

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

**Lecture - 09**  
**Introducing Functions**

So, we were discussing a little more complicated a little more different program that is using a function.

(Refer Slide Time: 00:19)

**Sample C program #4**

Preprocessor statement.  
Replace PI by 3.1415926  
before compilation.

```
#include <stdio.h>
#define PI 3.1415926

/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);

    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f\n", area);
}
```

Example of a function  
Called as per need from  
Main programme.

```
float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```

*# define resistance 10.2*

The slide includes a small video inset of the professor in the bottom right corner.

Here, what we had discussed in the last lecture is, that we are using a new type of statement that is hash define this is a preprocessor statement; this is called a preprocessor statement which is replacing PI with the value that is specified.

Now, anywhere I can do I can define other things like this also for example, I can write see hash define resistance is 10 ohm; 10.2 ohm whatever. That means, in my program wherever I get this variable name resistance, that will be replaced by this constant value 10.2 even before the program is compiled; that is the first new thing that we saw.

(Refer Slide Time: 01:35)

### Sample C program #4

Preprocessor statement.  
Replace PI by 3.1415926  
before compilation.

```
#include <stdio.h>
#define PI 3.1415926

/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);

    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f \n", area);
}
```

Example of a function  
Called as per need from  
Main programme.

```
float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```

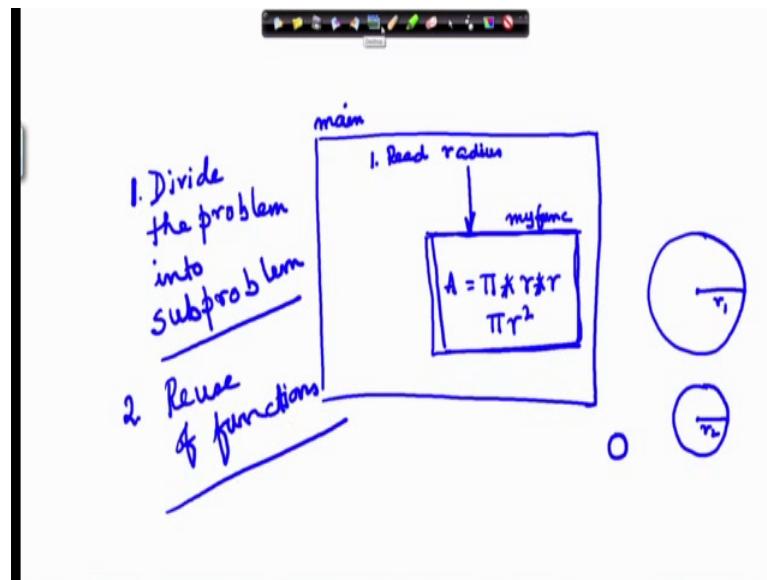
Function called. 15.25 radius

The second thing is look at this, let us study this program. First time putting an a comment and the comment tells me what this program is going to do. The comment is compute the area of a circle, and as usual I start with a main sorry as usual I start with a main an inside main let us see what we have done, we have done something different.

Now all of you will be able to say what is this. This is a type declaration for two variables radius and area. At what sort of type declaration is that it is float; that means, radius and area are two variables which will hold real numbers or floating point numbers right. Now I also declare myfunc this is another function, and the style in which I wrote tells a compiler that this is a function and this function is also of type float.

What is the type of a function when we will be studying function in detail, we will be understanding there. Now what have we done here? Scan f you know that by now scan f is reading the value of some variable. How many here I look at this point? I find that it is 1 percentage f. So, only one variable is being read and what is that variable that is that is radius alright. So, I will read the value of the radius and radius is of type float right. So, it will be say 15.25 that is being read now computing the area the main function is not doing itself. So, let us quickly go back to our old diagram.

(Refer Slide Time: 04:09)



So, this is the main machine of the main function. So, here what have you done? We have read I am writing in a pseudo code I have read the value of radius and there is a function called myfunc; my function. Now after I read the radius I do not want to take the botheration of computing the area, this machine submachine knows how to compute the area it knows that area will be computed as pi times r times r; that means, pi r square right area of a circle.

But this is a function that can compute area of any circle it has been design, because by pi r square you are not finding the area of only this circle, you can find the area of the circle or might be this circle only thing that is differing is the radius right; here it is one radius, here there is another radius, here is another radius. So, this machine or this function can compute any (Refer Time: 05:51) any area of any circle provided it gets the radius told to him told to this.

So, this machine is expecting some value to be passed on to this. Please note the terms I am using. This function once some value required some value to be passed on to this. Now you can ask that well be such a simple program, I could have written it here itself yes certainly you could have written it here itself, but this is just an example. Actually in a very complicated scenario, there are many complicated tasks and if we want to solve the entire task by yourself by through single program, there is a chance of error that is problem number one.

So, we want to divide the problem into sub problems; that is a general programming philosophy that we want to divide the problem into sub problems and solve each of the sub problems separately because they are more manageable, I can find out what are the errors whether they are working properly or not and then I combine them together and solve the overall problem that is advantage number 1.

Advantage number 2 is that suppose let us just for the sake of argument, assume that my function this finding the area is really a complicated program very complicated program. So, somebody has really taken the effort to develop this complicated program. Now whenever me or you or anybody any programmer wants to use this facility, then they do not need to reinvent the wheel and write that complicated program once again it is available and I can reuse it, and only thing that I need to do while using it is just to pass the parameter. Therefore, the second big advantage is reuse of functions that is why the concept of function is so important in programming alright. You will encounter this irrespective of programming languages. So, let us go back to this.

(Refer Slide Time: 08:41)

### Sample C program #4

Preprocessor statement.  
Replace PI by 3.1415926  
before compilation.

```
#include <stdio.h>
#define PI 3.1415926

/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);

    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f \n", area);
}
```

Example of a function  
Called as per need from  
Main programme.

```
float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```

Function called.

*main*  
*myfunc*  
*radius*

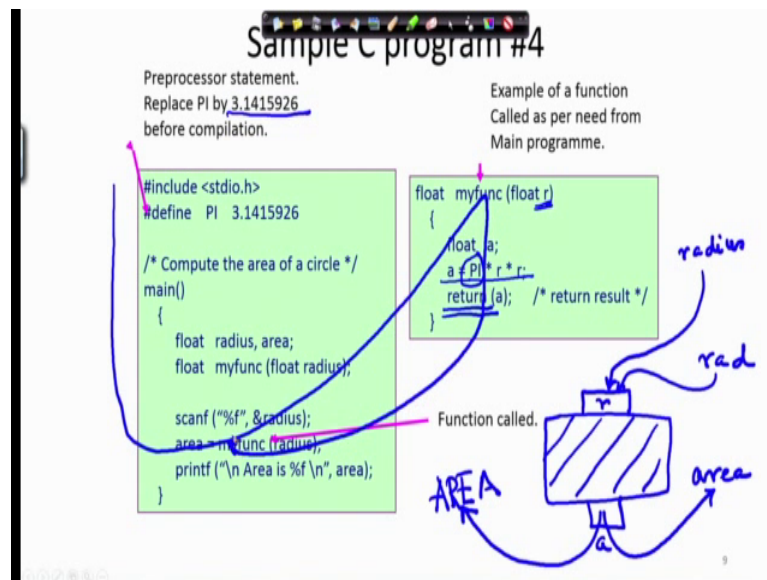
So, here you see I have I am not computing the area, I means the main is not computing the area what the main is doing? It is just here it is calling my function.

So, immediately this one is being called, and what will my function do? My function is ready my function is here and it is just expecting something which is r to come in because it will compute pi r square. This expecting r and this one has got the variable that

radius in the variable radius. So, this is being passed on to this. So, over here from the main function here is the main, from the main this radius is being connected to this r alright and this function my function is reading is getting the value of r, and inside that it is computing the area.

Now, there are some details which I will mention later, it is computing the area and it is finding the value in its own variable a. A is a variable now for any program, if this function is used then instead of radius you may have my radius or any other value any other sorry any other variable name can be there. Now any this will work for all of them let me clarify it once again.

(Refer Slide Time: 10:30)



See here there is a function and this this function accepts r and produces a.

Now from, as I said that it can be reused from my program, I am using it with the variable radius it is been connected here. Maybe from your program you have not used the variable name radius, you have used the variable name say rad so that will that when your program is using this function, this rad will be connected to r. Now suppose here in my program I have used the variable name area for noting the area. So, this output a will connect to area for example, for your program you said area.

Now, recall that in c small and capital makes difference to this is another variable name is a different variable. So, for you this a will connect over here, now this person this my

function is only concerned with r and a and the job of connecting them is up to the function that is using it. So, here what happened I called the main function my function computed a how here I wrote PI r r know how do I know PI; because pi has already been defined here. So, this pi will be replaced by this value. So, 3.1415926 times r times r and what is this r? Since it has been call when it has been called with radius the value of radius will come here and return a. Now this return term means where will it return, this is another new word that you are coming across. This return means it is returning to the point from where you are called. The programmers going on like this from here it called my function.

So, after my function is executed, it will return back to the same point.

(Refer Slide Time: 12:59)

The slide is titled "Sample C program #4" and is divided into two main sections. The left section, titled "Preprocessor statement. Replace PI by 3.1415926 before compilation.", contains the following code:

```
#include <stdio.h>
#define PI 3.1415926

/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);

    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f \n", area);
}
```

The right section, titled "Example of a function Called as per need from Main programme.", contains the following code:

```
float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```

Handwritten annotations include a pink arrow pointing from the function definition to the call in the main function, and the text "Function called." next to the call. A blue circle highlights the function call, and a blue arrow points from it to the text "Area is 255.72" written in blue ink.

So, we have come across a new word return and return is well returning the value a, and that is being assigned to my variable area all right and then print f you can now understand it will what will be printed? Area is dash and this one will take some floating point number maybe 255.72 whatever it is and this value of area will be filling up this place.

So, this is another example of a C program. So, through these examples, what we tried to do is to introduce you to some of the very common names, common words and common features of the C programming language.

(Refer Slide Time: 14:05)

main() is also a function

```
#include <stdio.h>
main()
{
    int a, b, c;

    a = 10;
    b = 20;
    c = a + b;
    printf ("\n The sum of %d and %d is %d\n",
           a,b,c);
}
```

Next we move to this that main is also a function I have already said that that main is also a function this is clear to you now, but just quickly have a look at this. Here I have identified some variables, have identified some variables a b c which are integers and the same old program and is also a function fine.

Now, here are some words of advice, when we are writing a program it has got a couple of things we have to keep in mind, first it must be correct so that it can be executed by a compiler. It should not it should adhere to the rules of the game the syntax in the grammar of the C language. The other thing is also this program has to be understood by others, because if they you have written program and someone else wants to extend this program. You are working in a big company and there are number of programmers and each of them are writing a small segment of the program and everybody must be able to understand everybody's program.

Therefore the programs should be written in such a way so that it is understandable more better understandable. Just like if we write in very bad hand writing write something here, something there, something there I not understand it, but if you write it nicely in a sequence it is much with proper paragraphs etcetera it is much better readable and better understandable. So, here we will see some of the requirements for desirable programming style one is of course, clarity.

(Refer Slide Time: 15:55)

**Desirable Programming Style**

- Clarity
  - The program should be clearly written.
  - It should be easy to follow the program logic.
- Meaningful variable names
  - Make variable/constant names meaningful to enhance program clarity.

Handwritten examples:

- $sum = a + b$
- $sum = num\ 1 + num\ 2$
- $pqr = (x+y+z)/3$
- $avg = (x+y+z)/3 ;$
- $Average = -$
- $(t*t) = t * t - x$

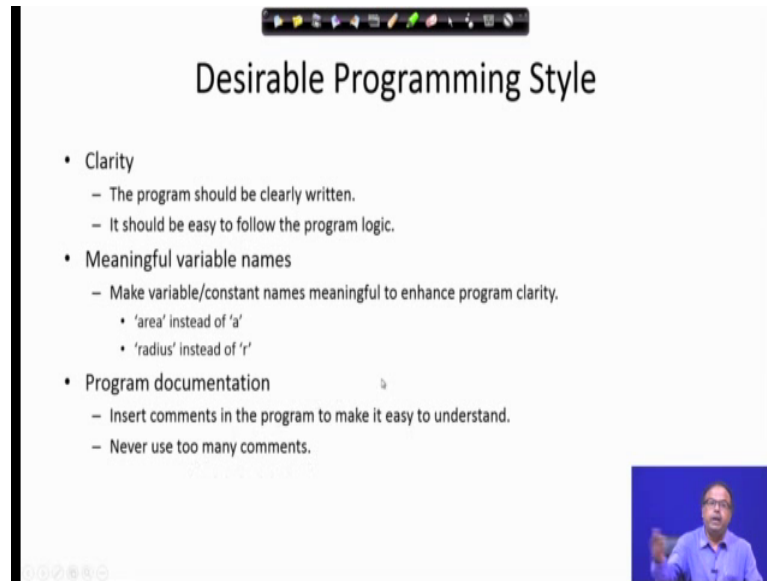
It must be very clear; the program should be clearly written it should be easy to follow the program logic. Now here is something that is very important I insist on that there should be meaningful variable names. For example, I want to add two numbers and when I say sum assigned a plus b or sum assigned num 1 plus num 2 it is good semi colon please do not forget the semi colon and listen.

Now, the sum is a meaningful variable name, it immediately tells me what this variable is meant for it is holding the sum. But suppose if I had written something like t t is equal to t t t multiplied by t multi minus x, then from there its not very clear what is this t t why are you using this t t. If say for example, average I am computing average as avg and that is x plus y plus z divided by 3 is very understanding semi colon it is understandable, I could have written the full thing average equals to this or I could have written avg all these things are fine.

But if I had written p q r is equal to x plus y plus z divided by 3, then its not very clear although I can look at this side and understand that therefore, it is p q r is the average, but this not a very good practice. Now so, make variable and constant names meaningful for example, pi should be written as p i is not good that 3, 4 3.1415 all those that value I (Refer Time: 18:21) chi because that is not very clear what this value standing for.



(Refer Slide Time: 18:28)



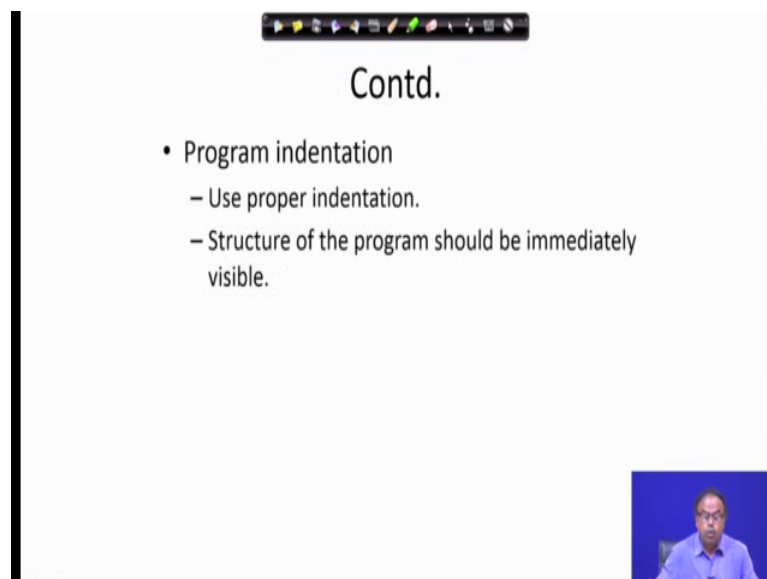
The slide is titled "Desirable Programming Style" and contains the following bulleted list:

- Clarity
  - The program should be clearly written.
  - It should be easy to follow the program logic.
- Meaningful variable names
  - Make variable/constant names meaningful to enhance program clarity.
    - 'area' instead of 'a'
    - 'radius' instead of 'r'
- Program documentation
  - Insert comments in the program to make it easy to understand.
  - Never use too many comments.

A small video inset in the bottom right corner shows a man in a blue shirt speaking.

So, here are some examples use area instead of a, radius instead of r and program documentation. We had said that we should be very generous about writing comments, but if that too many comments then that is also not very desirable.

(Refer Slide Time: 18:47)



The slide is titled "Contd." and contains the following bulleted list:

- Program indentation
  - Use proper indentation.
  - Structure of the program should be immediately visible.

A small video inset in the bottom right corner shows the same man in a blue shirt speaking.

Now you are coming to a very important point that is program indentation what is that? The structure of the program should be immediately feasible. We will give you some examples.

(Refer Slide Time: 18:58)


### Indentation Example #1 :: Good Style

```
#include <stdio.h>
#define PI 3.1415926
/* Compute the area of a circle */

main()
{
    float radius, area;
    float myfunc (float radius);

    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f\n", area);
}

float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```




Here is a good example, here this is simple I am writing in straight line, here there is a function that is fine let us come to another example which will be better illustrating this this is a bad style you see here compare these 2.

(Refer Slide Time: 19:13)

### Indentation Example #1 :: Bad Style

```
#include <stdio.h>
#define PI 3.1415926
/* Compute the area of a circle */
main()
{
    float radius, area;
    float myfunc (float radius);
    scanf ("%f", &radius);
    area = myfunc (radius);
    printf ("\n Area is %f\n", area);
}

float myfunc (float r)
{
    float a;
    a = PI * r * r;
    return (a); /* return result */
}
```



Here there is a declaration then I leave some blank, and then I put the actual code here these are the declaration. So, I understand this a declaration. Compared to this which is easy to understand this is something this is something else. Compared to that here why I where I do not put a gap it is difficult to understand, where the declaration ends and

where the code starts. So, here is another good example of finding the largest of three numbers look at this.

(Refer Slide Time: 19:50)

### Indentation Example #2 :: Good Style

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int a, b, c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>b) && (a>c)) /* Composite condition check */
        printf ("\n Largest is %d", a);
    else
        if (b>c) /* Simple condition check */
            printf ("\n Largest is %d", b);
        else
            printf ("\n Largest is %d", c);
}
```

As I said that if I have read this, but one bad thing is here, there should be a gap here there should have been a gap here. Now here if a and b. So, that is the first diamond a decision box if a is greater than b and a is greater than c then I am doing something alright then I am doing this this part of the program, otherwise I am doing this. So, here you can quickly looking at this indentation, that this printf is a under this and this part is under this, that is very clear from this indentation I have shifted it a little bit.

And so immediately you can understand that this printf is if this condition is true. That means, this part is for this diamond box to be true like that; is a good indentation good style now the same thing.

(Refer Slide Time: 21:06)

The slide displays C code for finding the largest of three numbers. The code is presented with clear indentation. A hand-drawn diagram in blue ink highlights the scope of the `main()` function and the two nested `if` statements. The first `if` statement is labeled 'Composite condition check' and the second is labeled 'Simple condition check'. A small inset video of a presenter is visible in the bottom right corner.

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
    int a, b, c;
    scanf ("%d %d %d", &a, &b, &c);
    if ((a>b) && (a>c)) /* Composite condition check */
        printf ("\n Largest is %d", a);
    else
        if (b>c) /* Simple condition check */
            printf ("\n Largest is %d", b);
        else
            printf ("\n Largest is %d", c);
}
```

Same thing would be bad style if I write it in this way.

(Refer Slide Time: 21:19)

The slide shows the same C code as the previous slide, but without any indentation. This makes it difficult to visually associate the `printf` statements with their respective `if` conditions. A small number '16' is visible in the bottom right corner of the slide.

```
#include <stdio.h>

/* FIND THE LARGEST OF THREE NUMBERS */

main()
{
int a, b, c;
scanf ("%d %d %d", &a, &b, &c);
if ((a>b) && (a>c)) /* Composite condition check */
printf ("\n Largest is %d", a);
else
if (b>c) /* Simple condition check */
printf ("\n Largest is %d", b);
else
printf ("\n Largest is %d", c);
}
```

From here it is not at all clear which one is a corresponding to which one, not very clear if I study it closely I will be able to understand sorry, I will be able to understand that this print f this print f corresponds to this condition.

But immediately when I look at it is not very visible. So, this is a bad style alright. So, indentation is something that is very much prescribed in good programming, this becomes very important when we actually go for more complicated programs.

(Refer Slide Time: 21:58)

The C Character Set

- The C language alphabet:
  - Uppercase letters 'A' to 'Z' ←
  - Lowercase letters 'a' to 'z' ←
  - Digits '0' to '9' ←
  - Certain special characters:

!	#	%	^	&	*	(	)
-	+	=	~	[	]	\	
	;	:	'	"	{	}	,
.	<	>	/	?	blank		

*# include*  
*%.d*  
*%.f*


Now coming to the C character set. In an earlier lecture I had briefly talked about this just as in English we have got a character set a to z capital A to Z. Similarly C language has got a character set with which we make the words we have already seen them.

So, the first thing is of course, the upper case letters, the lower case letters a to z digit 0 to 9 and some special characters like you can see this hash we have already encountered this when we said hash include that is a part of the c symbol set percentage we have seen that when we say percentage d, percentage f you are seen that and this sort of symbols this have already encounter all these are some certain special characters, that are allowed in C.

(Refer Slide Time: 23:09)

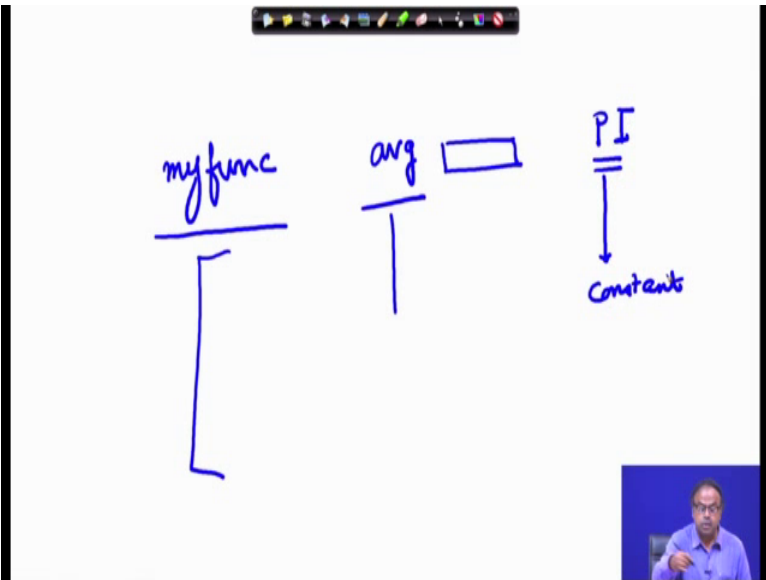
## Identifiers and Keywords

- Identifiers
  - Names given to various program elements (variables, constants, functions, etc.)
  - May consist of *letters*, *digits* and the *underscore* ('\_') character, with no space between.
  - First character must be a letter.
  - An identifier can be arbitrary long.
    - Some C compilers recognize only the first few characters of the name (16 or 31).
  - Case sensitive
    - 'area', 'AREA' and 'Area' are all different.




Now the identifiers and keywords we have also talked about that. The names are given to very different programming elements for example, variables we already know, constants we already know, functions we have already seen each of them are given some names.

(Refer Slide Time: 23:32)



The diagram shows three identifiers: 'myfunc', 'avg', and 'PI'. 'myfunc' is underlined with a single line, and a vertical line extends downwards from the underline. 'avg' is underlined with a single line, and a vertical line extends downwards from the underline, with a small rectangle to its right. 'PI' is underlined with a double line, and a vertical line extends downwards from the double underline to the word 'Constants'.



For example when we wrote myfunc that was the name given to a particular function. When I wrote avg that was the name given to a particular variable that was the name given to a particular variable. When I wrote pi that was the name given to a particular constant right. So, in that we have encountered this. So, now, the how there is some

restriction on the names that I have also mention, may consist of letters digits 0 to 9 and this underscore character with no space in between. Whenever I want to put space, I will put in this underscore character is very useful is very useful to put this underscore character hear what is happening here.

The first character must be a letter, this character must be a letter an identifier can be orbital is long, but that depends on some c compilers some c compilers recognize only the first few characters 16 or 31. And another very important thing that you had talked about that it is case sensitive. A small area written in small letters and area written in capital letters or an area with a mix of small and capital are all different these are not the same this we had mentioned in the last class.

(Refer Slide Time: 25:14)

Contd.

- Keywords
  - Reserved words that have standard, predefined meanings in C.
  - Cannot be used as identifiers.
  - OK within comments.
  - Standard C keywords:

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

There is some keywords some words which are reserved for and have got some predefined meanings in C for example, auto, brake, constant, float you know float has got a specific meaning int got a specific meaning return your seniors got a specific meaning etcetera. If it has got a specific meaning, I c seen it has got a specific meaning, but within comments you are free because it never being so, close the looked at by the compiler the compiler simply text it and print it out.

So, this is another thing that you have to remember, whatever keywords you face during programming you will gradually remember that these are the keywords and that should not be used as a variable name.

(Refer Slide Time: 26:34)

Valid and Invalid Identifiers

- Valid identifiers
  - X
  - abc
  - simple\_interest
  - a123
  - LIST
  - stud\_name
  - Empl\_1
  - Empl\_2
  - avg\_empl\_salary
- Invalid identifiers
  - 10abc
  - my\*name
  - "hello"
  - simple interest
  - (area)
  - %rate

So, here we will conclude this lecture with some examples, you can see some valid identifier x is a valid identifier, a b c of now simple interest I put in an underscore here, a 123 is fine because it starting with a, and alphabet list stud name (Refer Time: 26:55) student name very clear understood employee 1, employee 2 average employee salary see I wanted to write a big thing. And I could take to that buy instead of blank I just put underscore alright invalid identifier studies and why because I cannot start with sorry.

Because, I cannot start with blank sorry I cannot start with the numeral, for this is wrong this is wrong because there is a special character that is been put in here, I am in hide could have written my underscore name. My thing is also is also this not looking nice I could have done this my name, but (Refer Time: 27:41) is not allowed hello is not allowed because here I put in some special characters, simple interest is not allowed because hes a blank I cannot put in a blank, but I could have rewritten simple underscore interest that is quite for light. Area is not valid because it has got a parenthesis, and percentage is using special characters here. So, these are invalid identifier examples of invalid identifier. So, you should keep that in mind. We will continuity with R programming lectures.

So, till now what we have done is we have looked at some of the rules for writing the program and how the variables and the constant should be named; we will continue with this.