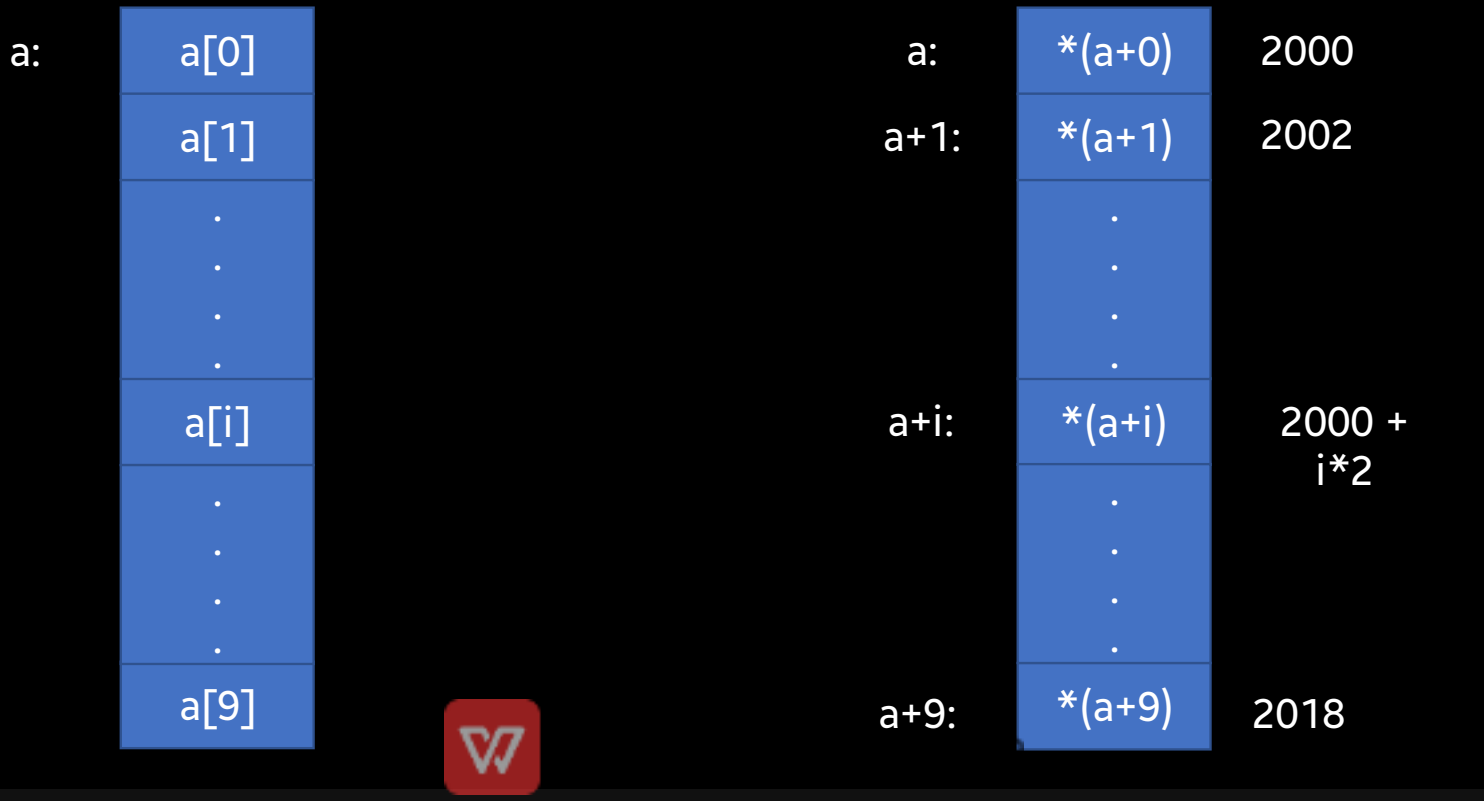
```
Value of var = 3000
Value available at *ptr = 3000
Value available at **pptr = 3000
```

Relation between Array & Pointer

- When we declare an array, compiler allocates continuous blocks of memory so that all the elements of an array can be stored in that memory.
- The address of first allocated byte or the address of first element is assigned to an array name.
- Thus array name works as pointer variable.
- The address of first element is also known as base address.

Relation between Array & Pointer – Cont.

- Example: int a[10], *p;
- a[0] is same as *(a+0), a[2] is same as *(a+2) and a[i] is same as *(a+i)



Array of Pointer

- As we have an array of char, int, float etc, same way we can have an array of pointer.
- Individual elements of an array will store the address values.
- So, an array is a collection of values of similar type. It can also be a collection of references of similar type known by single name.

Syntax

```
datatype *name[size];
```

Example

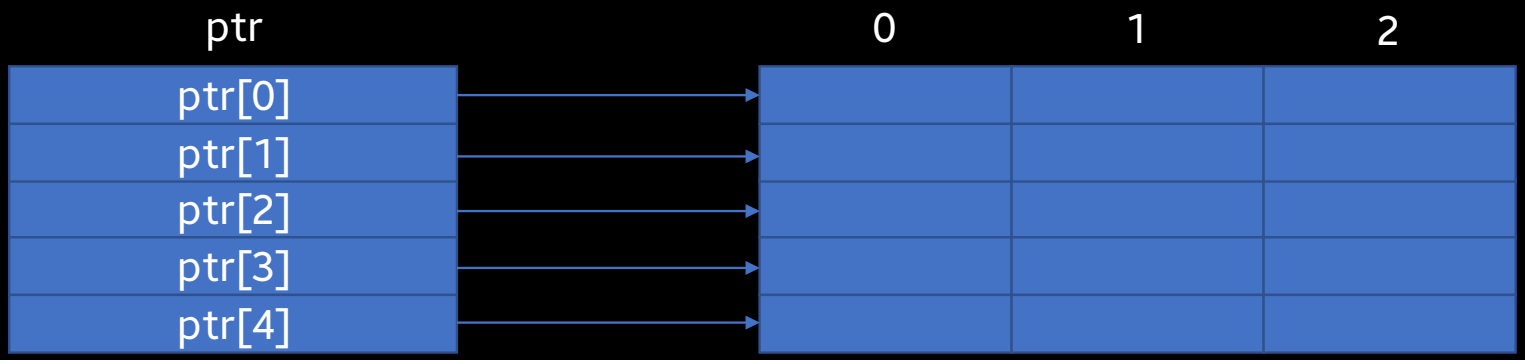
```
int *ptr[5]; //declares an array of integer pointer of size 5
```

Array of Pointer – Cont.

- An array of pointers ptr can be used to point to different rows of matrix as follow:

Example

```
for(i=0; i<5; i++)
{
    ptr[i]=&mat[i][0];
}
```



- By dynamic memory allocation, we do not require to declare two-dimensional array, it can be created dynamically using array of pointers.

Pointer and Function

- Like normal variable, pointer variable can be passed as function argument and function can return pointer as well.
- There are two approaches to passing argument to a function:
  - Call by value
  - Call by reference / address

Call by Value

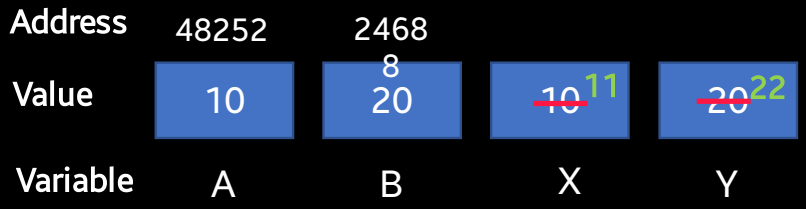
- In this approach, the values are passed as function argument to the definition of function.

Program

```
1  #include<stdio.h>
2  void fun(int,int);
3  int main()
4  {
5          int A=10,B=20;
6          printf("\nValues before calling %d, %d",A,B);
7          fun(A,B);
8          printf("\nValues after calling %d, %d",A,B);
9          return 0;
10 }
11 void fun(int X,int Y)
12 {
13         X=11;
14         Y=22;
15 }
```

Output

```
Values before calling 10, 20
Values after calling 10, 20
```



Call by Reference / Address

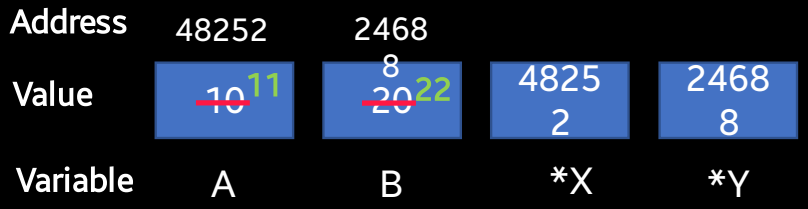
- In this approach, the references / addresses are passed as function argument to the definition of function.

Program

```
1  #include<stdio.h>
2  void fun(int*,int*);
3  int main()
4  {
5          int A=10,B=20;
6          printf("\nValues before calling %d, %d",A,B);
7          fun(&A,&B);
8          printf("\nValues after calling %d, %d",A,B);
9          return 0;
10 }
11 void fun(int *X,int *Y)
12 {
13         *X=11;
14         *Y=22;
15 }
```

Output

```
Values before calling 10, 20
Values after  calling 11, 22
```



Pointer to Function

- Every function has reference or address, and if we know the reference or address of function, we can access the function using its reference or address.
- This is the way of accessing function using pointer.

Syntax

```
1  return-type (*ptr-function)(argument list);
```

- return-type: Type of value function will return.
- argument list: Represents the type and number of value function will take, values are sent by the calling statement.
- (*ptr-function): The parentheses around *ptr-function tells the compiler that it is pointer to function.
- If we write *ptr-function without parentheses then it tells the compiler that ptr-function is a function that will return a pointer.

Practice Programs

1. Write a C program to print the address of variable using pointer.
2. Write a C a program to swap two elements using pointer.
3. Write a C a program to print value and address of a variable
4. Write a C a program to calculate sum of two numbers using pointer
5. Write a C a program to swap value of two numbers using pointer
6. Write a C a program to calculate sum of elements of an array using pointer
7. Write a C a program to swap value of two variables using function
8. Write a C a program to print the address of character and the character of string using pointer
9. Write a C a program for sorting using pointer

Dynamic Memory Allocation

Prof. Keyur Prajapati

Computer Engineering Department,
MBIT

Dynamic Memory Allocation (DMA)

- If memory is allocated at runtime (during execution of program) then it is called dynamic memory.
- It allocates memory from **heap** (*heap*: it is an empty area in memory)
- Memory can be accessed only through a pointer.

When DMA is needed?

- It is used when number of variables are not known in advance or large in size.
- Memory can be allocated at any time and can be released at any time during runtime.

malloc() function

- malloc () is used to allocate a fixed amount of memory during the execution of a program.
- malloc () allocates size_in_bytes of memory from heap, if the allocation succeeds, a pointer to the block of memory is returned else NULL is returned.
- Allocated memory space may not be contiguous.
- Each block contains a size, a pointer to the next block, and the space itself.
- The blocks are kept in ascending order of storage address, and the last block points to the first.
- The memory is not initialized.

Syntax	Description
ptr_var = (cast_type *) malloc (size_in_bytes);	This statement returns a pointer to size_in_bytes of uninitialized storage, or NULL if the request cannot be satisfied.

Example: fp = (int *)malloc(sizeof(int) *20);

Write a C program to allocate memory using malloc.

Program

```c
#include <stdio.h>
void main()
{
    int *fp; //fp is a pointer variable
    fp = (int *)malloc(sizeof(int)); //returns a pointer to int size storage
    *fp = 25; //store 25 in the address pointed by fp
    printf("%d", *fp); //print the value of fp, i.e. 25
    free(fp); //free up the space pointed to by fp
}
```

Output

```
25
```

calloc() function

- calloc() is used to allocate a block of memory during the execution of a program
- calloc() allocates a region of memory to hold no_of_blocks of size_of_block each, if the allocation succeeds then a pointer to the block of memory is returned else NULL is returned.
- The memory is initialized to ZERO.

Syntax	Description
ptr_var = (cast_type *) **calloc** (no_of_blocks, size_of_block);	This statement returns a pointer to no_of_blocks of size size_of_blocks, it returns NULL if the request cannot be satisfied.

Example:

```
int n = 20;
fp = (int *)calloc(n, sizeof(int));
```

Write a C program to allocate memory using calloc.

Program

```c
#include <stdio.h>
void main()
{
    int i, n; //i, n are integer variables
    int *fp; //fp is a pointer variable
    printf("Enter how many numbers: ");
    scanf("%d", &n);
    fp = (int *)calloc(n, sizeof(int)); //calloc returns a pointer to n blocks
    for(i = 0; i < n; i++)  //loop through until all the blocks are read
    {
        scanf("%d",fp); //read and store into location where fp points
        fp++; //increment the pointer variable
    }
    free(fp); //frees the space pointed to by fp
}
```

realloc() function

- realloc() changes the size of the object pointed to by pointer fp to specified size.
- The contents will be unchanged up to the minimum of the old and new sizes.
- If the new size is larger, the new space will be uninitialized.
- realloc() returns a pointer to the new space, or NULL if the request cannot be satisfied, in which case *fp is unchanged.

Syntax	Description
ptr_var = (cast_type *) realloc (void *fp, size_t);	This statement returns a pointer to new space, or NULL if the request cannot be satisfied. **Example**: fp = (int *)realloc(fp,sizeof(int)*20);

Write a C program to allocate memory using realloc.

Program

```c
#include <stdio.h>
void main()
{
    int *fp; //fp is a file pointer
    fp = (int *)malloc(sizeof(int)); //malloc returns a pointer to int size storage
    *fp = 25; //store 25 in the address pointed by fp
    fp =(int *)realloc(fp, 2*sizeof(int)); //returns a pointer to new space
    printf("%d", *fp); //print the value of fp
    free(fp); //free up the space pointed to by fp
}
```

Output

```
25
```

free() function

- free deallocates the space pointed to by fp.
- It does nothing if fp is NULL.
- fp must be a pointer to space previously allocated by calloc, malloc or realloc.

Syntax	Description
`void free(void *);`	This statement free up the memory not needed anymore. **Example**: free(fp);

Write a C program to sort numbers using malloc

Program

```c
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int i,j,t,n;
    int *p;
    printf("Enter value of n: ");
    scanf("%d", &n);
    p=(int *) malloc(n * sizeof(int));
    printf("Enter values\n");
    for(i=0; i<n; i++)
        scanf("%d", &p[i]);
    for(i=0; i<n; i++)
    {
        for(j= i+1; j<n; j++)
        {
```

Program (cont.)

```c
            if(p[i] > p[j])
            {
                t = p[i];
                p[i] = p[j];
                p[j] = t;
            }
        }
    }
    printf("Ascending order\n");
    for(i=0; i<n; i++)
        printf("%d\n", p[i]);
    free(p);
}
```

Write a C program to find square of numbers using calloc

Program

```c
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int i,n;
    int *p;
    printf("Enter value of n: ");
    scanf("%d",&n);
    p=(int*)calloc(n,sizeof(int));
    printf("Enter values\n");
    for(i=0;i<n;i++)
        scanf("%d",&p[i]);
    for(i=0;i<n;i++)
        printf("Square of %d = %d\n", p[i], p[i] * p[i]);
    free(p);
}

```

Output

```
Enter value of n: 3
Enter values
3
2
5
Square of 3 = 9
Square of 2 = 4
Square of 5 = 25
```

Write a C program to add/remove item from a list using realloc

Program

```c
#include<stdio.h>
#include<stdlib.h>
void main()
{
   int i, n1, n2;
   int *fp;
   printf("Enter size of list: ");
   scanf("%d", &n1);
   fp=(int *) malloc (n1 * sizeof(int));

   printf("Enter %d numbers\n", n1);
   for(i = 0; i < n1; i++)
      scanf("%d", &fp[i]);

   printf("The numbers in the list are\n");
   for(i = 0; i < n1; i++)
      printf("%d\n", fp[i]);
```

Program (cont.)

```c

   printf("Enter new size of list: ");
   scanf("%d", &n2);

   fp = realloc(fp, n2 * sizeof(int));
   if(n2 > n1)
   {
      printf("Enter %d numbers\n", n2 - n1);
      for(i = n1; i < n2; i++)
         scanf("%d", &fp[i]);
   }
   printf("The numbers in the list are\n");
   for(i = 0; i < n2; i++)
      printf("%d\n", fp[i]);
}
```

Practice Programs

1) Write a C program to calculate sum of n numbers entered by user.
2) Write a C program to input and print text using DMA
3) Write a C program to read and print student details using structure and DMA

File Management

Prof. Keyur Prajapati

Computer Engineering Department,
MBIT

Why File Management?

- In real life, we want to store data permanently so that later we can retrieve it and reuse it.
- A file is a collection of characters stored on a secondary storage device like hard disk, or pen drive.
- There are two kinds of files that programmer deals with:
  - Text Files are human readable and it is a stream of plain English characters
  - Binary Files are computer readable, and it is a stream of processed characters and ASCII symbols

Text File

```
Hello, this is a text file.
Whatever written here can be
read easily without the help of a
computer.
```

Binary File

```
11010011010100010110111010
10111010111010011010100010
11011101010111010111010011
```

File Opening Modes

- We can perform different operations on a file based on the file opening modes

Mode	Description
r	Open the file for reading only. If it exists, then the file is opened with the current contents; otherwise an error occurs.
w	Open the file for writing only. A file with specified name is created if the file does not exists. The contents are deleted, if the file already exists.
a	Open the file for appending (or adding data at the end of file) data to it. The file is opened with the current contents safe. A file with the specified name is created if the file does not exists.
r+	The existing file is opened to the beginning for both reading and writing.
w+	Same as w except both for reading and writing.
a+	Same as a except both for reading and writing.

Note: The main difference is w+ truncate the file to zero length if it exists or create a new file if it doesn't. While r+ neither deletes the content nor create a new file if it doesn't exist.

File Handling Functions

- Basic file operation performed on a file are opening, reading, writing, and closing a file.

Syntax	Description
fp=fopen(file_name, mode);	This statement opens the file and assigns an identifier to the FILE type pointer fp. Example: fp = fopen("printfile.c","r");
fclose(filepointer);	Closes a file and release the pointer. Example: fclose(fp);
fprintf(fp, “control string”, list);	Here fp is a file pointer associated with a file. The control string contains items to be printed. The list may includes variables, constants and strings. Example: fprintf(fp, "%s %d %c", name, age, gender);

File Handling Functions

Syntax	Description
fscanf(fp, “control string”, list);	Here fp is a file pointer associated with a file. The control string contains items to be printed. The list may includes variables, constants and strings. Example: fscanf(fp, "%s %d", &item, &qty);
int getc(FILE *fp);	getc() returns the next character from a file referred by fp; it require the FILE pointer to tell from which file. It returns EOF for end of file or error. Example: c = getc(fp);
int putc(int c, FILE *fp);	putc() writes or appends the character c to the FILE fp. If a putc function is successful, it returns the character written, EOF if an error occurs. Example: putc(c, fp);

File Handling Functions

Syntax	Description
int getw(FILE *pvar);	getw() reads an integer value from FILE pointer fp and returns an integer. Example: i = getw(fp);
putw(int, FILE *fp);	putw writes an integer value read from terminal and are written to the FILE using fp. Example: putw(i, fp);
EOF	EOF stands for “End of File”. EOF is an integer defined in <stdio.h> Example: while(ch != EOF)

Write a C program to display content of a given file.

Program

```c
#include <stdio.h>
void main()
{
    FILE *fp; //p is a FILE type pointer
    char ch; //ch is used to store single character
    fp = fopen("file1.c","r"); //open file in read mode and store file pointer in p
    do {  //repeat step 9 and 10 until EOF is reached
        ch = getc(fp); //get character pointed by p into ch
        putchar(ch); //print ch value on monitor
    }while(ch != EOF); //condition to check EOF is reached or not
    fclose(fp); //free up the file pointer pointed by fp
}

```

Write a C program to copy a given file.

Program

```c
#include <stdio.h>
void main()
{
    FILE *fp1, *fp2; //p and q is a FILE type pointer
    char ch; //ch is used to store temporary data
    fp1 = fopen("file1.c","r"); //open file “file1.c” in read mode
    fp2 = fopen("file2.c","w"); //open file “file2.c” in write mode
    do { //repeat step 9 and 10 until EOF is reached
        ch = getc(fp1); //get character pointed by p into ch
        putc(ch, fp2); //print ch value into file, pointed by pointer q
    }while(ch != EOF); //condition to check EOF is reached or not
    fclose(fp1); //free up the file pointer p
    fclose(fp2); //free up the file pointer q
    printf("File copied successfully...");
}
```

File Positioning Functions

- fseek, ftell, and rewind functions will set the file pointer to new location.
- A subsequent read or write will access data from the new position.

Syntax	Description
fseek(FILE *fp, long offset, int position);	fseek() function is used to move the file position to a desired location within the file. fseek(fp, 5, SEEK_SET);
long ftell(FILE *fp);	ftell takes a file pointer and returns a number of datatype long, that corresponds to the current position. This function is useful in saving the current position of a file. **Example**: /* n would give the relative offset of the current position. */ n = ftell(fp);

File Positioning Functions

Syntax	Description
rewind(fp);	rewind() takes a file pointer and resets the position to the start of the file. **Example**: /* The statement would assign 0 to n because the file position has been set to the start of the file by rewind. */ rewind(fp);

Write a C program to count lines, words, tabs, and characters

Program

```
1  #include <stdio.h>
2  void main()
3  {
4      FILE *p;
5      char ch;
6      int ln=0,t=0,w=0,c=0;
7      p = fopen("text1.txt","r");
8      ch = getc(p);
9      while (ch != EOF) {
10         if (ch == '\n')
11             ln++;
12         else if(ch == '\t')
13             t++;
14         else if(ch == ' ')
15             w++;
16         else
```

Program (contd.)

```
17             c++;
18         ch = getc(p);
19     }
20     fclose(p);
21     printf("Lines = %d, tabs = %d, words = %d,
22  characters = %d\n",ln, t, w, c);
   }

23
```

Output

```
Lines = 22, tabs = 0, words = 152, characters =
283
```

Practice Programs

1) Write a C program to write a string in file.
2) A file named data contains series of integer numbers. Write a C program to read all numbers from file and then write all the odd numbers into file named “odd” and write all even numbers into file named “even”. Display all the contents of these file on screen.
3) Write a C program to read name and marks of n number of students and store them in a file.
4) Write a C program to print contents in reverse order of a file.
5) Write a C program to compare contents of two files.
6) Write a C program to copy number of bytes from a specific offset to another file.
7) Write a C program to convert all characters in UPPER CASE of a File.

Thank you