



## Lec 01 Module 01 Recap of C (Lecture 01)

C++ Object oriented (Indian Institute of Technology Kharagpur)



Scan to open on Studocu

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

**Lecture - 01**  
**Recap of C (Part I)**

Welcome to programming in C++. This will be 20 hour course, where we will talk about various aspects of the C++ programming language and it will be divided naturally into about 40 modules that you will study one after the other.

The main emphasis of this course is to teach, how C++ programming language should be used in designing and implementing complex software systems.

(Refer Slide Time: 00:57)

**Module 01**  
Partha Pratim Das

**Module 01: Programming in C++**  
**Recap of C**

Partha Pratim Das  
Department of Computer Science and Engineering  
Indian Institute of Technology, Kharagpur  
[ppd@cse.iitkgp.ernet.in](mailto:ppd@cse.iitkgp.ernet.in)

Tanwi Mallick  
Srijoni Majumdar  
Himadri B G S Bhuyan

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

NPTEL MOOCs Programming in C++ Partha Pratim Das 1

So, you all will be aware that C++ is object oriented or object based programming language and I would assume that you know C language, may not be at a very depth, but you have the overall idea about the C language. So, we will start from there, in the module 1, we will primarily talk about recapitulating various specific aspects of C programming. This is just to make sure that you can, if required you can revisit those concepts and before we get deep into the C++ language, you can be familiar with all the

C programming requirements because C is a language, which is backward compatible to C++. So, we will first get started with recapitulation of C.

(Refer Slide Time: 01:56)

**Module Objectives**

**Module 01**  
Partha Pratim Das

**Objectives & Outline**

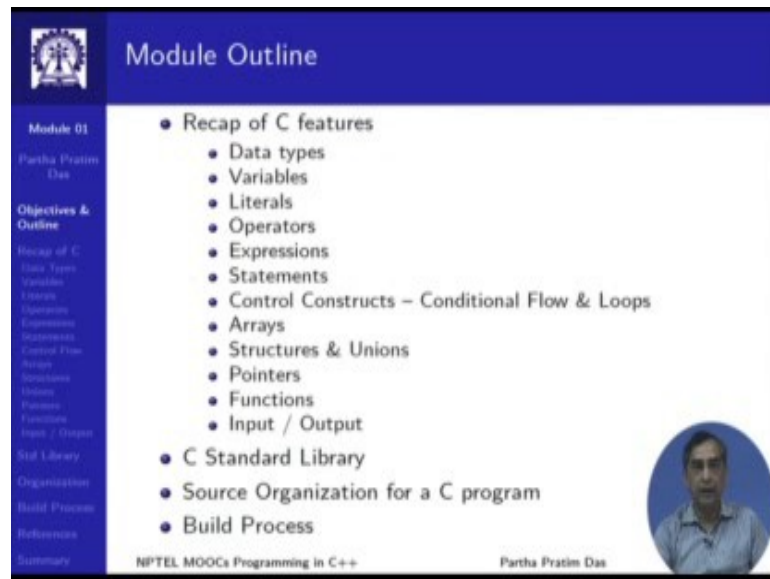
- Recap of C
- Data Types
- Variables
- Control
- Operators
- Expressions
- Statements
- Control Flow
- Arrays
- Structures
- Unions
- Enums
- Functions
- Input / Output
- Std Library
- Organization
- Build Process
- References
- Summary

- Revisit the concepts of C language
- Revisit C Standard Library components
- Revisit the Organization and Build Process for C programs
- Create the foundation for the concepts of C++ with backward compatibility to C

NPTEL MOOCs Programming in C++ Partha Pratim Das 2

So, these are objectives to revisit the concepts, particularly we will look into C Standard Library, besides the C language and programming aspects. We will briefly discuss about the organization of C program, how C program is to be organized possibly. So, far you have only written code in terms of one single file using possibly 1 or 2 functions only. One of them must be main, as you know, we will show how to organize programs better and with this we will have a foundation to; for the C++ programming language.

(Refer Slide Time: 02:36)



The slide titled "Module Outline" features a blue header with the NPTEL logo and title. A left sidebar lists navigation options: Module 01, Partha Pratim Das, Objectives & Outline, Recap of C, Data Types, Variables, Literals, Operators, Expressions, Statements, Control Constructs – Conditional Flow & Loops, Arrays, Structures & Unions, Pointers, Functions, Input / Output, Std Library, Organization, Build Process, References, and Summary. The main content area lists the module topics: Recap of C features (Data types, Variables, Literals, Operators, Expressions, Statements, Control Constructs – Conditional Flow & Loops, Arrays, Structures & Unions, Pointers, Functions, Input / Output), C Standard Library, Source Organization for a C program, and Build Process. A circular portrait of Partha Pratim Das is in the bottom right corner.

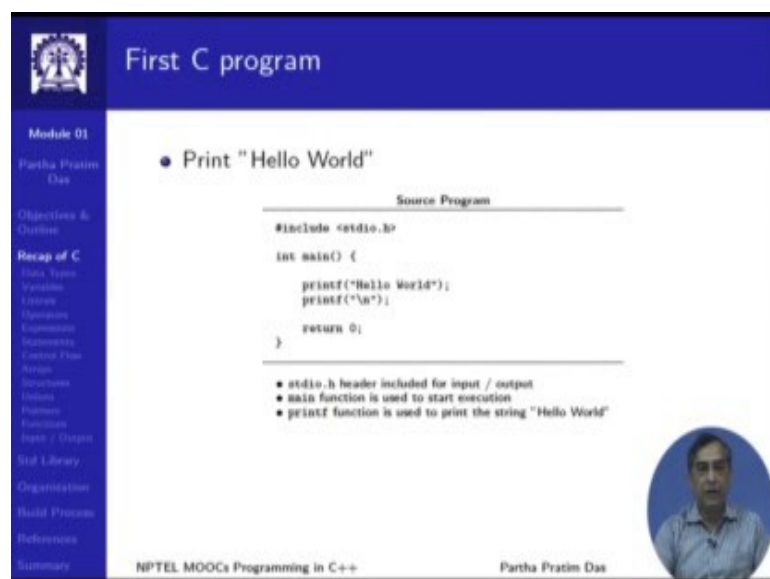
**Module Outline**

- Recap of C features
  - Data types
  - Variables
  - Literals
  - Operators
  - Expressions
  - Statements
  - Control Constructs – Conditional Flow & Loops
  - Arrays
  - Structures & Unions
  - Pointers
  - Functions
  - Input / Output
- C Standard Library
- Source Organization for a C program
- Build Process

NPTEL MOOCs Programming in C++ Partha Pratim Das

These are the different topics to be done, the outline of the module; this is for reference. As the presentation, we will proceed on the left of your screen, you will see this outline and it will be highlighted as to which particular topic we are talking about.

(Refer Slide Time: 02:54)



The slide titled "First C program" features a blue header with the NPTEL logo and title. The left sidebar is identical to the previous slide. The main content area shows the title "Print 'Hello World'" followed by a code block labeled "Source Program" containing the C code for printing "Hello World". Below the code, three bullet points explain the code: `stdio.h` header is included for input / output, `main` function is used to start execution, and `printf` function is used to print the string "Hello World". A circular portrait of Partha Pratim Das is in the bottom right corner.

**First C program**

- Print "Hello World"

Source Program

```
#include <stdio.h>

int main() {
    printf("Hello World");
    printf("\n");

    return 0;
}
```

- `stdio.h` header included for input / output
- `main` function is used to start execution
- `printf` function is used to print the string "Hello World"

NPTEL MOOCs Programming in C++ Partha Pratim Das

So, this is the first program “Hello World” which I am sure all of you have studied. This is also the starting program in Kerning and Ritchie’s famous book. We use ‘printf’ from the <stdio> library and print the hello world on to the terminal or which is formally set to with the <stdio> out file. The main function is one that you can see here is where the execution starts and then you print this string and print ‘\n’, which means you basically go to the next line; new line.

(Refer Slide Time: 03:37)

**Data Types**

Data types in C are used for declaring variables and deciding on storage and computations:

- **Built-in / Basic** data types are used to define raw data
  - char
  - int
  - float
  - double

Additionally, C99 defines:

- bool

All data items of a given type has the same size (in bytes). The size is implementation-defined.

- **Enumerated Type** data are internally of int type and operates on a select subset.

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

C has a number of data types. Those are known as char, which is character; int, float and double; float and double for the floating point numbers and integers are for the so called whole numbers.

Now, here I should mention that the C that you commonly use is known as C89, C89 is a first standard of C that was created by ANSI, the standardization organization and subsequently in 99, another standard was released, this is called C99, so most of the compilers today follows C99 standard. We will also expect C99 to be followed. So, when we talk about C, we will try to highlight, if few things have become different in C99. So, in terms of data type as you can see, there is a new data type bool, which has got added in C99.

In C89, you could still have Boolean values, which can be true or false based on it being an integer value. So, if it is 0, it is false; otherwise it is true. But in C99, there is a separate type `bool`. Every data type as you know, this built-in data types has a size that is given in bytes and you can use 'sizeof' operator to get that. You can define enumerated types which are basically integer values which are given some symbolic names.

(Refer Slide Time: 05:17)

The slide is titled "Data Types" and is part of "Module 01" by Partha Pratim Das. It lists "Data types in C further include:" followed by three categories: 

- void:** The type specifier `void` indicates no type.
- Derived data types include:**
  - Array
  - Structure – `struct` & `union`
  - Pointer
  - Function
  - String – C-Strings are really not a type; but can be made to behave as such using functions from `<string.h>` in standard library
- Type modifiers include:**
  - `short`
  - `long`
  - `signed`
  - `unsigned`

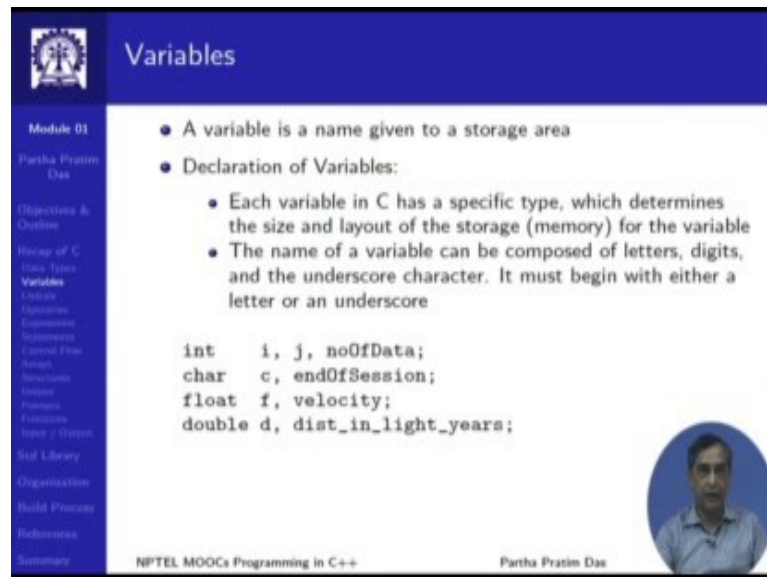
The slide footer includes "NPTEL MOOCs Programming in C++", "Partha Pratim Das", and the number "6".

Other data types in C include 'void'. 'void' is not a type, it is very interesting use and as we go into C++, we will see various different use of void. Void is where you would need to use a type, you can use the type void, but it actually says that there is no type. So, it is like, when we do arithmetic we have a 0. So, I can add 0 to x and it does not change x. So, as we say every system needs a 0. So, void is a 0 of the type system, as we will see more in C++.

Then based on this built in types, there are various derived types that supported the array, structure and union, pointer; we can have functions and it is the commonly called, there is a string type in C called C strings these days. Very strictly speaking string is not a type in C, you will understand that more when we go into C++. C strings are actually a collection of functions in `<string.h>` header, which allow us to manipulate strings in C.

Finally, the data types can be modified for their size and whether they will be signed or unsigned and these 4 type modifiers are used in C.

(Refer Slide Time: 06:42)



The slide is titled "Variables" and is part of Module 01. It contains a list of bullet points defining variables and their declaration rules in C. Below the text, there is a code block showing variable declarations for integers, characters, floats, and doubles. A small circular portrait of the instructor, Partha Pratim Das, is visible in the bottom right corner of the slide content area.

**Variables**

- A variable is a name given to a storage area
- Declaration of Variables:
  - Each variable in C has a specific type, which determines the size and layout of the storage (memory) for the variable
  - The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore


```
int    i, j, noOfData;  
char   c, endOfSession;  
float  f, velocity;  
double d, dist_in_light_years;
```

NPTEL MOOCs Programming in C++ Partha Pratim Das

We will move on there are as you know the variables in C. Variables have certain, their names can be defined in certain ways starting with an alpha or an underscore and then extended with alpha numeric.

Here are some examples of different variable names, while it is often convenient to name variables with single letters or 1-2 letters. It is advised that you use variable names which make some meaning. So, we are saying character 'endOfSession', you could have just called it 'c' or 'd' or 'a', but it is better to give it in name from which it can be understood as to what the variable means.

(Refer Slide Time: 07:28)




## Variables

Module 01  
Partha Pratim Das

Objectives & Outline  
Recap of C  
Data Types  
**Variables**  
Constants  
Operators  
Expressions  
Statements  
Control Flow  
Arrays  
Structures  
Unions  
Pointers  
Functions  
Input / Output  
Std Library  
Organization  
Build Programs  
References  
Summary

- Initialization of Variables:
  - Initialization is setting an initial value to a variable at its definition

```
int    i = 10, j = 20, numberOfWorkDays = 22;  
char   c = 'x';  
float  weight = 4.5;  
double density = 0.0;
```

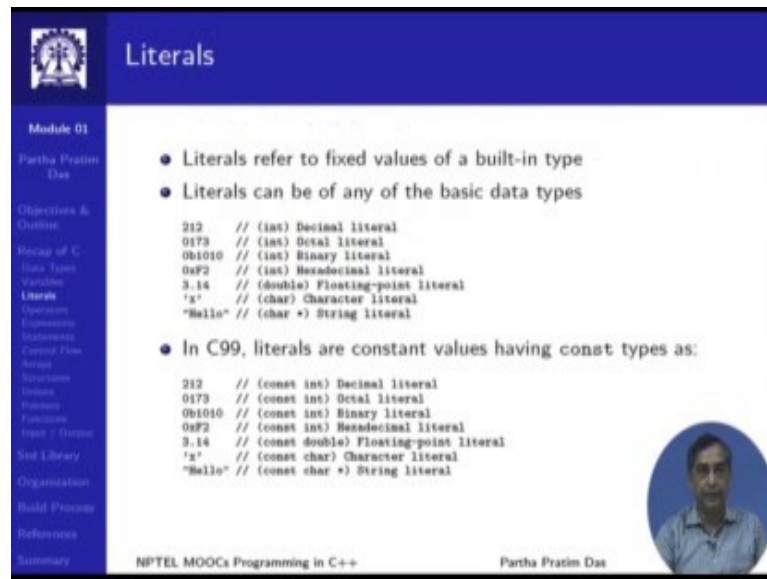


NPTEL MOOCs Programming in C++ Partha Pratim Das

When the variables are declared as they are declared here, then the variables can be initialized also. That initialization is optional. So, when we say `int i` initialized with 10. It means that 'i' is an int type variable whose value at the point of definition itself will become 10. So, if you do not give initialization then it is uninitialized variable which will have an unknown value to start with. Certainly, it is very good to initialize all variables that we declare and define.



(Refer Slide Time: 08:05)



**Literals**

- Literals refer to fixed values of a built-in type
- Literals can be of any of the basic data types

```
212    // (int) Decimal literal
0179   // (int) Octal literal
0b1010 // (int) Binary literal
0xF2   // (int) Hexadecimal literal
3.14   // (double) Floating-point literal
'x'    // (char) Character literal
"Hello" // (char *) String literal
```

- In C99, literals are constant values having const types as:

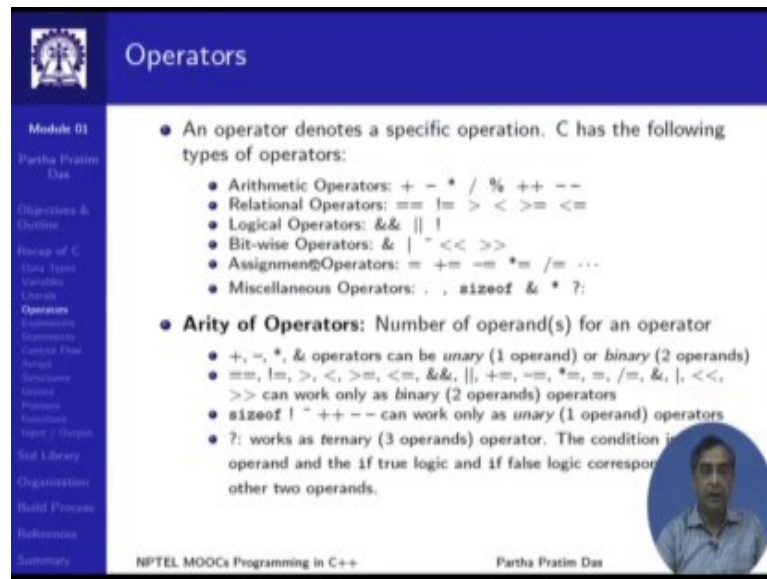
```
212    // (const int) Decimal literal
0179   // (const int) Octal literal
0b1010 // (const int) Binary literal
0xF2   // (const int) Hexadecimal literal
3.14   // (const double) Floating-point literal
'x'    // (const char) Character literal
"Hello" // (const char *) String literal
```

NPTEL MOOCs Programming in C++ Partha Pratim Das

C has a number of literals which are basically fixed values of built-in types depending on how you write a particular literal, the type of that literal is decided. For example, if you just have a sequence of numerals then it becomes a decimal integer type, but if you prefix that with 0, then it is considered to be an octal type, a base eight number. If you prefix it with 0x, then it is considered to be a hexadecimal literal and so on. Character literals are within single quotes and string literals are within double quotes.

With C99, we have the introduction of what is known as 'const' types, that are constants and we will have more discussions of that in depth when we do C++. So, in C89 the literals are basically fixed values, but in C99, they are considered to be constant type data. So, '212' in C99 it will be considered a const int.

(Refer Slide Time: 09:14)



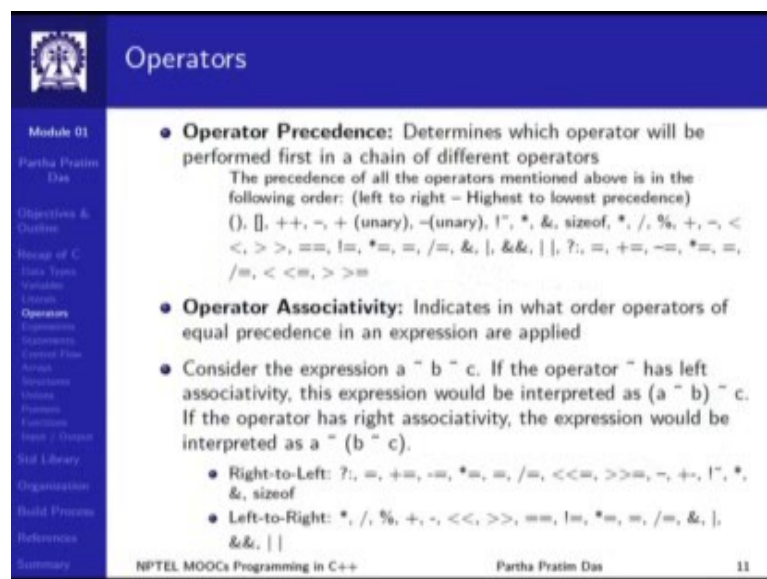
**Operators**

- An operator denotes a specific operation. C has the following types of operators:
  - Arithmetic Operators: `+` `-` `*` `/` `%` `++` `--`
  - Relational Operators: `==` `!=` `>` `<` `>=` `<=`
  - Logical Operators: `&&` `||` `!`
  - Bit-wise Operators: `&` `|` `~` `<<` `>>`
  - Assignment Operators: `=` `+=` `-=` `*=` `/=` `...`
  - Miscellaneous Operators: `.` `.` `sizeof` `&` `*` `?:`
- **Arity of Operators:** Number of operand(s) for an operator
  - `+`, `-`, `*`, `&` operators can be *unary* (1 operand) or *binary* (2 operands)
  - `==`, `!=`, `>`, `<`, `>=`, `<=`, `&&`, `||`, `+=`, `-=`, `*=`, `/=`, `&`, `|`, `<<`, `>>` can work only as *binary* (2 operands) operators
  - `sizeof` `!` `~` `++` `--` can work only as *unary* (1 operand) operators
  - `?:` works as *ternary* (3 operands) operator. The condition is the first operand and the *if true* logic and *if false* logic correspond to the other two operands.

NPTEL MOOCs Programming in C++ Partha Pratim Das

There are several operators in C; you will be familiar with many of them. There are by common or binary operators like addition, subtraction, multiplication. There are unary operations like negation. There are even ternary operations like question mark, colon. Every operator has a fixed arity that is a number of operands that it takes, which could be 1, 2 or 3.

(Refer Slide Time: 09:46)



**Operators**

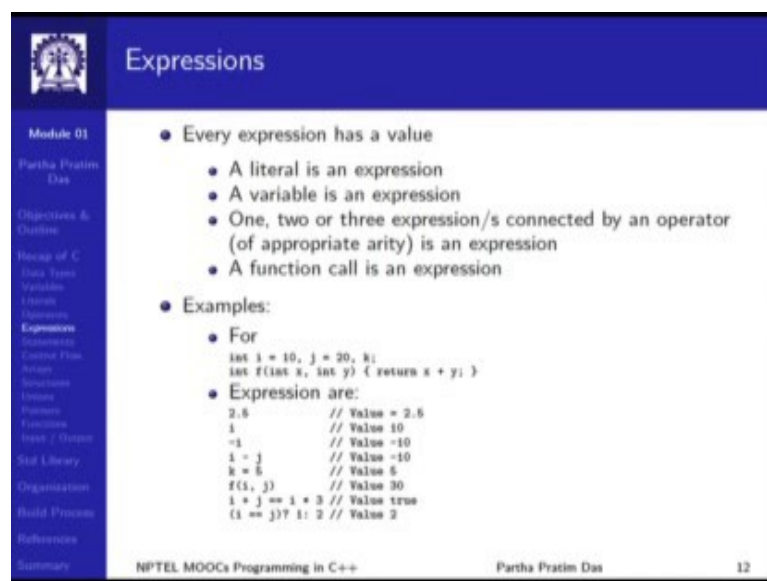
- **Operator Precedence:** Determines which operator will be performed first in a chain of different operators  
The precedence of all the operators mentioned above is in the following order: (left to right - Highest to lowest precedence)  
`()`, `[]`, `++`, `--`, `+` (unary), `-` (unary), `!`, `*`, `&`, `sizeof`, `*`, `/`, `%`, `+`, `<`, `>`, `>=`, `!=`, `*=`, `=`, `/=`, `&`, `|`, `&&`, `||`, `?:`, `=`, `+=`, `-=`, `*=`, `/=`, `<<`, `>>`
- **Operator Associativity:** Indicates in what order operators of equal precedence in an expression are applied
- Consider the expression `a ~ b ~ c`. If the operator `~` has left associativity, this expression would be interpreted as `(a ~ b) ~ c`. If the operator has right associativity, the expression would be interpreted as `a ~ (b ~ c)`.
  - Right-to-Left: `?:`, `=`, `+=`, `-=`, `*=`, `/=`, `<<=`, `>>=`, `+`, `+`, `!`, `*`, `&`, `sizeof`
  - Left-to-Right: `*`, `/`, `%`, `+`, `-`, `<<`, `>>`, `==`, `!=`, `*=`, `=`, `/=`, `&`, `|`, `&&`, `||`

NPTEL MOOCs Programming in C++ Partha Pratim Das 11

The operator in an expression is evaluated according to their order of precedence. Some operators have higher precedence, some have lower precedence. So, we know that if in the same expression there is multiplication as well as addition; multiplication has to be done earlier wherever it occurs in the expression.

Similarly, if there are more than one of the same operator in an expression then the order of their evaluation will depend on the associativity and some operators are left to right, some operators are right to left. So, here I have shown the different examples. This is just for your reference, you will certainly, if you know this. If you do not, please look up the text to understand this better.

(Refer Slide Time: 10:30)



**Expressions**

- Every expression has a value
  - A literal is an expression
  - A variable is an expression
  - One, two or three expression/s connected by an operator (of appropriate arity) is an expression
  - A function call is an expression
- Examples:
  - For

```
int i = 10, j = 20, k;  
int f(int x, int y) { return x + y; }
```
  - Expression are:

2.5	// Value = 2.5
i	// Value 10
-i	// Value -10
i + j	// Value 30
k = 5;	// Value 5
f(i, j)	// Value 30
i + j == i + 3	// Value true
(i == j) ? i : 2	// Value 2

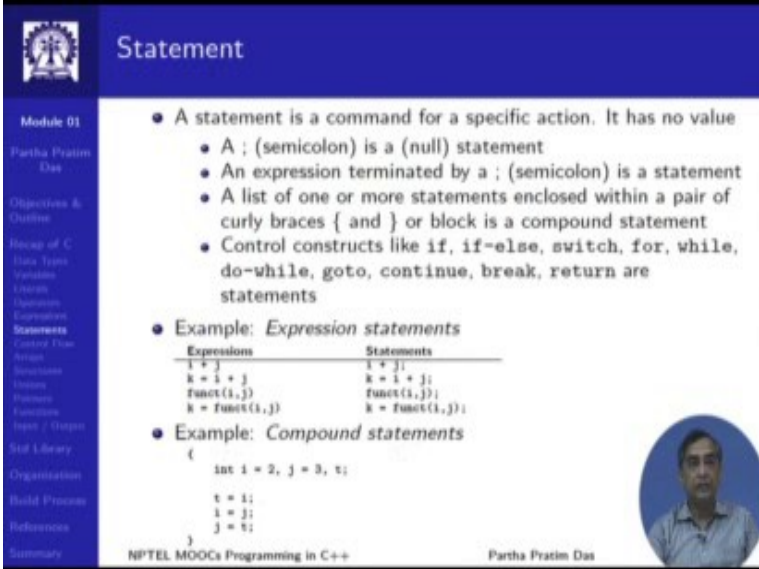
NPTEL MOOCs Programming in C++ Partha Pratim Das 12

Now, the next concept in C is an expression. That I have variables and I have operators. I have literals with those I can build expressions. So, expressions are defined in kind of a recursive form will say that every literal is an expression.

If I say number 5, it is an expression by itself. Every variable is an expression and if I have two expressions and connect them by a binary operator then that becomes a new expression. Similarly, I can have expressions with unary operator, ternary operator and so on. Any function call that is done is an expression. So, the basic point is that expression

must have value; anything that has a value in C is called an expression. So, there are different examples you can see here for the variables given here and different expressions are given below.

(Refer Slide Time: 11:34)



**Statement**

- A statement is a command for a specific action. It has no value
  - A ; (semicolon) is a (null) statement
  - An expression terminated by a ; (semicolon) is a statement
  - A list of one or more statements enclosed within a pair of curly braces { and } or block is a compound statement
  - Control constructs like if, if-else, switch, for, while, do-while, goto, continue, break, return are statements
- Example: *Expression statements*

Expressions	Statements
$i + j$	$i + j ;$
$k = i + j$	$k = i + j ;$
$\text{func}(i, j)$	$\text{func}(i, j) ;$
$k = \text{func}(i, j)$	$k = \text{func}(i, j) ;$
- Example: *Compound statements*

```

{
    int i = 2, j = 3, k;

    k = i;
    i = j;
    j = k;
}

```

NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, expressions cannot exist in C by themselves. So, expressions will have to exist as statement. A statement is a smallest unit of command that you can specify in a C program. So, the simplest or the smallest statement that you can have which is called a null statement; is a semicolon itself. Otherwise, if you have an expression, you can terminate that with a semicolon and once you terminate it with a semicolon then it becomes a statement.

So, if you look at the example below, in the expression statement ' $i + j$ ' is an expression because ' $i$ ' and ' $j$ ' are variables and  $+$  is an operator connecting them, but the moment you write ' $i + j ;$ ', it becomes a statement. It can occur independently anywhere, similar examples are shown for assignment for function call and so on.

Besides the expression statement, C has a number of control statements or control constructs, which basically allow the control flow in the program to be managed. So, there are selection statements and loop statements and so on. We will see little bit more

of them in the next slide and if there are number of statements one after the other which need to be grouped for use, and then we put a pair of curly braces around them. We say it becomes a block and such a statement is called a compound statement. So, that whole block of statements is a compound one you can see an example at the bottom.

(Refer Slide Time: 13:19)

**Control Constructs**

- These statements control the flow based on conditions:
  - Selection-statement:** if, if-else, switch
  - Labeled-statement:** Statements labeled with identifier, case, or default
  - Iteration-statement:** for, while, do-while
  - Jump-statement:** goto, continue, break, return
- Examples:
 

<pre>if (a &lt; b) {     int t;     t = a;     a = b;     b = t; }</pre>	<pre>if (x &lt; 5)     x = x + 1; else {     x = x + 2;     ++y; }</pre>	<pre>switch (i) {     case 1: x = 5;             break;     case 3: x = 10;             break;     default: x = 15; }</pre>
<pre>int sum = 0; for(i = 0; i &lt; 5; ++i) {     int j = i * i;     sum += j; }</pre>	<pre>while (n) {     sum += n;     if (sum &gt; 20)         break;     ++n; }</pre>	<pre>int f(int x) {     return }</pre>

NIPTEL MOOCs Programming in C++ Partha Pracin Das

Now, coming to control constructs which are the key area of a C program which basically tell you, how the execution of the program can happen. We have different ways to control, what will be executed after one statement has been executed. By default we say, the C program has a fall through control, which means that once a statement has been executed then the immediately next statement in the program code will be the next statement to be executed, but we can change that by the control flow.

So, the first kind of control flow is a selection statement; 'if' or 'if else'. So, in the example as you can see that we are saying, if ( $a < b$ ), then if that is true then you do the compound statement that follows it. You can easily understand that what the compound statement is saying that you interchange the value of 'a' and 'b' by using a third variable.

If you look at the next example of 'if', it is showing 'if else' kind of statement, where  $\text{if}(x < 5)$  that if the condition is true, it does one statement, else if the condition is false

then it does another statement. You can see on that the false part has a compound statement, whereas the true part as a single statement and selection can be done for a multi way. In their multi way form, that you can use the value of a variable.

In this case we have used the variable 'i' and you can switch on that depending on what value the variable has taken, you take any one of the cases that are listed. So, if 'i' is one then case one will be selected by which 'x' will become 6 and we have a default case which is executed, if the value of 'i' does not fall among the different cases that exist. Statements like 'case' as we have shown in 'switch' are also called labeled statement because there is a label to them.

Then we have iteration statements where you can repeat or loop statements very commonly these are called loop statements, where you can have a 'for' loop which has three parts. An initial part 'i' assigned '0', which is initially done. A second condition part which is checked every time the loop is executed and you continue in the loop provided that condition remains true and there is a body which is basically what follows the 'for' statement, which is the sequence of instructions or statements to be executed as a part of the loop and there is an end of loop statement like '++ i'.

Similarly, we have 'while' loop we can have do while iteration and the final type of control statements are 'go to', 'continue', 'break' and 'return'. As you know, C advises that you should not use 'go to'. So, we are not showing example of 'go to'. If you design a C program well then you will not have any reason to use the 'go to' at all. So, try to only use 'continue' and 'break' along with loop and different 'switch' statements to achieve your control flow, but you will need 'return' to return from a function. So, these are the four types of different types of control constructs that exist.

To sum up, what we have seen in this module so far, we have seen what are the basic components of a C program, which is how do you do a IO, how you; using the data type, how you define variables? How you initialize them? How to form them into expression using operators? How to convert the expressions into statements and different control flow statements to control the flow of the program?

So with this, we will end this part and next we will talk about the derived types and how to use the derived types in C.