

**Programming and Data Structures**  
**Prof. N. S. Narayanaswamy**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture - 01**  
**Outline and C Review**

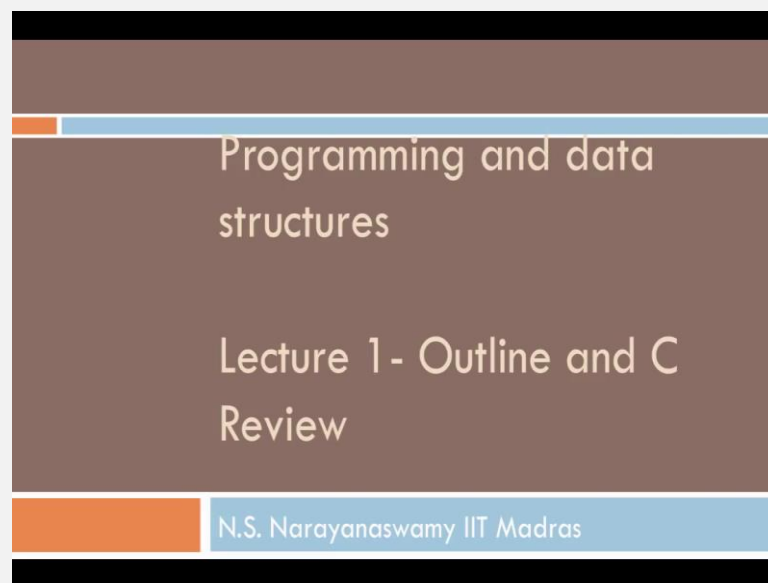
Hello Hi, wish you a happy 2015 and I hope you enjoy this course. The course is a programming data structures course offered by NPTEL and my name is Narayanaswamy and this is the first lecture of the first week of this course. The introduction to this course itself is interesting, we are going to see that this whole recording is made on my machine using four different pieces of software.

The screen of my machine is being recorded by a software called Camtasia, it records my presentation, it records my face, actually it records another the output of another program, which is called photo booth on my computer which is displaying my face here, it is also recording audio. And after some time when I move to the slides, it is going to record the screen with the power point slides unit and then I am going to show you some programs.

So, I am going to open a terminal and open a editor, it is going to record that also. So, observe that this is one big program the Camtasia format, which is actually recording the screen of my machine and this machine is running four different programs and it all looks very complicated and hope fully at the end of this course by seeing this experience, you will be able to appreciate the challenges that a programmer faces. So, periodically I will move between this application that displays my face on the screen therefore, you can see it.

But, most of the time you will be seeing either a set of slides or you will be seeing a program or you will be seeing the execution of the program and I will be talking to you about it, which will definitely be recorded. So, now I am going to move to the slides and I will be making such connectors in between and that should not destructive too much. So, I am just going to go to my slides, thus you can see my whole screen and the applications which are there, my slides are made in power points and I run power point into monitor the full screen, now I am going to play from start.

(Refer Slide Time: 02:29)



So, let us start, so the course as I said before is programming a data structures, this is the first week and this is lecture 1 and we are going to keep this informal. So, you will find that I will maybe started a little bit and so on, but technically the course is going to be complete and very interesting. In other words, I am not going to edit the videos to sanitize it and clean it up, so that everything is very clean and we do not hear ((Refer Time: 03:03)) moves and so on, they are going to be part of this course.

So, this is lecture 1 and the lecture 1 is going to last approximately 20 minutes and we are going to have a short discussion of the outline of the course and also a review of C programming. So, my name as I said before is Narayanaswamy by teacher IIT, Madras.

(Refer Slide Time: 03:27)

## Outline of the Course

- Programming in C
  - Design of Data Structures
  - Instantiating Data Structures
    - Create and Free Data Structures
  - Managing Data Structures
    - Add key, Delete Key, Search Key
- Programming using Data Structures
  - Using example programming exercises
- Minimal/Almost non-existent Asymptotic Analysis

So, here is the outline of the course, the course is going to be the design of data structures, we are going to instantiate data structures, we are going to create and free space and we are going to manage data structures which means you are going to add keys into the data structures, we may delete keys from data structures, we may search an existing key, we may search a data structure for a key to respond with whether the key is present or not.

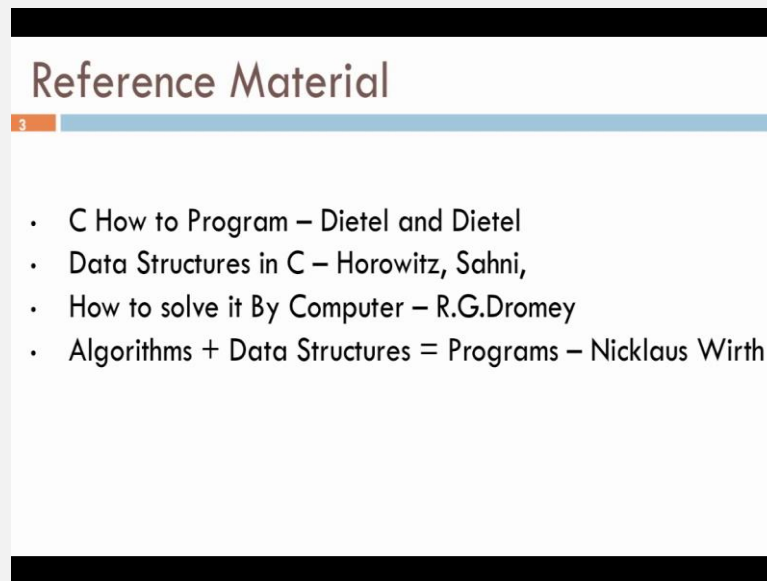
And after describing data structures, we will use example programming exercises to implement these data structures and all the programming will happen in the C programming language. Many of you have first sent emails to me already asking about, whether you can program in java and C plus plus, the answer to these questions is no, this is going to be a C programming course and I expect a basic knowledge of C programming, the ability to write a program, the ability to compile the program, the ability to debug the program and run the program.

And while data structures itself in most B.Tech, BE or Computer Science programs is often a course with the significant theoretical component, which means that there will be a lot of mathematical analysis. In this course, we will deliberately stay away from performing mathematical analysis of any kind. So, in other words there will be minimal and almost nonexistent asymptotic analysis. So, if you are looking forward to learning

asymptotics order notation, O omega, small omega, theta and all that this is not the course for you.

This is a course where you will do a significant amount of programming by implementing appropriate data structures that is the focus of this whole course.

(Refer Slide Time: 05:28)



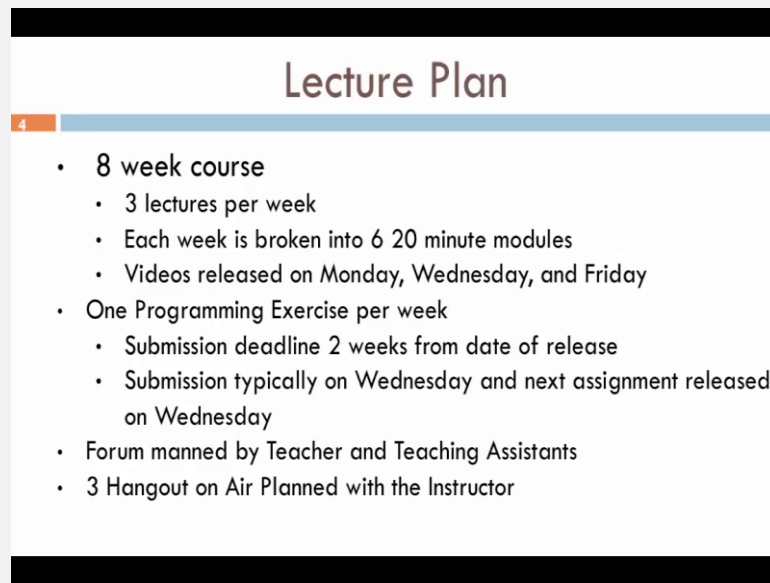
The slide is titled "Reference Material" in a large, dark red font. Below the title is a small orange square with the number "3" inside. The slide contains a bulleted list of four books:

- C How to Program – Dietel and Dietel
- Data Structures in C – Horowitz, Sahni,
- How to solve it By Computer – R.G.Dromey
- Algorithms + Data Structures = Programs – Nicklaus Wirth

So, let us go to the reference material for this course, we will use a combination of these references, as I said these are reference material, these are all text books. So, the book by Dietel and Dietel, C how to program is a good reference for software design methods, the C syntax and it is also very rich in terms of programming exercises, starting from a beginners level right up to an advance programmers level.

Data structures in C is another book, it is written by L. S. Horowitz, and Sardar Sahni and this is definitely a reference material for you to understand the different data structures that we will study. How to solve it by computer by R. G. Dromey is a classic and it talks about how to design programs. The last one by Nicklaus Wirth, the inventor of the Pascal programming language is title algorithm plus data structures equals programs and this is a reference material and I encourage you to take a look at.

(Refer Slide Time: 06:37)

A presentation slide titled "Lecture Plan" with a blue header bar. The slide number "4" is in a small orange box on the left. The content is a bulleted list of course details.

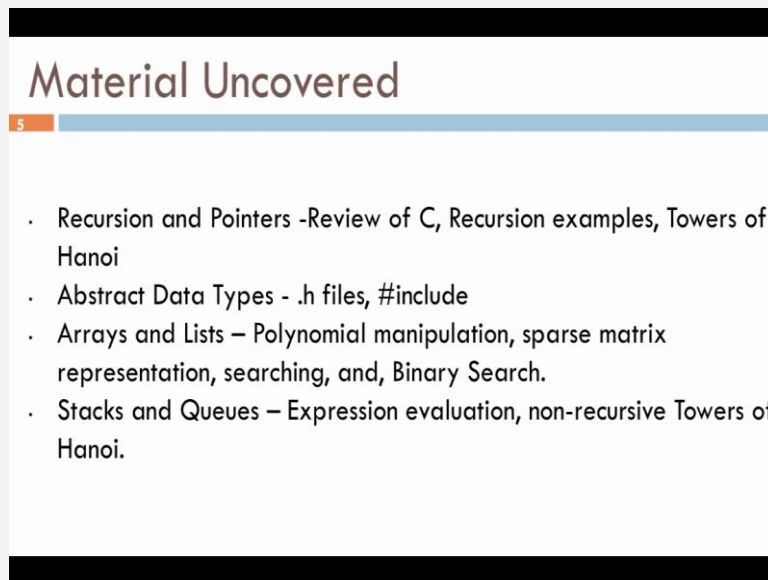
## Lecture Plan

- 8 week course
  - 3 lectures per week
  - Each week is broken into 6 20 minute modules
  - Videos released on Monday, Wednesday, and Friday
- One Programming Exercise per week
  - Submission deadline 2 weeks from date of release
  - Submission typically on Wednesday and next assignment released on Wednesday
- Forum manned by Teacher and Teaching Assistants
- 3 Hangout on Air Planned with the Instructor

So, the lecture plan it is an approximately 8 week course, there are 3 lectures per week. Each week will be broken into approximately 6, 20 minute modules. For example, today we will have two 20 minutes modules, the videos will be released on Monday, Wednesday and Friday at about 6 Am and all these 3 days and every week we will give one programming exercise. Typical submission dead line for the programming exercise will be 2 weeks from the date of release and submission will typically be on a Wednesday morning again, re-test and the next assignment is also released almost at the same time on a Wednesday.

There is a forum on the course website at the NPTEL portal and this is made by me and my teaching assistants and we will respond to your queries which are technical in nature and we will decide which of the questions that you have post that we should answer. We will for example, not post code there, we encourage you not to post code on the forum and there will also be 3 hangouts on air which are planned with the instructor that is made.

(Refer Slide Time: 08:02)



The slide is titled "Material Uncovered" in a large, dark font. Below the title is a small orange square with the number "5". The slide contains a bulleted list of topics:

- Recursion and Pointers -Review of C, Recursion examples, Towers of Hanoi
- Abstract Data Types - .h files, #include
- Arrays and Lists – Polynomial manipulation, sparse matrix representation, searching, and, Binary Search.
- Stacks and Queues – Expression evaluation, non-recursive Towers of Hanoi.

So, what are the material that we are going to uncover, I am not going to cover material, so I am going to uncover material and I am using this phrase which was calling by professor Ananth, the previous director of IIT, Madras. He said that the role of a teacher is to uncover the material to the student and not to cover portions. So, the material that we are going to uncover in this course or to start with, we will talk about recursion and the usage of pointers in the C programming language.

And this is really a review of C and it will be a quick review of C and you should be willing to go out and experiment and learn C fast enough, so that you can do well in this course. We will do examples of recursive programs, recursive functions, a classic example is the towers of the Hanoi problem which we will definitely study. Then, we will also study abstract data types and abstract data types are abstract descriptions of data structures. Many of you have already had questions as to what a data structures, that is a very good question and we will address that by formally defining what a data structure is.

The formal definition of a data structures is captured using the concept of an abstract data type. And this is the definition and as a C programmer, abstract data types are often specified in dot h files and you can also use your abstract data types by using the hash include primitive that comes with the C programming language. Then, specific data structures that we will study arrays and list and these are the starting, these are the basic

data types and we will solve different programming exercises using these basic data structures.

For example, we will write program show multiply polynomials, we will write programs to represent sparse matrices, that is matrixes where predominantly there are many 0s and very few number of 1s. We will then implement algorithms for searching and ordered searching for example, binary search. By order searching, I mean searching an ordered set.

We will then look at other abstract data types which most of you may be familiar with are hold off, for example we will study the two data types stacks and queues. Remember I am not saying data structures, I am calling stacks and queues as data types and we will study the stack and queue data type and we will use them in a programming exercises like expression evaluation and non recursive implementation of recursive functions.

For example, we will write a program which is iterative and nature, it is not a recursive program and it will solve the towers of Hanoi problem. It will maintain a stack and you will see how stacks play a very important role in simulating function calls.

(Refer Slide Time: 11:30)

## Material Uncovered

6

- Non-linear data structures – Tree, Expression tree, tree traversal and generating assembly code
- Search Tree – Insert-Key, Delete-Key, and Efficient find-key by Balancing
- Priority Queue – Heaps maintained in arrays and visualized as trees.
- Dictionary – Prefixes and Suffixes of words, Word Completion

We will then move on to non linear data structures which are examples of which are trees, then special kinds of trees are expression trees and then we will see study algorithms for tree traversal and the application of tree traversal in generating assembly

code. We will then look at search trees and for example, you might have heard a binary search trees and we will look at methods for inserting keys into a binary search tree, deleting keys for a binary search key and efficient find key by maintaining the search tree what is called balanced fashion.

Then, we will look at very advanced non linear data structures which are used in operating systems to maintain priority queues and priority queues are maintained using data types called heaps and we will see how heaps can be implemented on top of the basic data type arrays. And the analysis of heaps is often visualized as an analysis of some special properties on some underline trees. All these details will follow and you need not have to put off, if you do not understand any of these things.

But, this essentially the syllabus of the course, the last item that we will talk about is what is called the dictionary data type. The dictionary data type is a very powerful data type in which you can actually ask not only questions about whether a word is present in the dictionary, but you can also ask for prefixes is a words, suffix is a words and even word completions. So, that would really be the most advance data structure that we will study and use in your programming exercises. So, this is really the syllabus of the course and these we will, we expect to cover an approximately 8 week's time.

(Refer Slide Time: 13:27)

## A Typical Lecture

- The specification of the data structure
  - The Abstract Data Type – data + methods
- Definition of the methods and the data in C
  - In a .h file
- Instantiating objects of the newly created Data Type
  - Writing small wrapper programs
  - Compile and Execute

So, what is a typical lecture look like? A typical lecture looks like today's lecture, you saw me talking about the lecture itself that would last for approximately a minute and

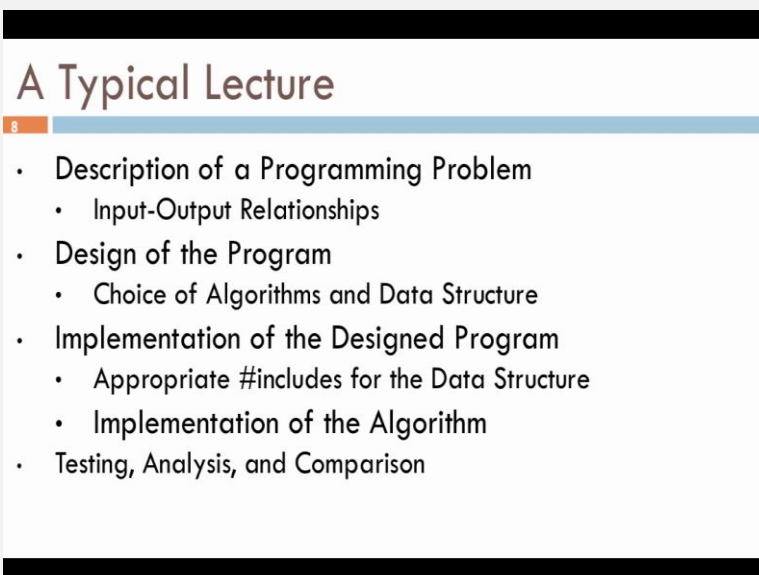


then we will go through above 15 to 20 minutes of a presentation of the lecture material of the day using power point slides. For example, in one lecture I might describe one particular abstract data type that is it is the explicit specification of a data structure.

And what would be the abstract data type actually be, it would say what are the different types of data. For example, you may say that your data type has 5 integers, 5 characters and apart from the abstract data type, we will also maintain or describe the access methods associated with this data structure. For example, we will look at methods which will add keys into the data structure, delete values from the data structure and query the data structure for the presents or absents of a certain value.

Now, the definitions of this method and the description of the data type would be written in the C programming language and they would always be put inside a dot h file, this is a programming practice. So, not only do you learn data structures, but you also learn a bit of programming practice in this course. And then we will write what is called wrapper programs, these wrapper programs will instantiate objects of the newly created data type and you will see what I mean by this and these wrapper programs will be compiled and executed and this is what a typical lecture will consist of.

(Refer Slide Time: 15:19)

A presentation slide titled "A Typical Lecture" with a blue header bar. The slide contains a bulleted list of topics. The slide number "8" is visible in a small orange box on the left side of the header bar.

## A Typical Lecture

- Description of a Programming Problem
  - Input-Output Relationships
- Design of the Program
  - Choice of Algorithms and Data Structure
- Implementation of the Designed Program
  - Appropriate #includes for the Data Structure
  - Implementation of the Algorithm
- Testing, Analysis, and Comparison

Then, after talking about the data structure by describing the data type and then looking at some programs which implement the methods associate with the data type, we will go on to the second part of the lecture and we will describe one programming exercise and

this programming exercise will be explain to you clearly by way of specific Input/Output relationships. After understanding the programming problem we will design the program and when we design the program, we will essentially make a choice of the appropriate algorithms and the appreciate data structures.

Then, we will move on to the implementation of the design program. What is an implementation? We will typically open a file look at a C program, we will analyze the different statements in the program, we will essentially look at an implementation of the algorithm. Then, we compile the C program, then we will test it, analyze it and compare it and the comparison will typically be a comparison between the usage of maybe one or two different data structures for one particular programming exercise.

(Refer Slide Time: 16:37)

## Let us Program

- Let us look at a simple program - sorting
- We now move to a C-file viewed using the vi editor.
  - You can use your own editor to write and save your programs
- Note that the program is well documented
  - What is the program supposed to do?
  - What is the input and output?
  - Who wrote it?
  - What is the role of different statements

So, this is essentially going to the, our typical lecture and now let us move on to a typical lecture and let us go and move to one particular program. And this is one warm up exercise and it also tells you, what the flow of a lecture is going to look like. So, now what are we going to do, we are going to look at very simple program. The goal of a program is to sort and I am going to show you an implementation of a very popular algorithm called the insertion sort algorithm.

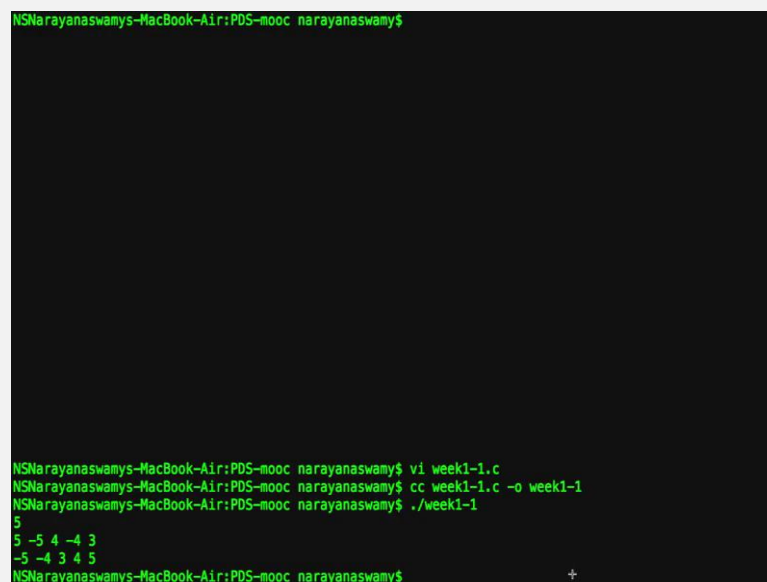
Now, what we do is we move to a C file which is viewed using an editor, now sometimes you may not completely understand what I am doing on the screen, because I am using the vi editor. You may use some other editor when you program on your machine, so

focus only on the C program and do not focus on the commands that I type inside the editor, those are not important at all for you to learn programming and data structures. Every editor that you use has its own features and you may choose your own editor.

Now, the most important thing for you to notice is that when I open this program, you will see that the program is very well documented, it is a programming practice that you must follow. You will see that there will be an initial part, where I will describe what the program is supposed to do, I would describe what the input and output is there will also be some small information about who wrote the program and inside the body of the program, there will be some small very informative comments as to the role of different statements.

So, let us go to our program and I am going to move away from this presentation, we will come back to this presentation in a sort of way ((Refer Time: 18:30)). So, that was the sort of description about what a typical lecture is, what the material to be covered in the course. Like I promise, we will now move to the C program, so we are moving into the second part of this first lecture in this first week.

(Refer Slide Time: 18:59)

A terminal window with a black background and green text. The window title is 'NSNarayanawamys-MacBook-Air:PD5-mooc narayanawamy\$'. The terminal shows the following commands and output:

```
NSNarayanawamys-MacBook-Air:PD5-mooc narayanawamy$ vi week1-1.c
NSNarayanawamys-MacBook-Air:PD5-mooc narayanawamy$ cc week1-1.c -o week1-1
NSNarayanawamys-MacBook-Air:PD5-mooc narayanawamy$ ./week1-1
5
5 -5 4 -4 3
-5 -4 3 4 5
NSNarayanawamys-MacBook-Air:PD5-mooc narayanawamy$
```

So, this is my terminal window and this is the command prompt, I am going to open this program, vi is a editor program please do not worry about it and the program have already created is called week 1, lecture 1 dot c.

(Refer Slide Time: 19:18)

```
#include <stdio.h>

/* This is a Insertion Sort Program written by N.S. Narayanaswamy.
Input is assumed to be in two lines: first line is a positive integer value not larger than a 1000.
In the next line is a space separated sequence of at most 1000 integers, terminated by an end-of-line
character
(otherwise called return).
The output is the input sequence printed out in an ascending order in a single line, with consecutive
elements separated by
EXACTLY one space.
*/

void insertion_sort(int *, int);
/* function prototype declaration, for the compiler. Conceptually not necessary in this example. As the
function is
defined in this file. But very useful in programs where a function definition may be in another file.
*/

int main()
{
    int number_of_keys;
    int values[1000];
    int i;
}

"week1-1.c" 82L, 1876C
```

So, this is the C program and this is my editor and if you know your C program, we should not be confused or we should not have a question as to what the first sentence here in this program is the hash include stdio dot h. Now, you can see that I have a long C command from starting from this place on words, this completely describes what the program does. For example, it says the program is written by me N. S. Narayanaswamy, it also says something about what the input to the programmers.

We will assume the input is 2 lines, the first line is a positive integer value, not larger than a 1000. So, this means that when your program runs you will have to give input, on the first line and this value should not be larger than a 1000 of course, you can ask what happens if you give a value larger than 1000, the behavior of the program is unpredictable. In the next line, the input consists of a space separated sequence of the number of integers that you have said will be given.

And at the end of this sequence of integers, you type up end of line character; otherwise, which is called you press a return key. Now, the output of the program is described in this line, the output is the input sequence printed out in ascending order in a single line and the consecutive elements are separated by exactly one space, that observe the written exactly in capital letters. You should keep in mind that when you submit your programming exercises, you will have to follow such important rules to ensure that the

evaluation system does not get confuse by your the output of a program, which it is exactly once spaces extremely important this command ends.

So, now we are into the C programming part of it, this sentence, this statement is a function prototype. And if you know the C programming or even if you do not know this is the new concept, it is a good programming practice to declare the functions by giving the name of the function, the return type and the argument type at the beginning of the program. I repeat this such a good programming practice, sometimes it is very necessary, but initially it is a very good programming practice to follow this habit of declaring your functions.

In this case observe that the declaration consist of only the prototype of the function, the name of the functions is insertion are, there is no return value that is why there return void there. And there are two arguments in this function, one is a pointer to integer that is int star, the other one is an integer argument. Note that, there are there are no variable names here, this is the difference between a declaration and function definition, we will see a function definition shortly.

Now, a declaration is very important, it is very important for the compiler. In this example it is not conceptually necessary, but it is very useful in many programs, when you become an export programmer that the function definition maybe in another file node. The function definition maybe in another file as suppose to the function declaration which is here. So, here is a function definition and here is a function declaration, now let us go to the main function of your program.

(Refer Slide Time: 23:33)

```
int main()
{
    int number_of_keys;

    int values[1000];

    int i;

    scanf("%d",&number_of_keys);
    /* reading the number of keys */

    for (i = 0; i <= number_of_keys-1; i++)
        {scanf("%d",&values[i]);
        }

    /* The keys have been read into values array. What does one do if user gives data which is smaller th
    an
    number_of_keys? Future work */

    insertion_sort(values, number_of_keys);

    /* values is now sorted */

    for (i = 0; i <= number_of_keys-1; i++)
        {printf("%d ",values[i]);
        }

    printf("\n");

    return 0;
}
```

So, the main function of the program is called main, it returns an integer and we will see how to use these return values very carefully, as we go through this holes. But, really that is not the focus of the course, it is that some that human being interested in as an program, but we will definitely visited, but not spend too much time up. So, since this is simple program it is a review of the C programming language, I have an integer which is called a number of keys and I have an integer array of 1000 which can contain 1000 values which called values, the array is called values and I have and integer variable I which I a use an index.

Now, note that I read the first data item which is an integer. So, here I am reading in the number of keys that are going to be stored in the array. Here is a loop, which then reach the values one after the other, note the usage of ampersand and the values of i. And notice that, this for loop reach starting from location 0, starting from the 0th element in the input to the element whose index is number of keys minus 1, that is a total of number of keys, number of input elements this is the standard for loop, that you be must a familiar with.

And as you do this I am sure some of you will ask questions like, what happens if the user gives data which is the smaller than the number of keys, what if the number of values which are given in the input is more than the value of number of keys and so on, that is a remark that I have put in saying that this is consider to be future work. And now

we make a function call to the function insertion sort, sending it values that is observe that this is a pointer to the first elements of the array.

And we are also sending the number of keys, which are the relevant values that need to be sorted in the array. Now, the way the program is decide is that when the execution returns from this function call, at this point the values array would be now sorted and in this for loop, we print the values one after the other with a space between the two of them.

Finally, we terminate the printing by giving a carriage return or a new line and the execution ends by retuning the value 0. I hope that was clear that was the execution of the main program, now we are going to look at the design of the function call insertion sort.

(Refer Slide Time: 26:55)

```
return 0;
}

void insertion_sort(int *ptr_2_values, int size)
/* ptr_2_values contains the address of values[0]. It has a copy of values, in other words */
{
    int i,j;
    int temp;

    for (i=0; i <= size; i++)
    {
        /* insert the i-th element into the correct place in [1..i] */
        for(j=i-1; j>=0; j--)
        {
            if (ptr_2_values[j+1] < ptr_2_values[j])
            {
                temp = ptr_2_values[j+1];
                ptr_2_values[j+1] = ptr_2_values[j];
                ptr_2_values[j] = temp;
            }
            else
            /* i-th element has found the correct place. Go to the next element */
            break;
            /* break from this iteration */
        }
    }
    /* The array pointed to by ptr_2_values is now sorted */
}
```

So, the function is called insertion sort and it does not return any value that is why you see the void here. Now, observe that the data type of the first argument is a pointer to integer and you will already recognize that the variable pointer to values contains the address of values of 0. Whenever this function called is made, when control enters this function ptr underscore 2 underscore values contains the address of values of 0.

Further the variable size which is a local variable, local to this function contains the size of or the number of relevant elements in the values array that need to be sort. Here at to

local variables from the C programming syntax, you will ensure not get confuse by the reuse of the variable name  $i$ . Because, when the execution enter this function, the meaning of  $i$  of this word, this variable name  $i$  is completely local to this function. Then, I have the other variable call  $j$  and I have a temporary variable, which is used to exchange the values in two locations in the array.

So, what is the insertion sort algorithm look like, observe that there is a first for loop, the role of this particular for loop. And let me just highlight this region the role of this for loop is return in this remark in the  $i$ th iteration, the  $i$ th element is inserted into the correct place in the range 1 to  $i$ , that is when the control enters the  $i$ th iteration, the goal is to take the  $i$ th element in the array.

What is the array called now? The array is called pointer underscore 2 pointer to values, which deliberately called pointer to values at as the control enters this loop in the  $i$ th iteration, the goal is to insert the  $i$ th element of pointers to values, the array called pointer to values in the correct location inside the elements 1 to  $i$  that is our goal. So, let us imagine a particular iteration number  $i$  and let us understand the next for loop, the role of this for loop is to insert the  $i$ th element into the correct place, let us see how to do it.

The way this is done is to move backwards from the  $i$  minus 1 with the element up to the 0th element, that is the role of the index  $j$ . The role of the index  $j$  is to run from the index  $i$  minus 1 in the array to the index 0 and in a sense it is running backward, it starts from  $i$  minus 1 and it goes up to 0. And what is this do, it checks in each loop it checks if the  $j$  plus 1 with the element, that is in the  $j$ th iteration, it checks if the next element, that is the  $j$  plus 1 with element is smaller than the  $j$ th element in the array.

If it is indeed smaller than the  $j$ th element in the array, then these three lines are very famous lines, which every programming student would recognize, this exchanges the values in the locations  $i$  plus  $j$  plus 1. These three statements exchanges the values in the locations  $j$  plus 1 with the value in the location  $j$ . How does this happen, temp stores a copy of the value in the location  $j$  plus 1, then the value in the location  $j$  is copied into the value into the location  $j$  plus 1, then the value in the location  $j$  is made to be the value which is stored in the temporary location.

When are these three locations, when are these three statement executed, they are executed when there is a wrong order between the  $j$  plus 1 of the element array and  $j$  plus



1 with element in the array and a jth element in the array. In other words, when the j plus 1 with the element is smaller than the jth element in the array, the exchanges if it. Otherwise, in other words if the j plus 1 at the element is at least as large as the jth elements, then we observe that the ith element as found the current as found it is correct place and we break from this iteration using the C programming statement, which is break.

Let us recall what break does, breaks from the enclosing loop which is this for loop that is a control after executing this break, exits from this for loop and comes to this particular location, which means it will move on to the next element of the array that we are sorting here. Eventually I will touch size and then it will getting increase to size plus 1 and then, the control will exit from the loop. And as you can see by just thinking about this little bit, after every iteration a certain prefix of elements of an array are sorted.

In other words, after the ith iteration the first i elements, that is the elements index by 0 to i are sorted, let me make this small change. So, that you can notices this I have written in the correct place in 1 to i it is actually 0 to i. Therefore, after the ith iteration, the first i plus 1 element are sorted and eventually the value of i will become equal to size and eventually it will cross, the value of size at which time the control will exit and return from this function.

You will also notices that I have missed the returns statement it does not matter, but again I missed it. So, that I can show you it is a good programming practice to put in some of these commands, put in the appropriate statements. So, once control reaches here it goes to the place where the function was called from which in this programming exercise is here. And then, we put in the remarks saying that values the array is now sorted, some of you may wonder how does value is get sorted, while the insertion sort was executed in this array.

Now, this is the beauty of passing addresses, which is a very powerful feature of the C programming language, observe that ptr of values points to the parent array. If I can use that phrase values, which actually contains the values that you wants to sort and this is the very useful feature, but also very dangerous feature. If you make some mistakes here, in this part of the code in some other programming exercise for example, if you delete values, then the values might be completely lost.

So, it is very important for you to remember to use pointers very carefully when you pass them from one function to another function.

(Refer Slide Time: 35:24)

```
#include <stdio.h>

/* This is a Insertion Sort Program written by N.S. Narayanaswamy.
Input is assumed to be in two lines: first line is a positive integer value not larger than a 1000.
In the next line is a space separated sequence of at most 1000 integers, terminated by an end-of-line
character
(otherwise called return).

The output is the input sequence printed out in an ascending order in a single line, with consecutive
elements separated by
EXACTLY one space.

*/

void insertion_sort(int *, int);
/* function prototype declaration, for the compiler. Conceptually not necessary in this example. As the
function is
defined in this file. But very useful in programs where a function definition may be in another file.
*/

int main()
{
    int number_of_keys;

    int values[1000];

    int i;

:wq
```

So, this was the whole program and now I am going to compile this program for you. So, that you can see it do not worry about the wq in the vi editor, it means write and quit.

(Refer Slide Time: 35:36)

```
NSNarayanaswamys-MacBook-Air:PDS-mooc narayanaswamy$ vi week1-1.c
NSNarayanaswamys-MacBook-Air:PDS-mooc narayanaswamy$ cc week1-1.c -o week1-1
NSNarayanaswamys-MacBook-Air:PDS-mooc narayanaswamy$ ./week1-1
5
5 -5 4 -4 3
-5 -4 3 4 5
NSNarayanaswamys-MacBook-Air:PDS-mooc narayanaswamy$ vi week1-1.c
NSNarayanaswamys-MacBook-Air:PDS-mooc narayanaswamy$ cc week1-1.c -o week1-1
NSNarayanaswamys-MacBook-Air:PDS-mooc narayanaswamy$ ./week1-1
6
6 5 4 3 2 1
1 2 3 4 5 6
NSNarayanaswamys-MacBook-Air:PDS-mooc narayanaswamy$ ./week1-1
6
-6 -9 1 0 3 4 7
-9 -6 0 1 3 4
NSNarayanaswamys-MacBook-Air:PDS-mooc narayanaswamy$ ./week1-1
6
-9 -6 0 1 3 4 -10
-9 -6 0 1 3 4
NSNarayanaswamys-MacBook-Air:PDS-mooc narayanaswamy$ ./week1-1
6
-9.0 1.1 -2 3 4 5
-9 32767 32767 32767 1771167728 1771171232
NSNarayanaswamys-MacBook-Air:PDS-mooc narayanaswamy$
```

Now, I am back to the command prompt and I am going to compile my C program. So, let me compile cc is a C compiler and I am going to compile the program that we just wrote. And I am going to send the output to this executable, which is week 1 dot 1. The

compilation is successfully completed and let us look at the kind of mistakes that we can make while given inputs and these are the kind of mistake that you are expected to program for when you start writing very sophisticated programs.

So, let us execute the program that we have just return, in other words we have complied it and created and executable, let us execute this program. Now, the executable is ready, now I have deliberately design the program. So, that there is no message to you, because when you submit your programming exercise, there will be no search interaction between your program and the program evaluation environment.

For example, it is useless for you or actually it is dangerous for you to give messages like, hello please print your input, such kind of messages are not to be given when you submit your programming exercises, just the values need to be given. So, let us give the value that we say 6 remember that 6 stands for the total number of keys that we want to sort. Now, the system waits for the 6 keys that need to be given, let us give 6 5 4 3 2 1 and you see the output is sorted an ascending order, because of our insertion sort algorithm.

I have let us run this on an input that consist of negative numbers and you will see that, but also works perfectly find minus 6 minus 9 1 0 3 4. And just for the sum of it let me introduce 7 which is one more key then what we have set we will give. And as you can see, the last element is last even more interestingly, let us give make the same mistake let us make a minus 9 minus 6 0 1 3 4 and minus 10 minus 10 is last.

Now, let us important for you to analyze why, it is not very hard, but it is important for you to analyze. Now, let us see what are all the mistake that we can make, when we give inputs and these are the kind of mistakes that you must be careful to avoid by reading the programming assignment statement very careful. Again let us give 6 values, let me give minus 9.0 1.1 minus 2 3 4 5 let us have finish giving 6 now observe that the output is highly likely to be some very unpredictable of this is a problematic situation.

So, you must be very careful when you give inputs true such programs, remember the programming exercise was to sort a given set of integers in not numbers. So, this is an extremely important think, similarly let us make another input, let us give lesser number of inputs then what was counts. So, let us give 6, let us give 2 4 1 3 and 4 and 3 and

more, more, more and observe that the program is waiting for numeric inputs from you and it is skipping all the space.

So, now, that is minus 7 now the whole input is been given and interestingly it is solve. So, this also tells you something mysterious about scanf, we will not talk about it, but I encourage you to explore how scanf works.

(Refer Slide Time: 40:47)

```
The output is the input sequence printed out in an ascending order in a single line, with consecutive
elements separated by
EXACTLY one space.

*/

void insertion_sort(int *, int);
/* function prototype declaration, for the compiler. Conceptually not necessary in this example. As th
e function is
defined in this file. But very useful in programs where a function definition may be in another file.
*/

int main()
{
    int number_of_keys;

    int values[1000];

    int i;

    scanf("%d", &number_of_keys);
    /* reading the number of keys */

    for (i = 0; i <= number_of_keys-1; i++)
    {
        scanf("%d", &values[i]);
    }

    /* The keys have been read into values array. What does one do if user gives data which is smaller th
an
number_of_keys? Future work */
```

Now, let us go to our program again and make some special kinds of errors just to understands and just to revisit the C programming language. For example, the standard mistake programmers make is in this scanf they forget the ampersand. Now, let us see what is going to happen by compiling our program

(Refer Slide Time: 41:00)

```
--9 -6 0 1 3 4
NSNarayanasmwmys-MacBook-Air:PDS-mooc narayanasmwamy$ ./week1-1
6
--9,0 1.1 -2 3 4 5
--9 32767 32767 32767 1771167728 1771171232
NSNarayanasmwmys-MacBook-Air:PDS-mooc narayanasmwamy$ ./week1-1
6
2 4 1 3

4 -7
-7 1 2 3 4 4
NSNarayanasmwmys-MacBook-Air:PDS-mooc narayanasmwamy$ vi week1-1.c
NSNarayanasmwmys-MacBook-Air:PDS-mooc narayanasmwamy$ !cc
cc week1-1.c -o week1-1
week1-1.c:20:12: warning: format specifies type 'int *' but the argument has type 'int' [-Wformat]
scanf("%d", number_of_keys);
      ~~~~~
1 warning generated.
NSNarayanasmwmys-MacBook-Air:PDS-mooc narayanasmwamy$
```

Now, this is very important observe that the compiler is given only a warning it does not say that there is a error it only generates a warning, which means a compilation successfully when through. So, let us run our program and see what happens and able to see that miss in the ampersand as of fatal on sequence at execution, as you can see the execution, terminated with the segments fault and this is a fatal execution. Because, the execution of the program was dominated and you have a error that we going to talk about, but again you can go out and study what segmentation fault actually means.

(Refer Slide Time: 41:57)

```
#include <stdio.h>

/* This is a Insertion Sort Program written by N.S. Narayanasmwamy.

Input is assumed to be in two lines: first line is a positive integer value not larger than a 1000.

In the next line is a space separated sequence of at most 1000 integers, terminated by an end-of-line
character

(otherwise called return).

The output is the input sequence printed out in an ascending order in a single line, with consecutive
elements separated by
EXACTLY one space.
*/

void insertion_sort(int *, int);
/* function prototype declaration, for the compiler. Conceptually not necessary in this example. As th
e function is
defined in this file. But very useful in programs where a function definition may be in another file.
*/

int main()
{
    int number_of_keys;

    int values[1000];

    int i;

"week1-1.c" 83L, 1083C
```

So, let us go back and correct that error the role of the ampersand is very crucial and you must understand what the ampersand actually means. Let us not very hard of few practice sessions with scanf will clarify the role of the ampersand, observe here that I said ampersand values of i. So, observe that in English you will read it as the address of values of i that would get rid of much of a confusion.

So, I hope you look at the program carefully and look at the execution and different kinds of errors that we can make... Also if you notices we have pass the values that is the address of values of 0 by passing a pointers to the function, which does the insertion sort. So, therefore, this is a simple program, but those of you who are using pointers for the first time. So, you should get brave and go out of study pointers use that.

Because, it is not possible to maintain data structures without being comfortable with manipulating pointers in the C programming language. So, that finishes in some sense our hands on session and let us get back to our slides. So, I hope you enjoy this session of looking at a program which was written by me and you can write similar program I encourage you to write similar programs. But, every programming exercise, every data structure, every algorithm that you study write a small piece of code make it work, make it work and toy and puts and you become a stronger programmer.

And I am motivation to showing you that is to show you that programming is not hard, it is an exercise for to be logical to follow or reject syntax do not forget details like the role of the ampersand and so on and so forth to pass pointers carefully to modularize your code, that is right appropriate functions to make your program very readable, put in appropriate commands, programming likes poetry it is very enjoyable task. So, let us get back to the slide to review what more needs to be done before we end today's lecture ((Refer Time: 44:47)).

So, this was a let us program slide where we went to the terminal, we opened our program and then made changes to it, studied the code compiled it and executed it and understood it is execution.

(Refer Slide Time: 44:59)

## What kind of errors

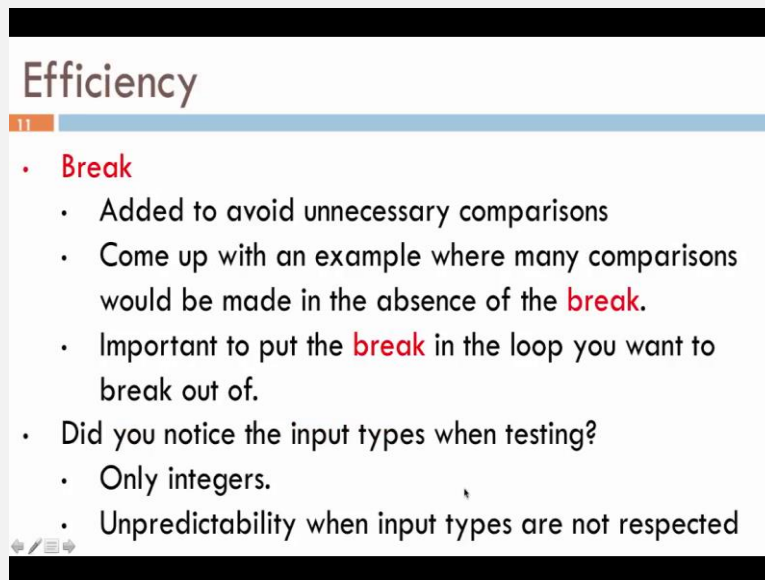
10

- Do not miss the & in the scanf, and apply it to the correct data item.
- With the format specifier associated with scanf, do not put additional characters (explore scanf)
- Did you note the value passed to insertion\_sort
  - It is an address.
  - **value** contains **&value[0]**
  - Did you notice the **ptr\_2\_value[i]**?

Very importantly what are the different kinds of errors we made, we should not miss the ampersand this scanf and very importantly you must apply it to the correct data item. And there was another very important thing, this was an error that we did not create we will definitely create after going through these slides. And when you use scanf it is very important for you, along with the format specifier not to add additional characters, we will go back and see what I mean by this, I hope you also note the value that was passed to insertion sort, it was an address.

In other words, the value that is read here, the memory location value consists of the address of value of 0 and did you also notice that I access pointer to value of i that is I access the i<sup>th</sup> element in the array, whose name is pointed to value. I hope you notice it unknown tell you the reason for these things you suppose to become familiar with these.

(Refer Slide Time: 46:22)



## Efficiency

- **Break**
  - Added to avoid unnecessary comparisons
  - Come up with an example where many comparisons would be made in the absence of the **break**.
  - Important to put the **break** in the loop you want to break out of.
- Did you notice the input types when testing?
  - Only integers.
  - Unpredictability when input types are not respected

And other very important think that we did is the also where worried about the efficiency that is why we added the break statement, with the break statement remember that we avoided unnecessary comparisons. Now, here is a small exercise come up with an example where many comparisons would be made, if we did not put the break. Observe that the break is not important for the correctness of the program, even without the break statement there, the program would have been correct.

Now, it is very important for you to put the break statement at the correct place, it is very important for you to put the break statement at the correct place, you must put it in the loop that you want to break out of. Also did you notice the input types when we tested the program, only integers and you also notice that the output is highly unpredictable when the input types are not respected. What are the outputs, those are not very important, we should not really care about what the output is when the input is wrong.



(Refer Slide Time: 47:37)

## Additional Reading

12

- Review your C programming from your favourite book
- Write as many programs as you can
- Next Lecture
  - Pointers
  - Structures
  - Function calls and return types.
- Check for it early day after tomorrow morning

So, what is the additional reading that you must do before you get to the next lecture, review your C programming from your favorite book. In particular, please become familiar with pointers, become familiar with the syntax or scanf, how to pass values to pointers, what are return types and so on and so forth. Write as many simple programs as you can and come back to the next lecture, we will study pointer, we will study structures, we will make function calls and we will also return values from functions and that video will be released day after tomorrow morning.

Thank you very much for your attention and enjoy yourself, I am going to stop the lecture now.