

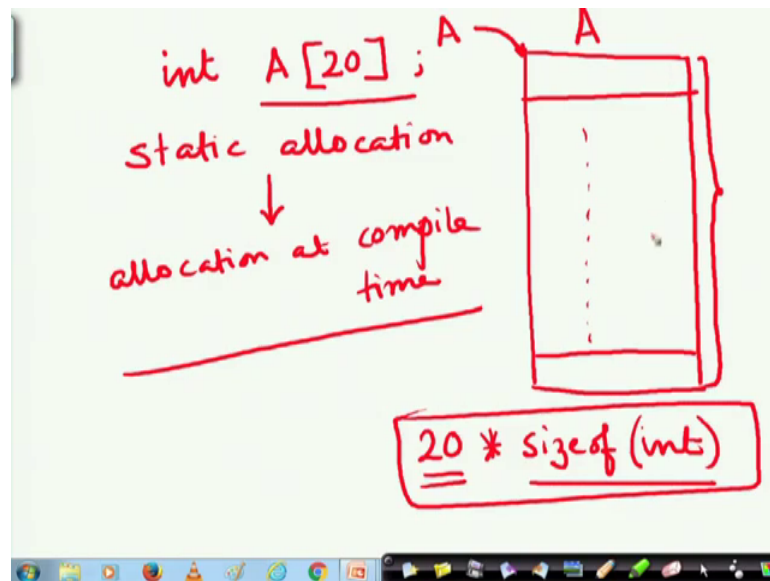
**Problem Solving through Programming In C**  
**Prof. Anupam Basu**  
**Department of Computer Science & Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 61**  
**Dynamic Allocation and File**

We have looked at pointers and structures in detail and we have also seen how structures can also utilize pointers or in other words how pointers can be used in conjunction with structures. Now, we look at another very interesting use of pointers, but in general let me say that it is a very fundamental concept, from the memory allocation point of view.

Dynamic memory allocation is what we will be discussing now. Now, when what is when we say that is dynamic memory allocation then; obviously, there must be something called the static memory allocation now what is static memory allocation?

(Refer Slide Time: 01:07)



When we declare an array say `int A 20`, you know that the compiler will allocate 20 locations, 20 locations to house 20 integers to you and that will be named as `A` or you can also consider that there is a pointer `A`, which is pointing to the first element of the array. But you have got 20 space for 20 locations 20 integers allocated to you.

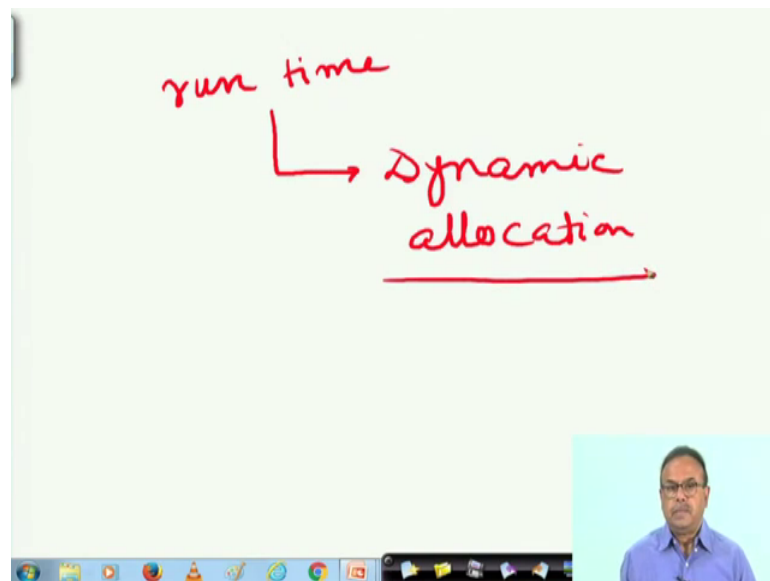
So, now, you know size of. So, how many bytes will be required? You can say 20 times size of `int`. So, size of `int` will return you how many bytes your particular system

allocates for an integer and 20 such allocation. So, so many bytes will be allocated to you. Now when that is statically allocated that is allocated at compile and compile time; so, when we say static allocation that means, allocation at compile time right.

Now, if for some reason you need more than 20 integers to be stored in this array A that you will need to redefine this whole thing or in some cases we do not know we do not have an idea of what will be the how many data items will come for example, you was you are actually storing student data's student records in an array class, and you do not know how many students will join that class beforehand.

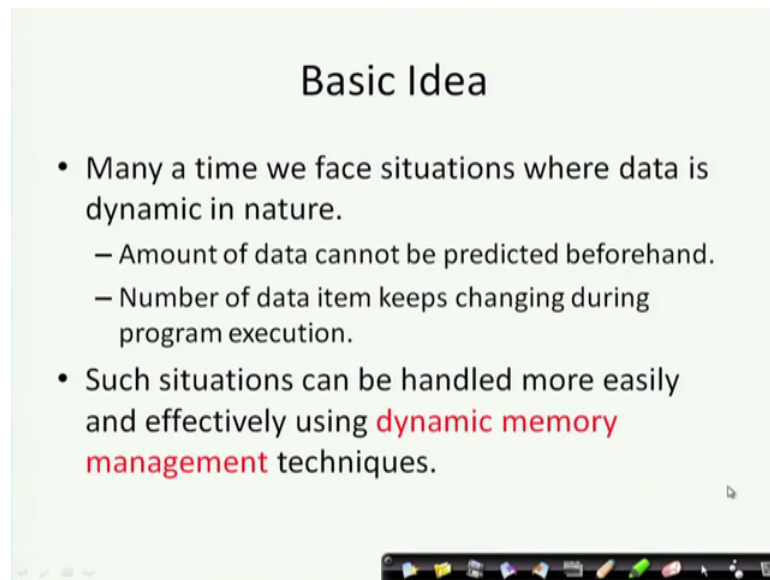
If you have if you know that beforehand its fine or if you have an idea that what is the maximum amount maximum number of students that can come, then its fine you can allocate it in the form of static allocation as we do in an array.

(Refer Slide Time: 03:57)



However, when we do not know and the information comes at a runtime; that means, when is being executed, that will lead to what we call dynamic allocation of memory that is dynamic allocation. So, let us look at how we can handle it.

(Refer Slide Time: 04:23)

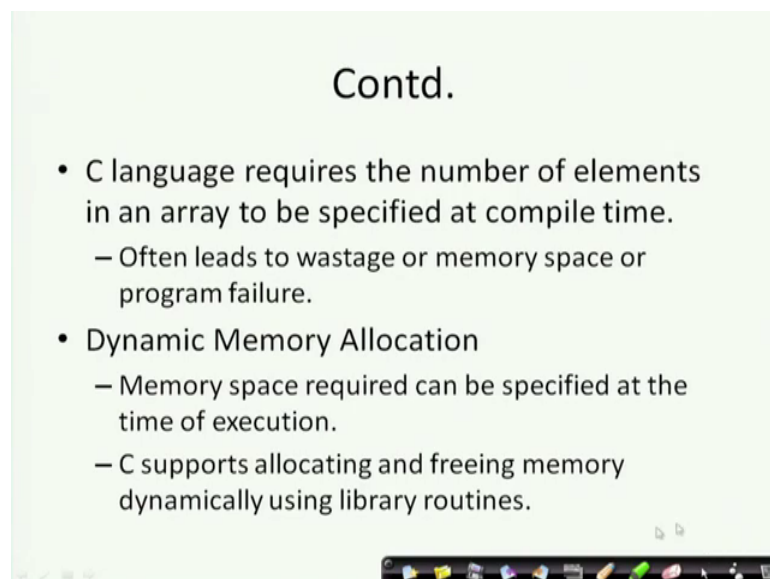


### Basic Idea

- Many a time we face situations where data is dynamic in nature.
  - Amount of data cannot be predicted beforehand.
  - Number of data item keeps changing during program execution.
- Such situations can be handled more easily and effectively using **dynamic memory management** techniques.

So, the basic idea is I have already explained that the amount of data we cannot predict beforehand. So, we will use effectively whose dynamic memory management, memory allocation technique to do that.

(Refer Slide Time: 04:36).



### Contd.

- C language requires the number of elements in an array to be specified at compile time.
  - Often leads to wastage or memory space or program failure.
- Dynamic Memory Allocation
  - Memory space required can be specified at the time of execution.
  - C supports allocating and freeing memory dynamically using library routines.

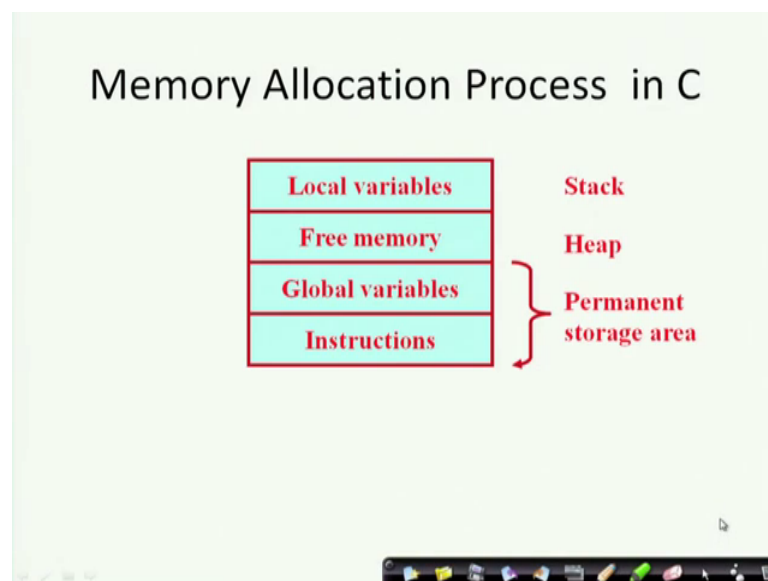
Now, C language requires a number of elements to be specified in compiled time when we define in an array we need to specify that in compile time. Now, often that leads to wastage of memory or program failure why program failure? Program failure because if we exceed the amount of space that has been allocated, there will be a failure the

program will give an error or it will exit abnormally. However, if we take recourse to dynamic memory allocation, we can solve this problem how? Memory space required can be specified at the time of execution how can we do that, how can I specify the amount of memory required at the time of execution?

If while running the program, just like the instructions and operators if we had some special means some special command, some special operator by which we can grab memory. Now here you should understand that who allocates memory to us it is the operating system who allocates the memory to us.

So, this like the `printf`, `scanf` all those things are system calls we are calling the we are making calls to the operating system, which is doing the required thing for us similarly there is a function called `Malloc` Memory Allocated, `mMalloc` using which we can grab memory from the operating system how let us look at this say.

(Refer Slide Time: 06:25)

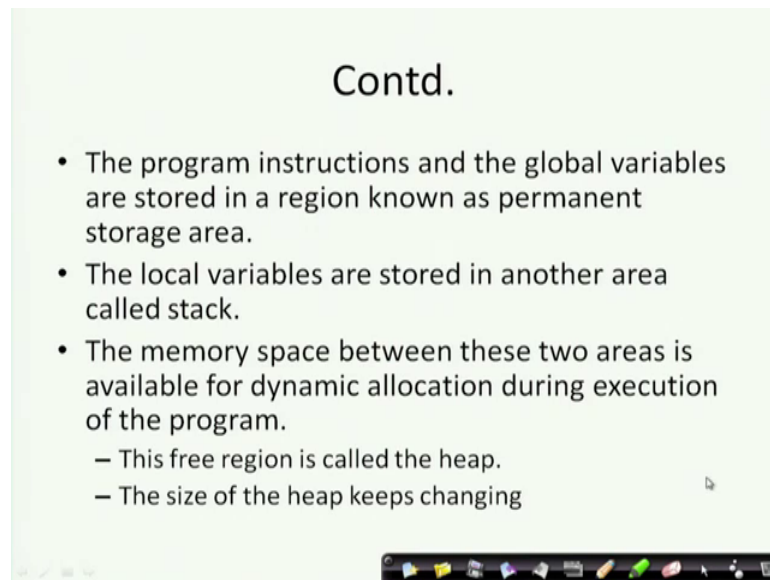


In memory in C, I have got different types of variables I that is not so, much relevant right now what is needed is part this global variables and instructions are there always told.

(Refer Slide Time: 06:37)

### Contd.

- The program instructions and the global variables are stored in a region known as permanent storage area.
- The local variables are stored in another area called stack.
- The memory space between these two areas is available for dynamic allocation during execution of the program.
  - This free region is called the heap.
  - The size of the heap keeps changing

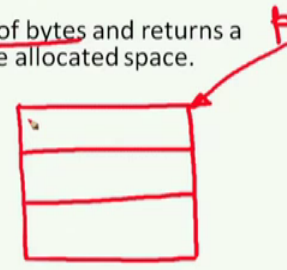


But the local variables are kept the local variables are there and there is some free memory. We can take from this free memory and put it use them as our local variables when free region is has got a name heap.

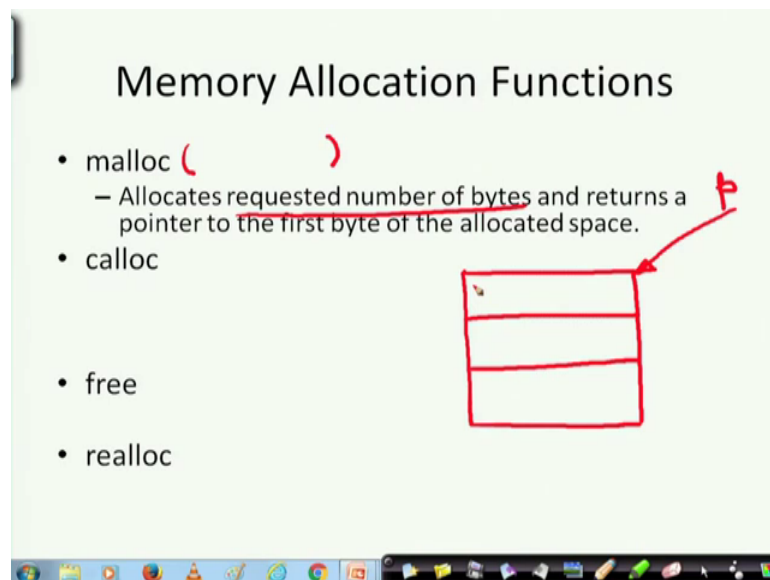
(Refer Slide Time: 07:09)

### Memory Allocation Functions

- malloc ( )
  - Allocates requested number of bytes and returns a pointer to the first byte of the allocated space.
- calloc
- free
- realloc



The diagram shows a rectangular box divided into three horizontal sections. A red arrow labeled 'p' points to the top-left corner of the box, indicating the start of the allocated memory.



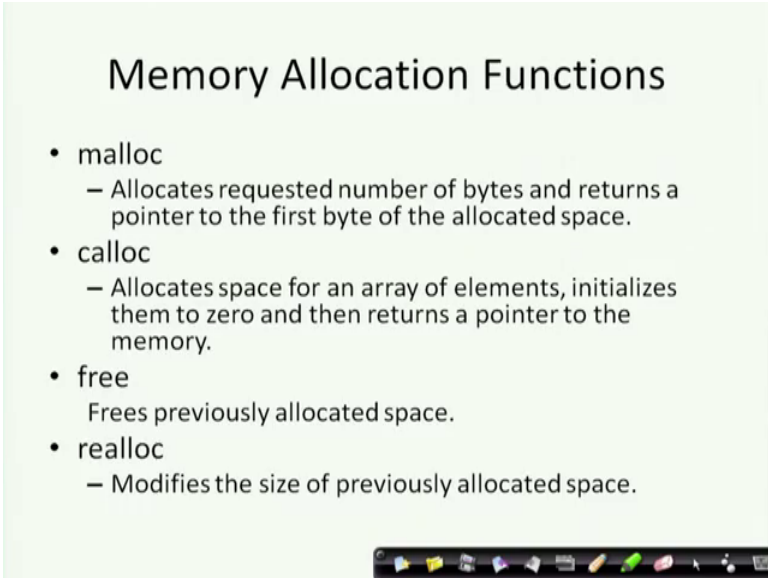
Now, the most important thing is that we need some functions, which will give the memory gave my program some memory addresses or memory blocks from the operating system storage of memory, which is known as heap from there it will come to be allocated to my program all right.

So, for that we have got four different functions one is malloc what does malloc do? Malloc allocates the requested number of bytes and returns a pointer to the first byte of the allocated space. So, what it does is something like this, what is happening? (Refer Time: 08:31). Malloc allocates a requested number of bytes and returns a pointer to the first byte of the allocated space. So, let us try to explain this.

So, when I do malloc, malloc is just like a function will return me some memory bytes all right some memory bytes how many memory bytes will depend on, how what I am what I am requesting for? It is a requested number of bytes. So, malloc will have some parameters which will show later.

So, it will give some give me some amount of memory and how do I know? Now, the operating system has got some free memory spread here and there. So, from there it is giving me some piece of memory, but how do I know where is that piece of memory for that, it is returning me a pointer say pointer p which is telling me that this if you follow this pointer, you will get this piece of memory location all right. So, let us proceed a little bit.

(Refer Slide Time: 09:46)



### Memory Allocation Functions

- malloc
  - Allocates requested number of bytes and returns a pointer to the first byte of the allocated space.
- calloc
  - Allocates space for an array of elements, initializes them to zero and then returns a pointer to the memory.
- free
  - Frees previously allocated space.
- realloc
  - Modifies the size of previously allocated space.

Now, similarly now when this is given that memory block is given in response to malloc request, the actual memory is not initialized to some value it can have any garbage value. But if I apply calloc; calloc then it allocates space for the array of elements, array of elements and initializes them to zero and returns a pointer. So, in this case if I want to

have a chunk of memory where everything has been initialized to zero then I should use calloc on the other hand this free what it does the free function call will return this amount of memory, that was given to me in request to malloc it will be returned back to the heap returned back to the operating system so, that it can be utilized by somebody else in future all right.

So, and realloc modifies the size of the previously allocated space. So, I have got some allocation and then I think that allocation is not enough I want to change it I can use realloc. However, we will be mostly concerned with malloc and free in our discussion.

(Refer Slide Time: 11:14)

**Allocating a Block of Memory**

- A block of memory can be allocated using the function **malloc**.
  - Reserves a block of memory of specified size and returns a pointer of type **void**.
  - The return pointer can be assigned to any pointer type.
- General format:  

```
ptr = (type *) (malloc (byte_size))
```

*Handwritten annotations in red:*  
- *pointer of type void* (with a slash over *void*)  
- *int*  
- *(int \*)* (underlined)

So, a block of memory can be allocated using the function malloc and it reserves a block of memory and returns a pointer of type void. You know every pointer has got some type, but in this case with malloc returns some memory block the pointer that it is returned is of type void, but then we have to do something what we need to do? I know why I needed this memory I need. So, accordingly I will have to do that typecasting all right. So, now the return once it returns me of type void, but that return pointer can be assigned to any pointer type. So, here you see.

So, here please note malloc has got a parameter byte size how much memory I want, how many bytes I want. Now this malloc has returned me up to this, it has returned me a pointer and pointer is of type void, but I type suppose if this amount of memory I want for the purpose of storing integer array then this type will be int star; that means, I am

casting this what is malloc returning? Malloc is returning a pointer some pointer, but that pointer was of type void of type void. Now when I am typecasting it to int star, then this void is no longer it is of type is becoming of type int and then I am assigning it to another variable p t r.

So, think of two things, first of all you have to decide on how many bytes you want accordingly you do malloc and then what type of data you want to store there. So, accordingly you do this typecasting like int star, float star, char star whatever you do and then you assign it to a particular pointer let us see how it will work. So, let us look an example.

(Refer Slide Time: 13:55)

Contd.

- Examples

```
p = (int *) malloc (100 * sizeof (int));
```

The diagram shows a red box containing the number '400'. Above the box is the number '4'. A red arrow points from the letter 'p' to the top-right corner of the box. A horizontal line is drawn below the '100' in the code above.

Here how do I know how many bytes I need? Suppose I need to store an element an array of 100 integers. So, what I do here is I ask for malloc 100 times size of int size of int if it is 4, if int is 4 then I am getting 400 bytes.

Now, these 400 bytes that have been given to me is being pointed we by some pointer of type void. So, next I make it int star and put it to p. So, p is now an integer point integer pointer that is pointing to this entire block of 400 integers.



(Refer Slide Time: 14:49)

Contd.

- Examples
  - `p = (int *) malloc (100 * sizeof (int));`
    - A memory space equivalent to "100 times the size of an int" bytes is reserved.
    - The address of the first byte of the allocated memory is assigned to the pointer p of type int.

The diagram illustrates the memory allocation process. A pink square labeled 'p' represents the pointer variable. A red arrow points from 'p' to a horizontal row of five cyan rectangles, representing the allocated memory. Below this row, the text '400 bytes of space' is written. A red squiggly line is drawn above the number '100' in the code snippet above.

So, a memory space equivalent to 100 times the size, we have got. So, here 400 bytes of space and p is a pointer pointing to the beginning of this. So, I have sorry I have got this just using malloc, it was not declared beforehand.

Now so, this 100 can also be a variable n if I read a particular variable n now how many students are there scan f m and n. So, I read the number of students, then I can multiply that with n as well ok.

(Refer Slide Time: 15:35)

Contd.

```
cptr = (char *) malloc (20);
```

- Allocates 10 bytes of space for the pointer cptr of type char.

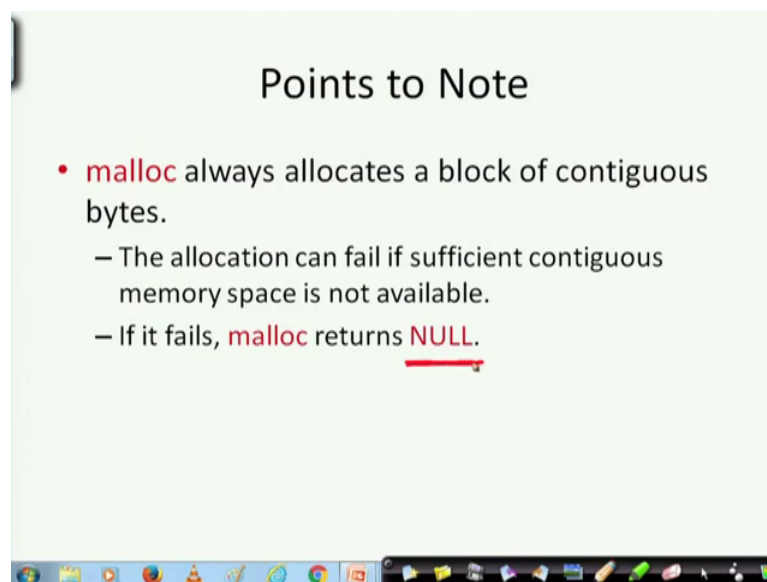
```
sptr = (struct stud *) malloc (10 * sizeof (struct stud));
```

The diagram shows the calculation of memory size for a struct pointer. The code snippet `sizeof (struct stud)` is underlined, and a red bracket underneath it is labeled '40'. The number '10' in `10 *` is also underlined, and a red squiggly line is drawn above it.

Next you see here I am initializing to I am I am seeking memory for 20 characters, I do malloc 20 because I know a character takes one byte and then the pointer is of type void I am typecasting it to type character char star and assigning it to c p t r. Now it is actually wrong it is allocating 20 bytes of space for the pointer.

So, structure stud now for example, I now need. So, integer character was simple now I want to have for space for the entire structure students, now that size is larger. So, I do not know. So, I just have employ this function size of struct stud. So, I gets how many bytes it requires say 40 bytes and say 10 such for 10 such students or n such students I multiply with that I get so, much memory, now I have to typecast that to struct stud star and that goes to a as a structure pointer.

(Refer Slide Time: 17:00)



Now, malloc always allocates a block of contiguous bytes. Now, it may be that sufficient suppose you are asking for 100 bytes and 100 bytes are not available then malloc will not be able to allocate you the space, in that case malloc will return in null that is a null pointer that is a special character special value it will return, that shows that I could not allocate a space.

So, I could not allocate it to you a valid pointer. So, it is a null pointer meaning thereby that I could not I failed in allocating you memory.

(Refer Slide Time: 17:44)

```
Example

#include <stdio.h>
main()
{
    int i,N;
    float *height;
    float sum=0;

    printf("Input heights for %d\n",N);
    for(i=0;i<N;i++)
        scanf("%f",&height[i]);

    for(i=0;i<N;i++)
        sum+=height[i];

    printf("Input the number of students. \n");
    scanf("%d",&N);

    height=(float *) malloc(N * sizeof(float));

    printf("Average height= %f\n",avg);
}

Input the number of students.
5
Input heights for 5 students
23 24 25 26 27
Average height= 25.000000
```

So, here is an example here you can see let us look at from one side I n float is a pointer height is a pointer of type float sum is 0 and average. So, what I am trying to do probably I am trying to find the average height of the class. So, input the number of students and I am reading ampersand. So, this one is ampersand n. So, this is n number of students.

Now, see I did not know how many students are there. So, I am getting this number of students here, I am getting n number of students here. Now I want to have so, many spaces for the height. So, what I am doing? I am allocating n number of spaces n is a variable here and size of float whatever size of float is 4 bytes. So, n times 4 bytes so much space is being allocated, and the pointer is height is a pointer of type float.

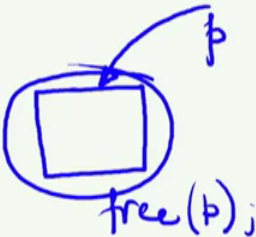
So, this pointer is being type casted to float all right then I get the input scan f in a loop, I am getting the heights one after another in an array and I am finding the sum of the heights finding the average of the heights, where I am dividing sum which is a floating point number with float n here is another example of typecasting. So, you see I am dividing by n, but this is a floating point real number n was an integer. So, I convert it to float and convert it divided right.

So, this is how malloc works now. So, we have we have explained that.

(Refer Slide Time: 19:43)

## Releasing the Used Space

- When we no longer need the data stored in a block of memory, we may release the block for future use.
- How?
  - By using the **free** function.
- General format:  
`free (ptr);`



The diagram illustrates a pointer variable 'p' pointing to a rectangular memory block. Below the block, the code 'free(p);' is written, indicating the release of the memory block.

So, now how do we allocate space? Similarly, the general format for freeing space is using the free function. So, generally, suppose I have got a space allocated to me, some space is allocated to me and that space the only handle to that space is the pointer p.

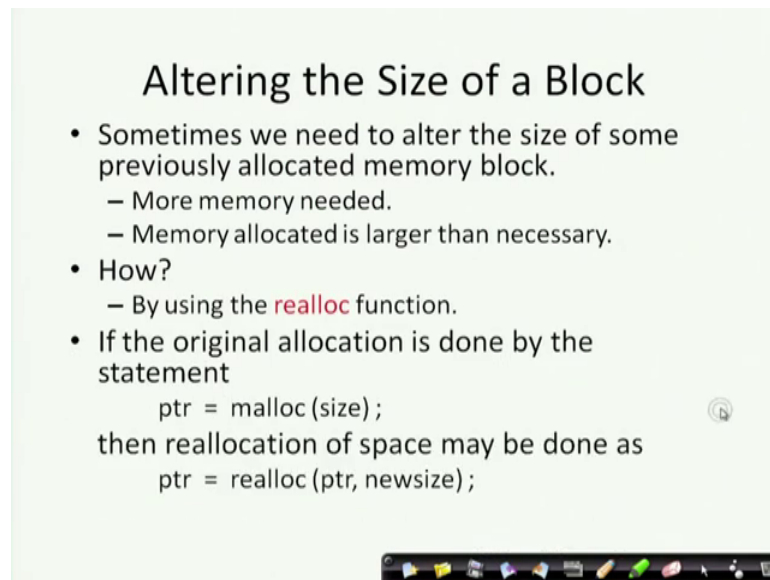
So, I free that pointer, I free p. So, the pointer is freed; that means, this pointer is freed means this location this amount of memory goes back to the storage of the operating system and that is the heap.

(Refer Slide Time: 20:30)

## Releasing the Used Space

- When we no longer need the data stored in a block of memory, we may release the block for future use.
- How?
  - By using the **free** function.
- General format:  
`free (ptr);`  
where ptr is a pointer to a memory block which has been already created using **malloc**.

(Refer Slide Time: 20:32)



### Altering the Size of a Block

- Sometimes we need to alter the size of some previously allocated memory block.
  - More memory needed.
  - Memory allocated is larger than necessary.
- How?
  - By using the `realloc` function.
- If the original allocation is done by the statement  

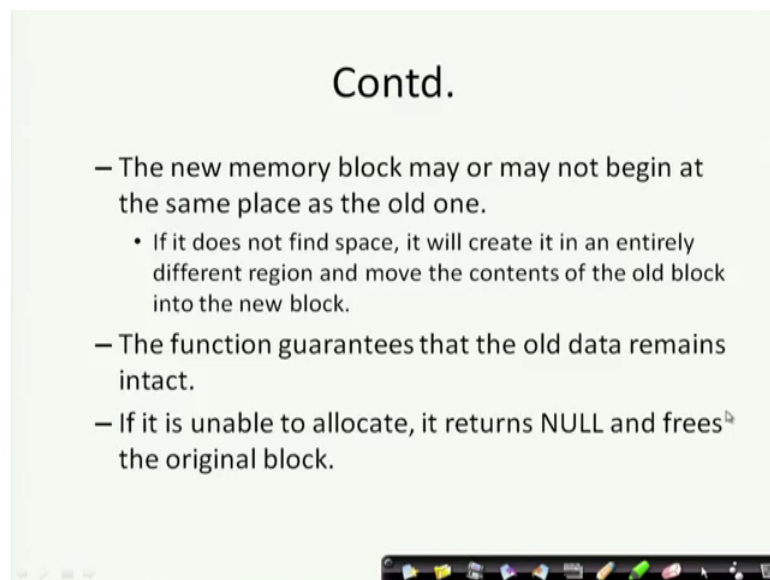
```
ptr = malloc (size);
```

then reallocation of space may be done as  

```
ptr = realloc (ptr, newsize);
```

So, we whatever we got in malloc; so, that gives us some idea about how we can get space and reallocate space.

(Refer Slide Time: 20:36)



### Contd.

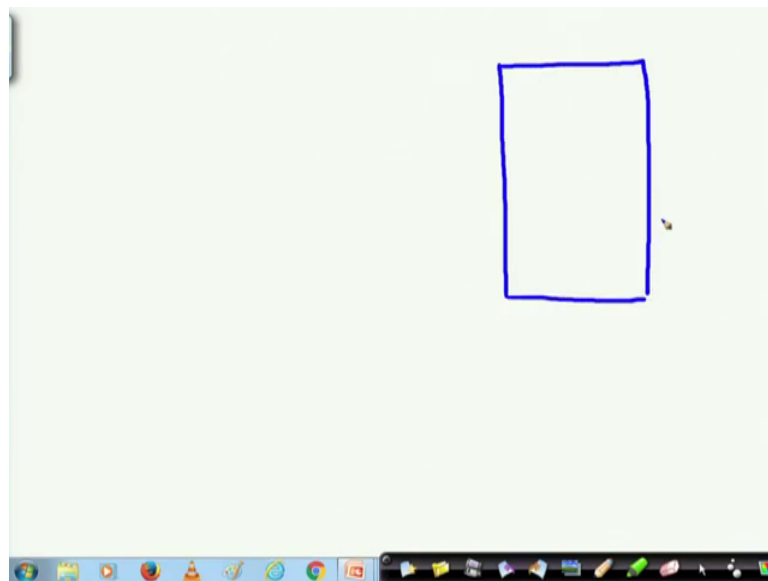
- The new memory block may or may not begin at the same place as the old one.
  - If it does not find space, it will create it in an entirely different region and move the contents of the old block into the new block.
- The function guarantees that the old data remains intact.
- If it is unable to allocate, it returns NULL and frees the original block.

Now, briefly we I will be talking for the next 5 or 10 minutes on file handling; there is not much to understand about file handling thus you will learn as you do. Now, what is a file that is something you have to understand. File is something where wherever I want to write something write or read from.

So, I want to store something I will take a particular file all right; I will take a particular file and I will open that file and then I will write in the into that file then close that file and then whenever I need in that way I may have 10 files.

Now, at a particular point of time I want to read a particular thing. So, I choose the particular file what do I do next? Open the file and read the file. Now, some files may be allowed to be read by others, some files can only be written in to and not read from, some files can have the option of read or write both.

(Refer Slide Time: 21:55)



So, file is some space where I will be writing or reading from some storage.

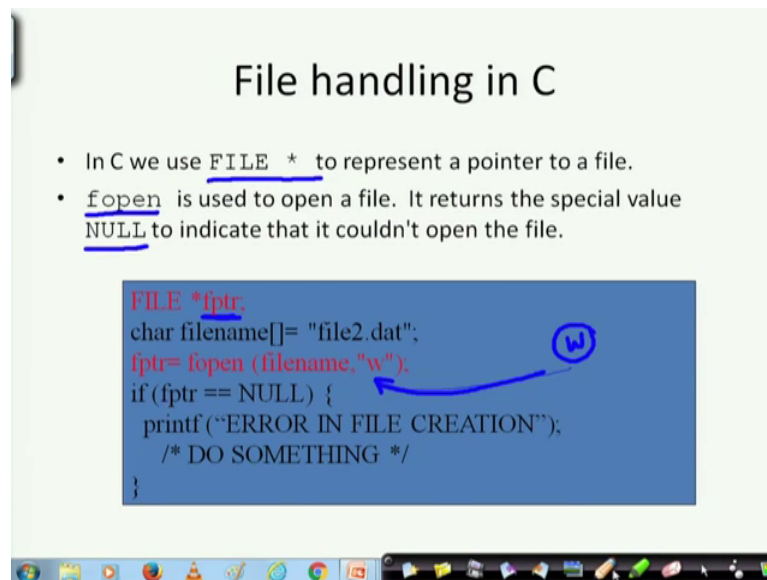
So, this is there in the secondary memory and what till now whatever variables we are talking about, those who are all in the primary memory. So, if I store it in a file it goes into the secondary memory. So, let us have a little idea of how files are handled.

(Refer Slide Time: 22:18)

## File handling in C

- In C we use FILE \* to represent a pointer to a file.
- fopen is used to open a file. It returns the special value NULL to indicate that it couldn't open the file.

```
FILE *fptr;
char filename[] = "file2.dat";
fptr = fopen(filename, "w");
if (fptr == NULL) {
    printf("ERROR IN FILE CREATION");
    /* DO SOMETHING */
}
```



So, now again now we have learnt pointers. So, any file can be accessed using a pointer. Just as if I have the file of income tax all right. So, I will have a pointer that there is the file of income tax, here there is some file of road tax it will be here some file of your salary it will be somewhere here, some file of your expenses it will be somewhere else.

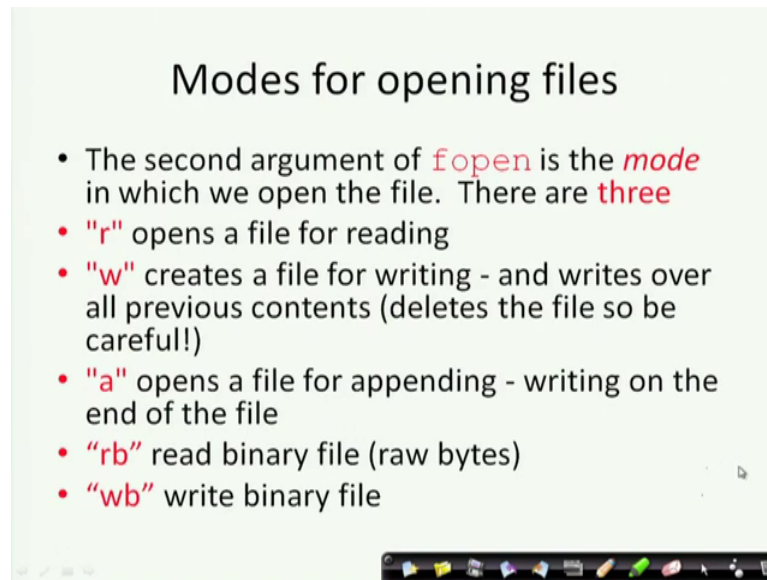
So, there will be pointers. So, we use in c the sorry we use file star we use file star to represent pointed to a file and if open is the command for opening a file. If a file cannot be opened then it will return a null just as in the case of malloc, we saw if nothing could be returned it was returning a null.

So, here for example, you see f p t r is a pointer of type file. That means, f p t r will be pointing to file. Now, I have got a character file name is an array file 2 dot dat it is a data file. So, fptr is f open file name and here when I do f open I give the file name as well as the mode in which it can be opened the mode in which it can be opened and this w means it is in the right mode. So, what have I done here I have called fopen.

So, I am trying to open the file if the f. So, this fopen will return up file pointer now if this file pointer is null; that means, there was some error in file creation otherwise it will go on doing something ok.

So, quickly let us look at this, when I do `fopen` it will open a file and will open it in a particular mode read or write whatever I specify and it will return me a pointer. If a file is created successfully it will return me a non null pointer all right.

(Refer Slide Time: 24:38)



The second argument of `fopen` is the mode and there are three modes there are three modes. `R` is the file is opened for reading, `w` means it creates a file for writing and writes over all the previous contents. So, if I open it in the write mode whatever content was in that file is erased. And `a` opens a file for appending; that means, whatever is there after that it will be added ok.

So, if you have got something already stored and you do not want to destroy that, and you want to add something more to that you will open it in the form in the mode `a` all right. And `rb` reads a binary file raw bytes we need not bother about that.



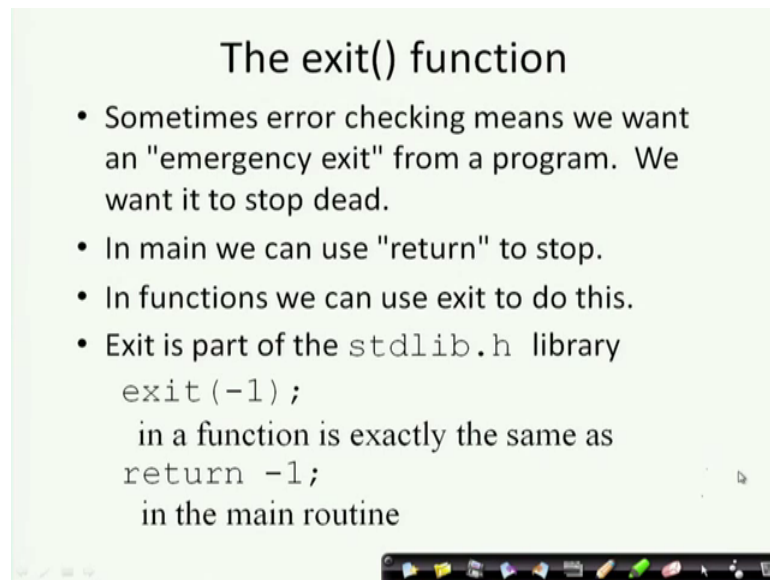
(Refer Slide Time: 25:25)

### The exit() function

- Sometimes error checking means we want an "emergency exit" from a program. We want it to stop dead.
- In main we can use "return" to stop.
- In functions we can use exit to do this.
- Exit is part of the `stdlib.h` library

```
exit(-1);
```

in a function is exactly the same as  
`return -1;`  
in the main routine

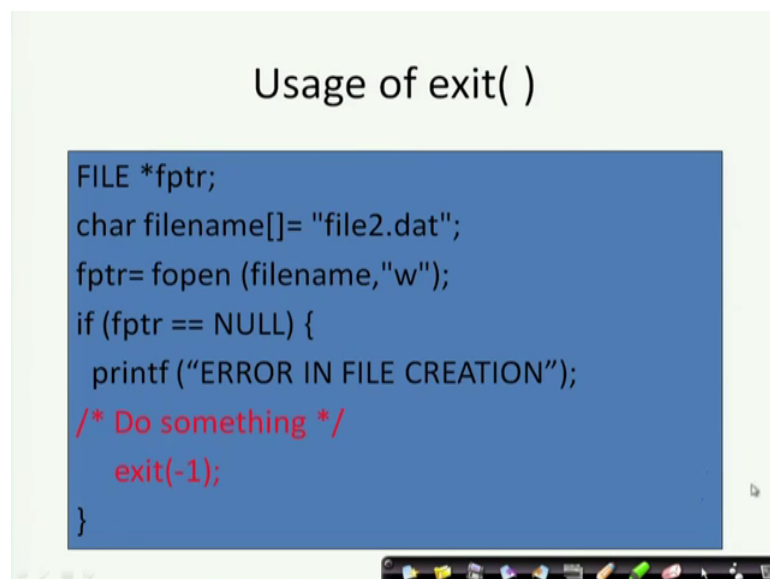


And there is a function called exit which you have seen, that exit for sometimes in the emergency we can put exit minus one; that means, it tells that I have exited the function without success.

(Refer Slide Time: 25:38)

### Usage of exit( )

```
FILE *fptr;  
char filename[] = "file2.dat";  
fptr = fopen(filename, "w");  
if (fptr == NULL) {  
    printf("ERROR IN FILE CREATION");  
    /* Do something */  
    exit(-1);  
}
```



Now, here you see use of exit file f pointer character file name is an array, file 2 dot dat I tried to do something. So, the file pointer was null.


So, if it be null then what can I do? I will have to exit because of some reason the file could not be created so, that apart.

(Refer Slide Time: 26:02)

### Writing to a file using fprintf( )

- **fprintf( )** works just like printf and sprintf except that its first argument is a file pointer.

```
FILE *fptr;  
fptr= fopen ("file.dat", "w");  
/* Check it's open */  
fprintf (fptr, "Hello World\n");
```



So, f open we have seen f print is a very important command f print works just like print f and s print f except that the first argument is a file pointer. So, we will see how it works. So, fptr is again the file pointer and I have opened the file dot dat in the right mode. Now, if print f means now I am printing where am I printing a file called file dot dat has been opened.

The name of the file is file dot dat and how do I identify it? I identify it with the fptr the file pointer ok. So, I am writing it is a open in the right mode. So, it is everything whatever was there has been erased. So, I am writing just as you have been done print f then automatically by default it goes to the screen here it is not default here I have said fptr.

So, whatever I write hello world it will be written in this file not in the screen all right.

(Refer Slide Time: 27:27)

### Reading Data Using fscanf( )

•We also read data from a file using fscanf( ).

```
FILE *fptr;
fptr= fopen ("input.dat","r");
/* Check it's open */
if (fptr==NULL)
{
    printf("Error in opening file \n");
}
fscanf(fptr,"%d%d",&x,&y);
```

The diagram illustrates the process of reading data from a file. A box labeled 'input.dat' contains the values '20 30'. An arrow points from this box to the fscanf function call in the code. Another arrow points from the fscanf call to variables 'x=20' and 'y=30', indicating that the data from the file is being stored in these variables.

Screen is another file, but that is a default file. Reading a data similarly we printed using f print f reading we can do using f scan f forget about that part look at this fptr I am reading from not from the keyboard now. I am now not reading from the keyboard, I am reading x and y two integers from a file which is pointed out by f p t r all right and what is that file f p t r? I have opened the file input dot dat.

So, you see in that file input dot dat 20 and 30 was written and so, f scan f I have read that that was input dot dat, from I have opened that in the read mode in the read mode and I am reading from there. So, I am getting x to be 20 and y to be 30 not from the keyboard, but from the file.

(Refer Slide Time: 28:21)

### Reading lines from a file using fgets( )

We can read a string using **fgets( )**.

```
FILE *fptr;  
char line [1000];  
/* Open file and check it is open */  
while (fgets(line,1000,fptr) != NULL) {  
    printf("Read line %s\n",line);  
}
```

**fgets( )** takes 3 arguments, a string, a maximum number of characters to read and a file pointer. It returns NULL if there is an error (such as EOF).

So, in that way we can now here are some powerful commands just to know, we can read a string using if gets from a file I can read a string. So, here you see a file is f p t r and a line is of size 1000, while f gets line is not null; that means, I am getting from f p t r, I am getting a value and if it is not null; that means, it is not the end of the line.

I mean I have got the file is open I am reading the line using f gets all right I am getting the line f gets takes three arguments what are the three arguments its taking a string a maximum number of characters 1000 and a pointer from it returns. If there is an error such as end of it end of file EOF is end of file.

(Refer Slide Time: 29:22)

## Closing a file

- We can close a file simply using `fclose()` and the file pointer.

```
FILE *fptr;
char filename[] = "myfile.dat";
fptr = fopen (filename, "w");
if (fptr == NULL) {
    printf ("Cannot open file to write!\n");
    exit(-1);
}
fprintf (fptr, "Hello World of filing!\n");
fclose (fptr);
```

**Opening**      **Access**      **closing**

Now, this I think you can understand much better when you use, now when we open a file after that we must close that file we can simply use a command `f close` and the file pointer. So, here you see `f p t r` I opened the file in the right mode, I have print written hello world over here `f print f` means saying I am printing in the file, and then I am `f close` I am doing `f closing` the file by `f close f p t r` all right. So, here it is opening and here is access and here is closing.

(Refer Slide Time: 30:01)

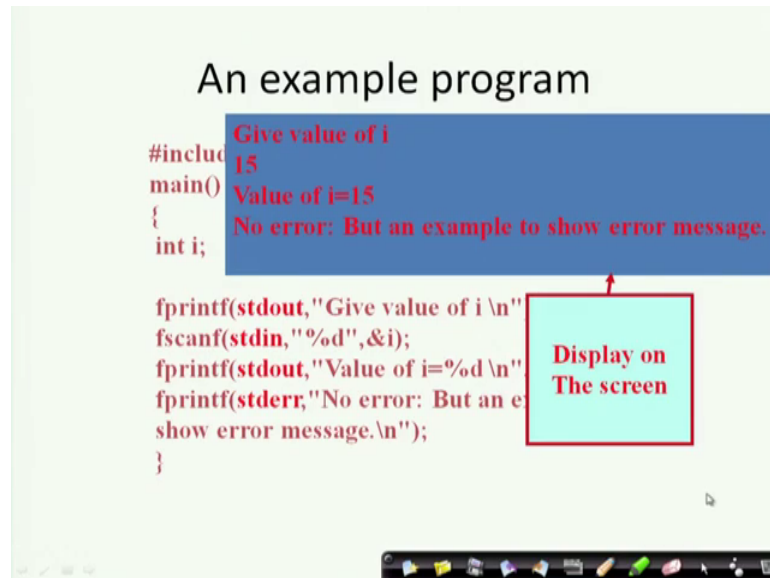
## Three special streams

- Three special file streams are defined in the `<stdio.h>` header
- `stdin` reads input from the keyboard
- `stdout` send output to the screen
- `stderr` prints errors to an error device (usually also the screen)
- What might this do?

```
fprintf (stdout, "Hello World!\n");
```

Ah We have got you have seen that `stdout` in `stdout`, where two special cases of files which are default files and `stderr` was the printing of the error.

(Refer Slide Time: 30:16)



```

An example program

#include <stdio.h>
main()
{
    int i;

    fprintf(stdout, "Give value of i \n");
    fscanf(stdin, "%d", &i);
    fprintf(stdout, "Value of i=%d \n", i);
    fprintf(stderr, "No error: But an example to show error message.\n");
}

```

Give value of i  
15  
Value of i=15  
No error: But an example to show error message.

Display on The screen

So, here is an example program you can see that `main` `fprintf` `stdout` give value of `i`; that means, where am I printing this? Here I am printing it to the standard output I am reading I from the standard input. Now, `fprintf` I am writing that the value of `i` is whatever value of `i` read. So, and there is no error ok. So, give value of `i` it will first `main` give value of `i` you give 15, then `fprintf` that I then it will say value of `i` is 50 is equal to 15 and there are no error, but an example to show error message.


So, if you do this `stderr` then you can if there is an error. So, for example, my I am returned I am being returned a null pointer in that case I can use some output `stderr` and say the file failed to open the file that sort of message.

(Refer Slide Time: 31:31)

## Input File & Output File redirection

- One may redirect the input and output files to other files (other than **stdin** and **stdout**).
- Usage: Suppose the executable file is **a.out**

```
$ ./a.out <in.dat >out.dat
```



The diagram shows the command `./a.out` underlined. An upward-pointing arrow from the label `stdin` below points to the command. A downward-pointing arrow from the command points to the label `stdout` below.

15

So, now another thing I will just talk about here, that will come in very handy to you that is say for example, must have you must be running the programs and you after you compile the program and link them you are creating an executable file, which is a dot out right dot slash a dot out now usually what you do? You have got the dollar those of you using Linux shell a dot out right and return.

Now, this a dot out in that case dot slash a dot out what it is expecting? The input from the keyboard `stdin` and the output is going to `stdout`, but I do not want that I want that I have got a file I have got a file where my input data is there.

(Refer Slide Time: 32:28)

### Input File & Output File redirection

- One may redirect the input and output files to other files (other than **stdin** and **stdout**).
- Usage: Suppose the executable file is **a.out**  

```
$ ./a.out <in.dat >out.dat
```

15

And I call that in dot dat and I have got another file which is known as out dot dat. I want that the input be taken from this.

So, I want a dot out to read the data from here and the result should be written here I can do that in the unix environment very simply by this redirection operation you see, a dot out will run taking data from in dot dat and sending the output data to out dot dat.

(Refer Slide Time: 33:10)

### Input File & Output File redirection

- One may redirect the input and output files to other files (other than **stdin** and **stdout**).
- Usage: Suppose the executable file is **a.out**  

```
$ ./a.out <in.dat >out.dat
```

15

Give value of i  
Value of i=15

So, say for example, in dot dat has got 15. So, I do that and the program runs and says give the value of I, think of the earlier example earlier program that we are thinking of



give the value of i and it means the it reads from here and it prints the value of i is 15. So, that is coming this whole thing is coming in out dot i.

So, let us once again look at this thing here instead of std out dot here I am asking, them to give the value of i is being given and that is being scanned from the input file and this is being written on the output file. So, in the output file both these things are being written; consequently, you see what am I getting is I will be getting this output this is my output out dot dat and in dot dat there are two files.

So, this in this way you can use files for storing data you have to open the file let me summarize a little bit you will have to open the file if you want to read a in the read mode, read the file from there read the data from that file, do the operation, open another file in the write mode and write the data into that file. Thereby, whenever you require some file operations, you can easily do that and this is one example that we have shown which is very common when you during running your programs. If you store some data in a particular file and read from there and write into another file you can utilize this sort of structures this sort of commands.

So, thank you very much; I think you have got an overall idea of how to write c programs and programs and solve problems using programs because our the essence of our course was to solve problems through c programming.

So, you should choose different problems and you should try to write commands sorry I will try to write programs for solving those problems. So, first you have to find out the proper algorithm, and then write the C code for that you have learnt everything about basic things about the C programming, I have not touched upon some special features, which you can also learn from the book like static variables and all those I have left out intentionally.

So, that you are not overloaded, you can solve it, you can the more you run the programs using the basic concepts that has been taught you will be a good programmer and most importantly you will be able to think logically like a programmer you will be able to think of an algorithm, and you will be able to translate that into a program.

Thank you very much.