

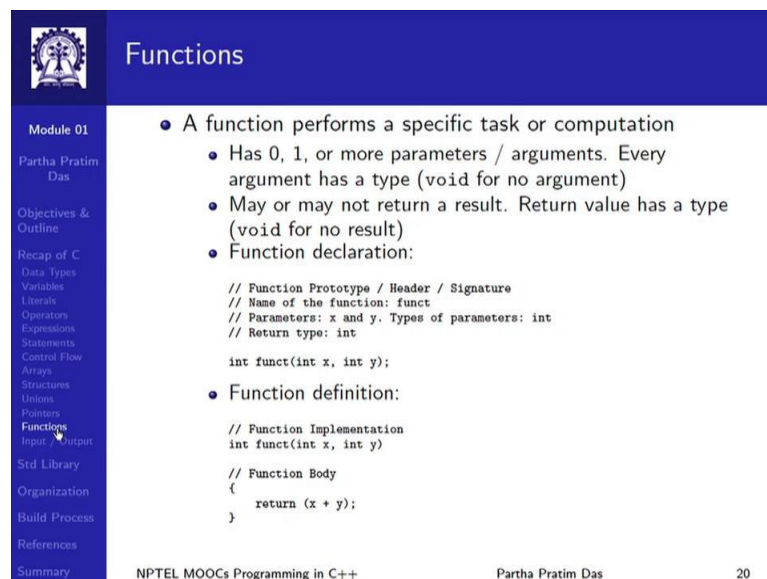
Programming in C++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 03
Recap of C (Part III)

We will continue on Module 01, recapitalization of C. This is the third part. In the first two parts we have talked about the data types, variables, expressions, statements. And, in the second part we have talked about different derived types, arrays, structure, union and pointer.

In the spot, we will start with the basic modular concept of C, which is a function.

(Refer Slide Time: 00:51)



Functions

- A function performs a specific task or computation
 - Has 0, 1, or more parameters / arguments. Every argument has a type (void for no argument)
 - May or may not return a result. Return value has a type (void for no result)
 - Function declaration:

```
// Function Prototype / Header / Signature
// Name of the function: funct
// Parameters: x and y. Types of parameters: int
// Return type: int

int funct(int x, int y);
```
 - Function definition:

```
// Function Implementation
int funct(int x, int y)
{
    // Function Body
    {
        return (x + y);
    }
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 20

So, as you all know a function performs a specific task of computation. A function is like can be treated as a black box, which can take a number of parameters and will give you a result. Now, while usually we will expect a function to take one or more parameters, it is possible that you write a function which does not take any parameter. And, it is typical that it gives you a value at the end of computation; we say it returns a result. But, it is also possible that it may not return any result.

The number of parameters that a function has; each parameter also called argument, we will have a type. And, if we do not want to specify the type we can use void. Return will also have a type. And, we will use void if there is nothing to return.

A typical declaration will look like this; `funct` is the name of the function. On the left is a return type and on the right within this pair of parenthesis, we have the parameters. If there are more than one parameter, they are separated by comma. And, at the end of this parenthesis if you put a semicolon, then we know that you are just interested to talk about what parameters the function takes and what type of value it returns. But, you are not interested to specify how actually this `funct` function computes the result integer from the parameters `x` and `y`.

In such cases if you just dominate the list of arguments with the semicolon, we will say this is a function header or a function prototype or a function signature. And more and more, the signature or prototype kind of terms will keep on occurring in C++. And since in this case, in case of a signature we are not actually specifying how `x` and `y` will be used to compute the result. It is optional whether you specify `x` or `y` or both of them. You could just write it as `int funct parenthesis open int comma int parenthesis close semicolon`; that will also be a valid header.

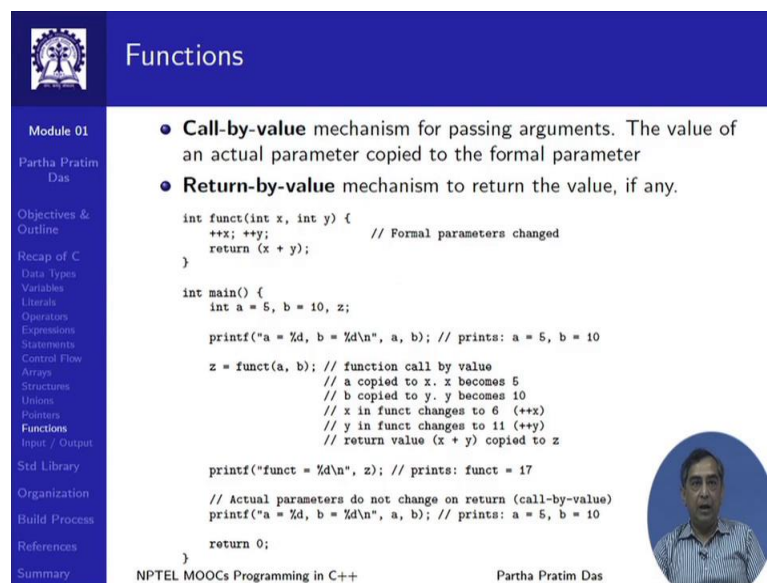
So, what this tells us? It tells us that there are two parameters. First parameter is an integer; second parameter is another integer. It tells that the name of the function is `funct`, it tells us that the type of value it will return is `int`. This is the purpose of the function declaration or the function header.

Now, when we are ready to specify as to what this function will compute or how this function will compute the result from the parameters, then we provide the function body; which is the whole function body is a compound statement. So, it is a pair of parenthesis, curly braces, within which the function body has to be return. So, within that there could be multiple declarations and statements specifying the function body.

A function will have a return statement, which returns an expression of the return type as a final result. If the function is not returning anything, if the function return type is void,

then the return statement will not have an expression associated with it. Please note that in C 89, it was allowed that if a function does not return anything, then it is not necessary to specify the return statement. That protocol still continues. But, for several reasons that will become clear, when we do more of C++. It is strictly avoidable that you write a function and do not put a return. So, even though you may not return anything from the function that is return type is void, please provide a return statement.

(Refer Slide Time: 05:04)



Functions

- **Call-by-value** mechanism for passing arguments. The value of an actual parameter copied to the formal parameter
- **Return-by-value** mechanism to return the value, if any.

```
int funct(int x, int y) {
    ++x; ++y;           // Formal parameters changed
    return (x + y);
}

int main() {
    int a = 5, b = 10, z;

    printf("a = %d, b = %d\n", a, b); // prints: a = 5, b = 10

    z = funct(a, b); // function call by value
                    // a copied to x. x becomes 6
                    // b copied to y. y becomes 11
                    // x in funct changes to 7 (++x)
                    // y in funct changes to 12 (++y)
                    // return value (x + y) copied to z

    printf("funct = %d\n", z); // prints: funct = 17

    // Actual parameters do not change on return (call-by-value)
    printf("a = %d, b = %d\n", a, b); // prints: a = 5, b = 10

    return 0;
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, functions get their parameters by a mechanism, which is known as call by value. I assume that you all know this where at the call site, the example is here. On top, you can see the function body, the whole definition of the function. And, here you see the function invocation or function call, where we are using two parameters that are local to. That their variables local to main to call this function. So the result of this, the first parameter a is copied to the first formal parameter x. So, ab are called actual parameters at the call site; xy are called the formal parameters at the definition site.

And, as I had mentioned in reference to accessing different components of an array and accessing different components of a structure that there are different conventions. Here C follows a positional parameter convention to call function.

So, the first actual parameter corresponds to the first formal parameter; the second actual parameter corresponds to the second actual formal parameter and so on. So, you do not care about the name of the formal parameter at all. The actual parameter value is copied from one parameter to the next and then it is used in the function. Since these are copies, so they have separate memory locations. So, after when funct starts executing, x will have a location different from a.

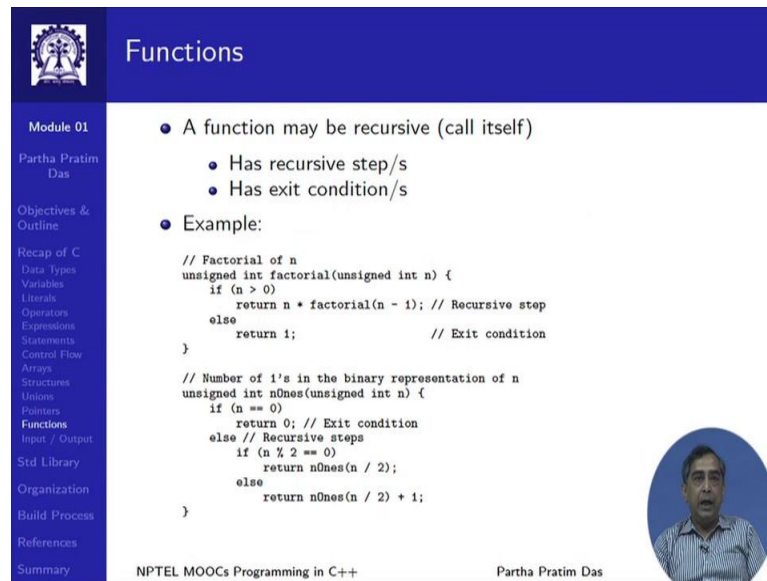
But, since the value has been copied, the value of a is 5. Value of x also to start with will be 5. But, then within the funct body, we have made changes to x and y. We have incremented x, incremented y. We have computed the return value. So, when the function returns, that is, when this value, return value x plus y is computed and given back to the caller, main, and that value is copied to z. When that happens, then we lose the values of the formal parameters x and y. Interestingly, a and b which are actual parameters, they will not change, even though x and y had changed.

And, it is very easy to see. In call by value, you are using separate location. You just, at the beginning you had copied the values. So, since you had copied the values at the beginning, the different locations, when you lose x and y, all the changes we have made to x or to y or to both, only simply gets lost. The actual parameters are not affected.

So, just to illustrate that we are printing the values of n be here. And, we find that they are as original. They are still 5 and 10, even though x and y, the copied values had changed. Which is not very common particularly to the C program? Or, it is a notion that the mechanism by which you return the value is called a return by value mechanism.

And, it is not explicitly mentioned in C because there is no other way to return, to put it straight. I mean, see the only way to return is by copying the value back; because at this point, the function funct which was executing is terminated; will get terminated after giving you the value. So, there is no other way than to keep a copy of that value, which we have assigned to z here to get that value back. As you go to C++, we will see a different mechanism both for call by value as well as for return by value, which will be very powerful.

(Refer Slide Time: 08:53)



Functions

- A function may be recursive (call itself)
 - Has recursive step/s
 - Has exit condition/s
- Example:

```
// Factorial of n
unsigned int factorial(unsigned int n) {
    if (n > 0)
        return n * factorial(n - 1); // Recursive step
    else
        return 1; // Exit condition
}

// Number of 1's in the binary representation of n
unsigned int nOnes(unsigned int n) {
    if (n == 0)
        return 0; // Exit condition
    else // Recursive steps
        if (n % 2 == 0)
            return nOnes(n / 2);
        else
            return nOnes(n / 2) + 1;
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das

Functions can be recursive. I am sure you all have seen this. You all have seen factorial functions like this or the very famous Fibonacci function, which uses two recursive calls. We have seen merge sort, we have seen quick sort; these are all very typical examples of recursive function. Any recursive functions will have a recursive step and one or more exit condition to end the recursion.

Here is another example, which is less commonly used. So, I just put it here for illustration. You have given an unsigned integer n . This recursive function computes a number of 1's in the binary representation of n . You just go through this carefully and is familiarly with itself with the recursion mechanism.

(Refer Slide Time: 09:38)

Function pointers

```
#include <stdio.h>
struct GeoObject {
    enum { CIR = 0, REC, TRG } gCode;
    union {
        struct Cir { double x, y, r; } c;
        struct Rec { double x, y, w, h; } r;
        struct Trg { double x, y, b, h; } t;
    };
};
typedef void(*DrawFunc) (struct GeoObject);
void drawCir(struct GeoObject go) {
    printf("Circle: (%lf, %lf, %lf)\n",
        go.c.x, go.c.y, go.c.r); }
void drawRec(struct GeoObject go) {
    printf("Rect: (%lf, %lf, %lf, %lf)\n",
        go.r.x, go.r.y, go.r.w, go.r.h); }
void drawTrg(struct GeoObject go) {
    printf("Triag: (%lf, %lf, %lf, %lf)\n",
        go.t.x, go.t.y, go.t.b, go.t.h); }
DrawFunc DrawArr[] = { // Array of func. ptrs
    drawCir, drawRec, drawTrg };
int main() {
    struct GeoObject go;
    go.gCode = CIR;
    go.c.x = 2.3; go.c.y = 3.6;
    go.c.r = 1.2;
    DrawArr[go.gCode](go); // Call by ptr
    go.gCode = REC;
    go.r.x = 4.5; go.r.y = 1.9;
    go.r.w = 4.2; go.r.h = 3.8;
    DrawArr[go.gCode](go); // Call by ptr
    go.gCode = TRG;
    go.t.x = 3.1; go.t.y = 2.8;
    go.t.b = 4.4; go.t.h = 2.7;
    DrawArr[go.gCode](go); // Call by ptr
    return 0;
}
```

Circle: (2.300000, 3.600000, 1.200000)
Rect: (4.500000, 1.900000, 4.200000, 3.800000)
Triag: (3.100000, 2.800000, 4.400000, 2.700000)

NPTEL MOOCs Programming in C++ Partha Pratim Das

Functions and pointers can be mixed in a very interesting way. And, this is again one of the very powerful features of C. Before highlighting what we want to specify for the C languages as such, let us just briefly understand this example; because this is an explanation by example.

So, what we are showing at the top or the situation is like this; that we have a collection of geometric objects. The geometric objects could be circle, rectangle or triangles. We assume that the triangles to be right angled triangles, which are aligned with the axis. So, we have structure for each of one them. So, a circle is x, y that the center and the radius, rectangle is x, y, that is, say left bottom corner with a height and a triangle is x, y, which is the right angled corner, the base and the height. And, here we make a structure. Again, this is a typical way to write in C. We make a union of all these because a geometric object is any one of them. And given a particular GeoObject, I, it could be either a circle or a rectangle or a triangle.

Now, how do I know which one is done? So, for that in this structure we are using another enumerated value, gCode, which keeps one of the codes of Cir, Rec or Trg. So, the idea is if you have put, in this union if you have put the structure for a circle you set gCode to cir, which is 0; if you have put in this union, the structure for a rectangle, then

you put gCode as Rec and so on. So, this is a typical use. So, you can see that we have a union of structures.

And then, we have a structure containing that union and indexing code to understand what that union has. And, if you, in C programming this was a very common paradigm to do. And, we will use this example later on in C++, as I said in the context of inheritance. And, show that how in C++ this can be done lot more efficiently.

Now, the situation is since these are three different types of geometric objects and my task is to draw them on the screen. So I have, now the way you draw a circle and the way you draw a rectangle and way you draw a right angled triangle are all different. So, naturally one pro code cannot be written. One function cannot be written, which can draw all three of them. So, you assume that there are 3 different functions

In C, all of them are actually drawing. But in C, certainly we cannot have two functions having the same name. So, I call them as DrawCircle, draw a rectangle, draw triangle. Now, each one of them takes the same structure GeoObject. For example, if I look into DrawCircle, then it takes the GeoObject. And, in the GeoObject it will be able to find out that it indeed is a circle, if it checks for the gCode. And, it will print all these values here. Actually this, in the real DrawFuncion, the print will not be there. There will be graphics function calls to actually draw the circle. But just to illustrate the point, I am showing you here through prints. So, these are the three different functions, which can draw three different types of geometric objects that exist.

Now, we want to write a routine main function, which given the gCode of any of this objects. And, the structural parameters of that object should be able to draw it. So, the idea is I would like to call the DrawFuncion. Any of this DrawFuncion not using their name. But, from the fact that if my gCode is Cir and I want to call the draw, this draw should get called. But, if my gCode is Rec and I want to call this, drawRec should get called and so on.

So, we want to make the call uniform, so that I do not really need to know what particular object I have at this point of time. What object I have is already a part of the

object structure; because I have a gCode. And then, it should be possible that the corresponding function will get called. The mechanism that is done here is I create an array called DrawFuncion and put all this function names. As you put this function names, these are called function pointers. A function name used by itself without the pair of parenthesis; the pair of parentheses, as we will see is called function operators, which tells you that the function is being invoked here.

Those operators are not used. So, these are just the function names. They are the function pointers. Or, in other words these are the address where the respective function starts. So, this is an array. DrawFunc draw, i am sorry. DrawArrr is an array of these function pointers. And, what will you see? That the zeroth entry in this is DrawCir, which is drawing the circle. So, if the gCode is zero, that is, Cir. And then, if I access this array with that gCode, so go dot gCode is Cir here, which is zero; which means the zeroeth function pointers.

So, the value of DrawArrr go dot gCode turns out to be DrawCir, whereas if I put go dot gCode as Rec, if it is a rectangle structure, then go dot gCode here is 1. After zero, this is 1; which means in the array I am accessing location 1. So, this particular function becomes drawRec function and so on.

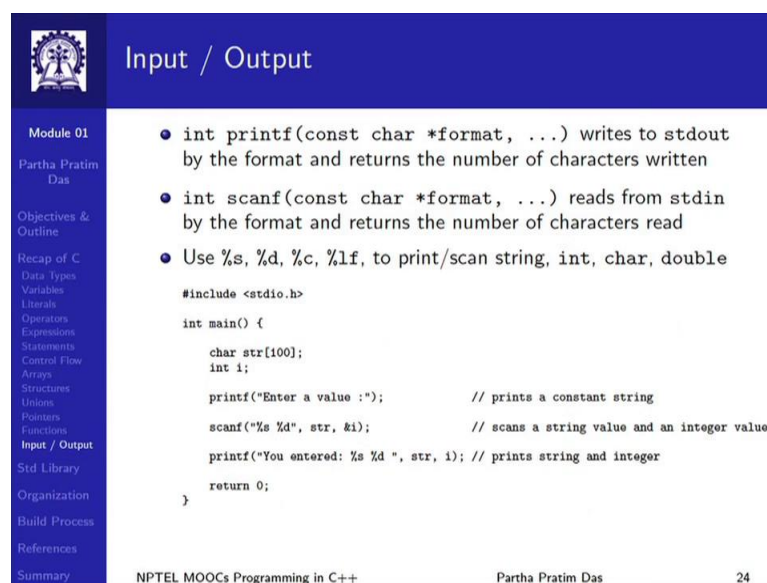
So, the notation is little bit not so common. So, you do not see the typical function like name. But you, basically it looks like an array element which is the pointer. And, but when you get the pair of parentheses, you know that there is a function operator which invokes the function that you have access from this array.

Now, to be able to create a function pointer or array of function pointers, what does a array need? Array needs all elements must be of the same time. So, all these function pointers must be of the same type; which means they must take the same type of parameters and must return the same type of value. So, we ensure that by creating a type or aliasing a type DrawFunc here by type def; which says that it takes a structure GeoObject and returns a void. We want to say that this is a pointer type. That is why the name is not DrawFunc. The name is given as star DrawFunc. The way, if we have to define a integer pointer we say int star p. So, basically the name is not p. Name is as if

you are saying that the star p is an integer. So, here what you are saying is star DrawFunc is a function which takes a GeoObject, gives a void.

And, now if you look into all these three candidate functions, all of them have that same signature. So, they have the uniform signature. So, they are all of this DrawFunc type, which is a pointer to a function taking a structure geo, struct GeoObject and returning nothing. So, this is a typical use of; the function pointers have very powerful mechanism in C. If you; all the graphic systems, the menus, everything you use is this concept of function pointers. And, as we go to C++, we will show that is how C++ is significantly improved this and made it easier and more robust to use, in terms of what is known as a virtual function table. That will come when we discuss that.

(Refer Slide Time: 18:16)



The slide is titled "Input / Output" and features a blue header with a logo on the left. A vertical sidebar on the left lists the module's contents, with "Input / Output" highlighted. The main content area contains a bulleted list of points about printf and scanf, followed by a C code snippet demonstrating their use. The footer includes the course name "NPTEL MOOCs Programming in C++", the presenter's name "Partha Pratim Das", and the slide number "24".

Module 01
Partha Pratim Das
Objectives & Outline
Recap of C
Data Types
Variables
Literals
Operators
Expressions
Statements
Control Flow
Arrays
Structures
Unions
Pointers
Functions
Input / Output
Std Library
Organization
Build Process
References
Summary

- `int printf(const char *format, ...)` writes to stdout by the format and returns the number of characters written
- `int scanf(const char *format, ...)` reads from stdin by the format and returns the number of characters read
- Use `%s, %d, %c, %lf`, to print/scan string, int, char, double

```
#include <stdio.h>

int main() {
    char str[100];
    int i;
    printf("Enter a value :");           // prints a constant string
    scanf("%s %d", str, &i);           // scans a string value and an integer value
    printf("You entered: %s %d ", str, i); // prints string and integer
    return 0;
}
```

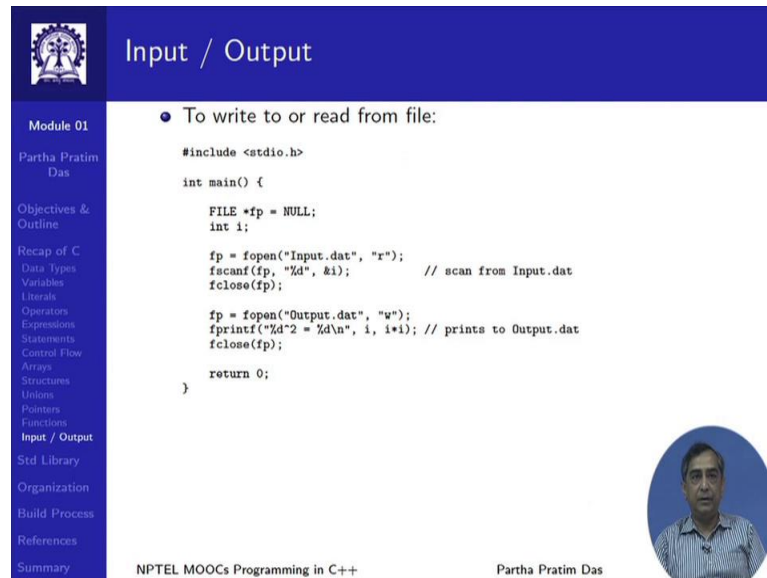
NPTEL MOOCs Programming in C++ Partha Pratim Das 24

Finally, to close the (Refer Time: 18:20) on C programming language, you know there are input, output functions available in the stdio dot h. These are common; most common are printf and scanf, which use format strings. It uses; also another interesting part of C programming known as ellipsis or these are known as variadic functions, given that you do not know how many parameters you will put in printf.

So, we ever wrote the code of printf, did not know whether you want to print 5 values or

10 values or 1 value. So, a typical format bit structure has been created. I will not go into depth of that. We will cross that when we come to dealing with discussing I/O s in C++. To contrast, how C++ avoids variadic input output functions. But, these are the typical ways to do input output in C, which I am sure all of you are very familiar with.

(Refer Slide Time: 19:13)



The slide is titled "Input / Output" and is part of "Module 01" by Partha Pratim Das. It contains a bullet point: "To write to or read from file:". Below this, there is a C code snippet demonstrating file operations. The code includes the header <stdio.h>, declares a FILE pointer *fp and an integer i, opens "Input.dat" for reading, scans an integer from the file, closes the file, opens "Output.dat" for writing, prints the value of i to the output file, and returns 0. A small circular portrait of Partha Pratim Das is visible in the bottom right corner of the slide content area.

```
#include <stdio.h>

int main() {

    FILE *fp = NULL;
    int i;

    fp = fopen("Input.dat", "r");
    fscanf(fp, "%d", &i); // scan from Input.dat
    fclose(fp);

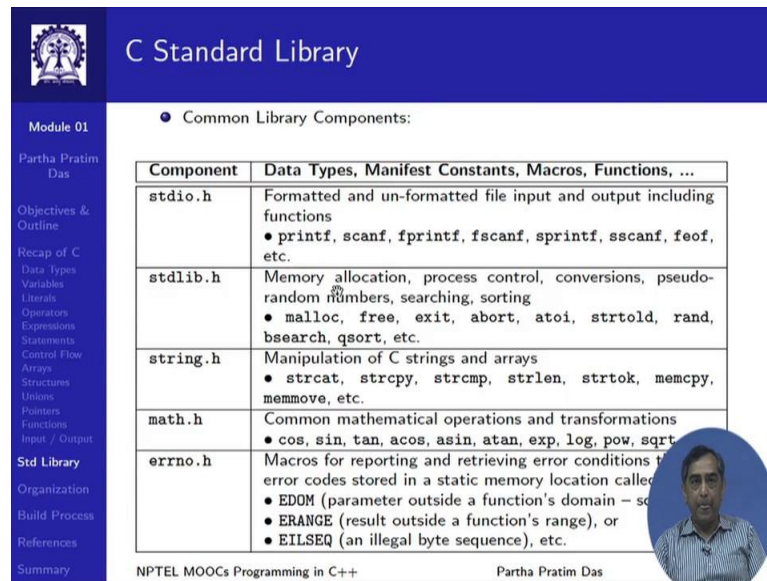
    fp = fopen("Output.dat", "w");
    fprintf("%d^2 = %d\n", i, i*i); // prints to Output.dat
    fclose(fp);

    return 0;
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das

These are examples showing you how to do the input output with array, with files. So, use a file pointer, you do f open to open that and so on.

(Refer Slide Time: 19:25)



The slide is titled "C Standard Library" and features a blue header. On the left, there is a vertical navigation menu with the following items: Module 01, Partha Pratim Das, Objectives & Outline, Recap of C, Data Types, Variables, Literals, Operators, Expressions, Statements, Control Flow, Arrays, Structures, Unions, Pointers, Functions, Input / Output, Std Library, Organization, Build Process, References, and Summary. The main content area is titled "Common Library Components:" and contains a table with two columns: "Component" and "Data Types, Manifest Constants, Macros, Functions, ...". The table lists five components: `stdio.h`, `stdlib.h`, `string.h`, `math.h`, and `errno.h`. A small circular inset image of a man is visible in the bottom right corner of the slide content.

Component	Data Types, Manifest Constants, Macros, Functions, ...
<code>stdio.h</code>	Formatted and un-formatted file input and output including functions <ul style="list-style-type: none">• <code>printf</code>, <code>scanf</code>, <code>fprintf</code>, <code>fscanf</code>, <code>sprintf</code>, <code>sscanf</code>, <code>feof</code>, etc.
<code>stdlib.h</code>	Memory allocation, process control, conversions, pseudo-random numbers, searching, sorting <ul style="list-style-type: none">• <code>malloc</code>, <code>free</code>, <code>exit</code>, <code>abort</code>, <code>atoi</code>, <code>strtol</code>, <code>rand</code>, <code>bsearch</code>, <code>qsort</code>, etc.
<code>string.h</code>	Manipulation of C strings and arrays <ul style="list-style-type: none">• <code>strcat</code>, <code>strcpy</code>, <code>strcmp</code>, <code>strlen</code>, <code>strtok</code>, <code>memcpy</code>, <code>memmove</code>, etc.
<code>math.h</code>	Common mathematical operations and transformations <ul style="list-style-type: none">• <code>cos</code>, <code>sin</code>, <code>tan</code>, <code>acos</code>, <code>asin</code>, <code>atan</code>, <code>exp</code>, <code>log</code>, <code>pow</code>, <code>sqrt</code>, etc.
<code>errno.h</code>	Macros for reporting and retrieving error conditions to error codes stored in a static memory location called <code>errno</code> . <ul style="list-style-type: none">• <code>EDOM</code> (parameter outside a function's domain – see <code>atan</code>)• <code>ERANGE</code> (result outside a function's range), or• <code>EILSEQ</code> (an illegal byte sequence), etc.

NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, we will move on to. Just mention that if you just had the C language, then you could not have written any program because you see, saw the first program which we discussed in the first part of this module; “Hello World” program, we need a `printf`. And, it is not easy to write a `printf`.

So, to be able to effectively use a programming language, you need a basic library to be available. And, any language specifies what is a library which every compiler on must provide, and that library is known as a standard library. That library has all the function definition, function names, the hidden names and all of them, fixed by the language design. C has a standard library. As we will see as we move to C++, we will see that C++ has a much stronger standard library. But, C standard library also is quite powerful and interesting. The whole of the C standard library is also available in C++. So, it is not that in C++ things will get replaced; only new things will get added.

Now, C standard library has many components. Every components comes under one header file. And there are, you can look up the total list. But, I have just put in the 5, which are most frequently used. The input output header; standard library header, which has a mix of things to do memory allocation, conversion, searching, sorting and so on.

The string for manipulating C strings, the math library for all different kinds of common mathematical functions and the error header which deals with the error, different error numbers and error ranges and so on. So, please familiarize yourself more with the standard library. And, I would suggest that whenever you are using some standard library component, please look up the manual to see what all other functions that component has. And, it will really, it might so happen that in many places, you are writing certain functions for doing a task, which is already available in the standard library.

(Refer Slide Time: 21:33)

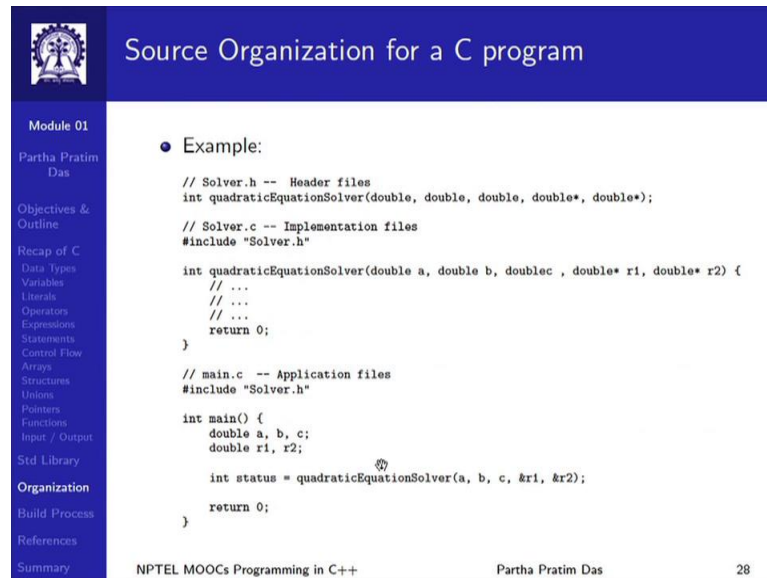
The slide is titled "Source Organization for a C program". On the left side, there is a vertical sidebar with a blue background and white text. At the top of the sidebar is a small logo. Below the logo, the text reads: "Module 01", "Partha Pratim Das", "Objectives & Outline", "Recap of C", "Data Types", "Variables", "Literals", "Operators", "Expressions", "Statements", "Control Flow", "Arrays", "Structures", "Unions", "Pointers", "Functions", "Input / Output", "Std Library", "Organization", "Build Process", "References", and "Summary". The "Organization" item is highlighted in white. The main content area of the slide has a white background and a blue header. The title "Source Organization for a C program" is in white text on the blue header. Below the title, the text "Header Files" is in blue. There are three bullet points in blue text: "A header file has extension .h and contains C function declarations and macro definitions to be shared between several source files", "There are two types of header files:", and "Header files are included using the #include pre-processing directive". The second bullet point has two sub-bullets: "Files that the programmer writes" and "Files from standard library". The third bullet point has two sub-bullets: "#include <file> for system header files" and "#include \"file\" for header files of your own program". At the bottom of the slide, there is a footer with the text "NPTEL MOOCs Programming in C++", "Partha Pratim Das", and "27".

At the end, I should also mention that a C program needs to be properly organized. And, it is good to separate between header files and source files, as I said, like the way the library is organized. When you include `stdio.h`, you are actually including the function prototypes or function headers. The implementations are given somewhere else.

So, when you make use of certain, write some functions of your own, you should also separate them in terms of header, and the files that implement those functions. So, in your program you will typically have two kinds of header files; the header files that you have written and the header files that the standard has come from the standard library. So, the header files that comes from the standard library or system header files, must come, must be specified with these kinds of angular brackets.

And, the header files written by you must be within double quotes. We will explain the reason for this more, when we go deeper into C++. This is basically, this basically tells the system as to where to look for this file; whether to look for them in system directories or you have to look for them in your current (Refer Time: 22:52) directory.

(Refer Slide Time: 22:54)



The slide is titled "Source Organization for a C program" and features a blue header with a logo on the left. A vertical sidebar on the left contains a navigation menu with items like "Module 01", "Partha Pratim Das", "Objectives & Outline", "Recap of C", "Data Types", "Variables", "Literals", "Operators", "Expressions", "Statements", "Control Flow", "Arrays", "Structures", "Unions", "Pointers", "Functions", "Input / Output", "Std Library", "Organization", "Build Process", "References", and "Summary". The main content area displays an example of source code organization:

- Example:

```
// Solver.h -- Header files
int quadraticEquationSolver(double, double, double, double*, double*);

// Solver.c -- Implementation files
#include "Solver.h"

int quadraticEquationSolver(double a, double b, double c, double* r1, double* r2) {
    // ...
    // ...
    // ...
    return 0;
}

// main.c -- Application files
#include "Solver.h"

int main() {
    double a, b, c;
    double r1, r2;

    int status = quadraticEquationSolver(a, b, c, &r1, &r2);

    return 0;
}
```

At the bottom of the slide, it says "NPTEL MOOCs Programming in C++" on the left, "Partha Pratim Das" in the center, and "28" on the right.

So, this is a typical example. I am just trying to deal with an application that needs quadratic equations to be solved. So, there will be several context of why you need to solve the quadratic equation. It could be computing interest; it could be solving some (Refer Time: 23:14) equations, whatever.

So, this is the application program, which assumes that you have a quadratic equation solver. So, you are implementing that. So, you have one header file Solver dot h, which gives a interface of that function or the prototype of the function with all different parameters. You have another, where you put the code of this function or the implementation of the function body. So, you call this Solver dot h; you call this Solver dot c. Solver dot h has all the details. So, here formal parameters have name; here formal parameters do not have name. They are just types given.

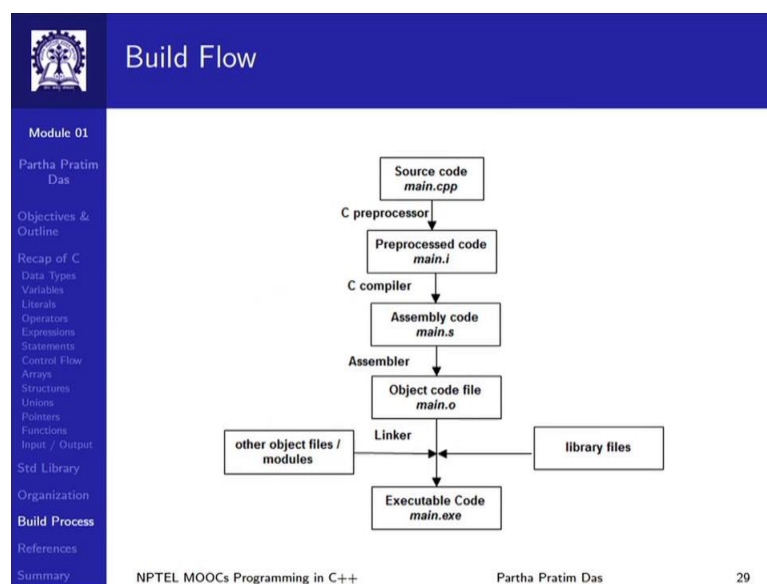
And, in the application you just need to know the header. You do not need to know how

the quadratic equation is solved. All that you need to know is in the signature it has five parameters; first three of them are the three coefficients of the quadratic equation and the next two are the two pointers or two address locations, where the two parts of the result needs to be stored, in case the quadratic equation is solved with a complex value result. So, you need the real and imaginary part to be stored.

So, you just need to know this information to be able to write the application. So, your application file includes Solver dot h, just the header. Your implementation of the quadratic equation solver also includes Solver dot h, which makes sure that between this solver and your application, the same header is included. So, there is no chance of a mistake. So, you should always separate this out that headers which are common functions, macros, constants, manifest constants that are to be shared between source files.

Then for every function or for group of functions, you have separate implementation files, which will include the same header. And, this application files will only include the headers. And, we will not need the implementation files to be referred. So, please follow this organization. And, we will see more of that as we go to C++.

(Refer Slide Time: 25:22)



In a typical situation, this is how you prepare your code for execution. You start with a source code, which could be C or C++. Then as you compile, you may compile through an IDE. Just say built or compile something like that or you could do it through a text based interface like you say g plus plus or some C++, something like that. This is the process stages that happen. And, this is just the outline. We will discuss more as we go into the C++ stages.

Your file, your source file first goes through C preprocessor, which takes care of hash include, hash define, this kind of preprocessor directives, then it gets compiled into some kind of a low level language known as assembly language. You see those files have dot s or dot, may be some other systems will have different kinds of file name extensions. Then you assemble them to generate the object files which are called dot o or dot obj. And, there could be several other object files that come from your libraries, like if you have included stdio dot h.

Then, that stdio dot h gets included at this stage, but the actual implementation is not available. So, that has already been compiled through this process separately and a dot o file has been created. And that dot o file is getting, as you say, linked to your program. If you are using an IDE, you will not be able to see this explicitly because IDE sets it up for you. If you are doing it from the command line, you will have to actually use minus l and include that standard library. And, all of these together will give you the executable code. And, as you go more into C++ we will see that how the built process also impacts the C++ programming and the different programming features.

(Refer Slide Time: 27:31)

The slide is titled "Tools" and is part of "Module 01" by Partha Pratim Das. It lists two bullet points: "Development IDE: Code::Blocks 16.01" and "Compiler: -std=c++98 and -std=c99". The slide also features a navigation menu on the left with items like "Objectives & Outline", "Recap of C", "Data Types", "Variables", "Literals", "Operators", "Expressions", "Statements", "Control Flow", "Arrays", "Structures", "Unions", "Pointers", "Functions", "Input / Output", "Std Library", "Organization", "Build Process", "References", and "Summary". The footer includes "NPTEL MOOCs Programming in C++", "Partha Pratim Das", and the slide number "31".

In terms of tools, there is multiple IDE s available, like Code::Block. There is a visual studio. There is eclipse. So, we will advise that you use some IDE, which is open source. Typically, Code Block or eclipse. And, whatever IDE you are using, you specify that these are the standards; like C 99 is a standard that you are using. Otherwise, the behavior of the programs that we, as we show them and the behavior of the programs as we will experience in the IDE will be different. So always use, for C you use C 99; for C++ we will use C++ 98. Or, we will see, in some context we will use C++ 03. We will come to that.

(Refer Slide Time: 28:26)

References

- Kernighan, Brian W., and Dennis M. Ritchie. The C Programming Language. Vol. 2. Englewood Cliffs: Prentice-Hall, 1988.
- King, Kim N., and Kim King. C programming: A Modern Approach. Norton, 1996.

NPTEL MOOCs Programming in C++ Partha Pratim Das

These are the reference books; in case you want to refer to.

(Refer Slide Time: 28:33)

Module Summary

- Revised the concept of variables and literals in C
- Revised the various data types and operators of C
- Re-iterated through the control constructs of C
- Re-iterated through the concepts of functions and pointers of C
- Re-iterated through the program organization of C and the build process.

NPTEL MOOCs Programming in C++ Partha Pratim Das

And, so in summary in this whole module we have achieved this. We have revised the concepts of fundamental concepts in C; variables, data types, literals, control constructs. And, we have iterated through functions and pointers and the organization and the build

process.

So, I would expect that you, this will give a quick recap to your fundamentals in C. And, if you have found certain points that you did not understand well or the examples were not well absorbed, then please go through them again or ask questions to us on the blog. But, with this module we will expect that this is a level of C that you are prepared with to be able to proceed with the C++ language training.

We will close module one here. In module two onwards, we will start showing you how in C++ some of the common examples that we have seen here or some of the other common examples in C can be done more efficiently, effectively and in a more robust manner in C++.