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

Lecture - 08
Constants and Inline Functions

Welcome to module 6 of Programming in C++. In the first five modules, we have recapitulated the C Programming language, C Standard library and different constructs of the C language itself. We have also taken several examples starting from elementary input output, arithmetic operation, loop kind of examples to use of arrays, strings and specifically data structure, we took example of char to show how programming in C++ and judicious use of the C++ standard library can make programming in C++ really easier more efficient and less error prone.

From, this module onwards we will now start getting into the C++ programming language discussing different features. In the next few modules, we will specifically deal with a set of features which are commonly called as Better C Features that is these features do not make use of the core paradigm of object orientation that exists in C++. But these are procedural extensions to the C language which are required for making the object oriented programming possible and are as such nice features to have they could have been in C also incidentally they were not thought off when C was designed. And the interesting part is that some of this features after they were introduced in C++, and we are going to discuss one of those features right in this module. Some of these features have been later on taken into C programming and are now available in the C 99 standard. We start this module 6, where we will discuss about constants and inline functions.

(Refer slide Time: 02:36)

Module Objectives

- Understand const in C++ and contrast with Manifest Constants
- Understand inline in C++ and contrast with Macros

NPTEL MOOCs Programming in C++ Partha Pratim Das 2

So, we will try to understand const in C++ and contrast that with the similar concept not exactly the same concept, but similar concept of manifest constant in C and we will try to explain the inline functions in C++ and contrast them with macros.

(Refer slide Time: 03:01)

Module Outline

- const-ness and cv-qualifier
  - Notion of const
  - Advantages of const
    - Natural Constants – π, e
    - Program Constants – array size
    - Prefer const to #define
  - const and pointer
    - const-ness of pointer / pointee. How to decide?
  - Notion of volatile
- inline functions
  - Macros with params
    - Advantages
    - Disadvantages
  - Notion of inline functions
    - Advantages

NPTEL MOOCs Programming in C++ Partha Pratim Das 3

So, these are the topics that we will discuss, we will slowly unfold that you can see it on the left side of the screen.

(Refer slide Time: 03:09)

So let us start with the Manifest Constants in C. All of us know that we can define a constant value or a fixed value using any literal or using an expression, if we write hash define followed by a name and then followed by the particular expression that we want to define.

(Refer slide Time: 03:49)

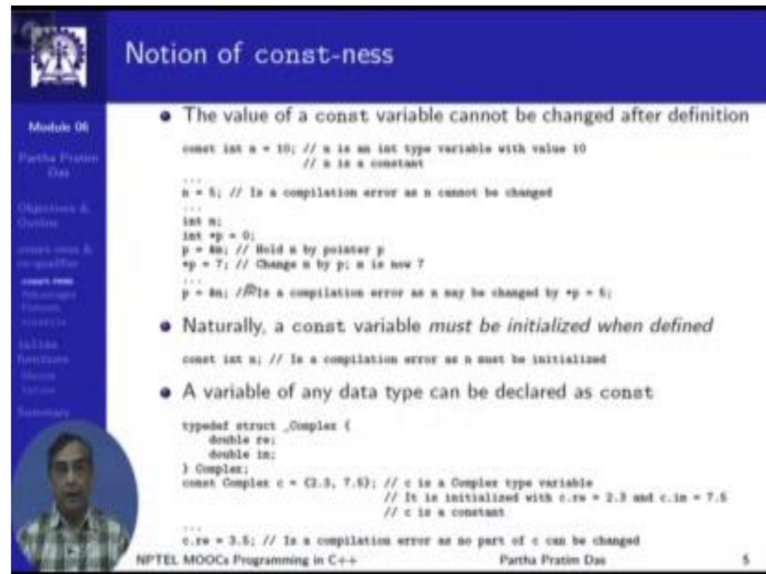
So, on left we can see examples of this in terms of what we have here; see two is a defined to be the value 2. Similarly, we have shown how to define pi so which is defined in terms of an expression. Pi is a ten 1.0 is pi by 4 so if you multiply it by 4 you get the value of pi. Then we use them in the expression here to compute the perimeter of a circle. This is a program which is a very commonly used in C and many of you have written this earlier.

Now, let us look into this program little bit differently, let us look on the right hand side. The hash define that we had here of two finally gets replaced at this point by the C preprocessor. So, before the program goes into compilation this line is removed and wherever t w o, this symbol had occurred, earlier as in here as in here will get replaced by whatever I have been defined that symbol to be. So, you can see that pi has been replaced by this whole expression in this and this is the code which actually goes for compilation to the C compiler.

This is the behind the scene scenario and we do not normally unless you put special options in your compiler you would not be able to see this version of the program, where just this hash defines has been replaced. What is the purpose of the hash define is to is simply to give the symbol and the expression as equivalent names and C preprocessor

can do the replacement. This is just to make you understand, what is the scenario of a manifest constant?

(Refer slide Time: 06:07)



So, what will be the consequence of this? The consequence is that I wanted to actually use a value which I wanted to treat as constant, but since I have got it replaced if I just again look into this and concentrate on the last line in the comment I wanted to use that as a constant and in the process the compiler actually never gets to know that they were as a variable called two or they were a symbol called t w o, the compiler sees that numerical 3 because it has been replaced. So to take care of this a Notion of Const-ness has been introduced.

(Refer slide Time: 07:02)

So, if we look into how const-ness is done, so you see we are doing a declaration where we prefix the declaration of n by a new keyword const. If I just write int n initialize 10 we know n is an integer type of variable whose initial value is 10. We are prefixing it with this const keyword, if we do that what it means is initial value of n is 10 and it also says that it cannot be changed in future, that is I cannot by any assignment or by any other means I can change n, n will remain to be 10 all through the program. So, if I try to do something like this here which is in assigned 5 and try to compile that code, the compiler will give an error will say that n is constant it cannot be changed.

I can try to bypass that and do something like this as in here, as usually if I had another variable m and a pointer p which is a integer type pointer I take the address of m and p I can certainly use the pointer to change the value of m, if I do assign seven to star p what it means that it actually changes m. But, if I try to do the same thing here, if I try to change this value of n by assigning the address of n into the pointed variable p and subsequently, possibly I can do star p assigned 5 I will not be allowed to do that. So, you may be little bit surprised that if we define a variable to be const and then try to use a pointer and take its address even that is given to be a compilation error. And the reason it is a compilation error is if this is not an error then you will be able to do this, which is in

the violation of the principle of const-ness that we are trying to define, that we are trying to say that n cannot be changed.

(Refer slide Time: 09:24)

Notion of const-ness

- The value of a const variable cannot be changed after definition
- Naturally, a const variable *must be initialized when defined*
- A variable of any data type can be declared as const

What is a consequence of that? The next natural consequence of that is a const variable must be initialized. As soon as it is getting defined it must be initialized, because if you do not initialize it then there is no way to change its value so whatever garbage value it has that garbage only will. So, if you declare a const variable without initialization that will become a compilation error

We can also declare variables of different types as const here is an example of using the struct type to variable which is a complex number say and we can define that to be constant which will mean that with this const-ness you will no more be able to change the value of the variable of a component say c dot re. C dot re by definition has become 2.3 because we have initialized and because we have said that c is const, if c is const then whole of it is const I cannot change any of the component. So, if I try to assign 3.5 to c dot re it will be a compilation error. This is the notion of the const-ness.

(Refer slide Time: 10:37)

So let us see how we use it. So, let us now put two programs side by side on the left, the typical C program which uses hash define and on the right we write an equivalent program in C++ which makes use of const-ness to achieve the same purpose. Earlier we were writing hash define two to the value 2, now we are saying that two is a variable of type integer which is initialized with two, but it is a const variable so you cannot change it. The major consequence of this is when this program on the right hand side, when this program gets true the C preprocessor certainly said it is no hash define, so that symbol two will not get replaced at this point.

Similarly, the symbol pi will stay and the compiler will get to see that these are the different variables that exist in the program and the compiler knows that these are constant these cannot be changed. So you can achieve the same purpose that you had in C and you get the added advantage that now the compiler can see all of these and I compiler would know what is the type of two, compiler would know what is the type of pi or for that matter any variable, any value that you define to be constant using the const keyword.

(Refer slide Time: 12:24)

So you get a lot of advantages in terms of using that. There are two major contexts in which you would like to use constant values; one context is, when you deal with different natural constants like pi like e like the golden ratio phi the boolean truth values false value null value and so on. There are several natural constants that occur in the program certainly you can always define them with const with that they will have their value, they will have their type, and they will have their basic property that natural constants naturally you cannot change the value of pi or you cannot change the value of e, so that property will also be retained.

In addition another place where we would frequently use constant is where something is constant for my program or something is constant for a particular function. So, for that we will use the second set of definitions like we can have an array size defined to be a constant we can have number of elements defined to be a constant. These are not universal natural constants, but these are constants for my function. If I have done that then the advantage that we get is when we write the program, we can write them in terms of these variables so that later on if we have to change them we can just change the initialization of the constant which is their possibly at the top of the program or in some header file.

There is one added advantage of doing this, if you do hash define, the hash define has a scope over the whole file. If I hash define some value n to a certain specific constant value then wherever I have n in my program that gets replaced by this hash define value. But const is a variable declaration so it can be done in any scope I can do it within a function, I can do it within a block within a function and like any variable declaration the variable declaration of const will also remain limited within that scope, so it is possible that I have the same variable n in the same file occurring in two different functions both in both places is this constant but it has different values, you cannot achieve this kind of effect with hash define.

(Refer slide Time: 15:07)

Advantages of const

- Prefer const over #define

Using #define Manifest Constant	Using const Constant Variable
• Is not type safe • Replaced textually by CPP • Cannot be *watched* in debugger • Evaluated as many times as replaced	• Has its type • Visible to the compiler • Can be *watched* in debugger • Evaluated only on initialization

NPTEL MOOCs Programming in C++ Partha Pratim Das 8

We summarize that we prefer const over hash define because it is not safe in terms of type it is replaced by CPP where as const is not. So, if you are using a debugger you will not be able to see the hash define symbols in the debugger with const you will be able to see that. The other side effect is since the hash define replaces the expression at every point it needs to be evaluated as many times as it is replaced, where as in case of const it is evaluated only at the initialization point. So, const certainly has a complete advantage over the hash define.

(Refer slide Time: 15:46)

Now, let us see some consequences of defining const particularly, we will look at the const-ness of pointed type data. In a pointed type data we know that we have a pointer and it points to a variable. So the question is, if we talk about const-ness then whose const-ness are we talking about, are you talking about the const-ness of the pointer or the const-ness of the pointed data. Here, we talk about two things pointer to constant data whether data is constant, but the pointer is not or the pointer itself is constant, but the data may or may not be constant.

Below here I just show an typical example of how we compute with pointer and pointee we have defined two variables, so we have a pointer which takes the, if you just look in here it takes the address of one variable and then using that I can directly change the variable or I could I can change it through the pointer. Similarly at this line, earlier the pointer was pointing to n now it has been changed to point to m and they can again use it to change the value of m. This is the typical use of a pointer-pointee scenario.

(Refer slide Time: 17:23)

So, with this we would next like to discuss as to how we can control these changes with the use of const.

(Refer slide Time: 17:34)

So first is, if I have a pointer to a constant data. So, what we are doing here, is we have written const before the data before the type of the value that the pointer points to. If I

write the const at this point it means that the pointed data is constant, it cannot be changed. So, n has been defined to be a constant value. We already know an attempt to change that value of n is an error because if n is constant, and we have defined p to be a pointer to n. So, trying to change the value of n using p that is star p assigned 7 is also an error. But, p itself is not a constant, that is if I want now I can make p point to some other variable, so m is a variable here which is not a constant variable I can make p point to m and then I can use this star p assigned 8 to change the value of m, m will now become 8, it was 4, it will now become 8.

(Refer slide Time: 18:55)

So, now if you will look into, if I have a variable which is not a constant say int has been defined to be a integer type variable initialized with 5 and I have a pointer p which points to a constant type of integer value and I put the address of n into p. Now, naturally aim assignment of 6 to n is valid because n itself is not a constant. It is also that star p assigned 6 is valid, but if I try to do star p assigned 6 that is not valid because p says that am pointing to a constant integer. So very interesting scenario because I have a p here which is pointing to n. P knows, this knows that if I write star p is constant. That is p cannot be used to change this value, but n by itself is not constant. So, n can be changed.

Now, this is valid because what you are saying is you are saying more than what is required, you are saying that n by itself can change. So, whether I change it directly as n or I change it through a pointer it does not make a difference because n can change, but the pointer as said that I am restricted not to change. The pointer is said that if you go through me then I will not allow you to change the value. So, here this is a scenario where the variable actually can change, but the pointer gives a view which does not allow you to change that, but if I directly go or if I use some other pointer which does not point a constant value then we will be able to change that. Finally, if we try to do the reverse that is if I have a constant variable n and if I try to use a pointer to a non constant value p then however I will not be able to do this.

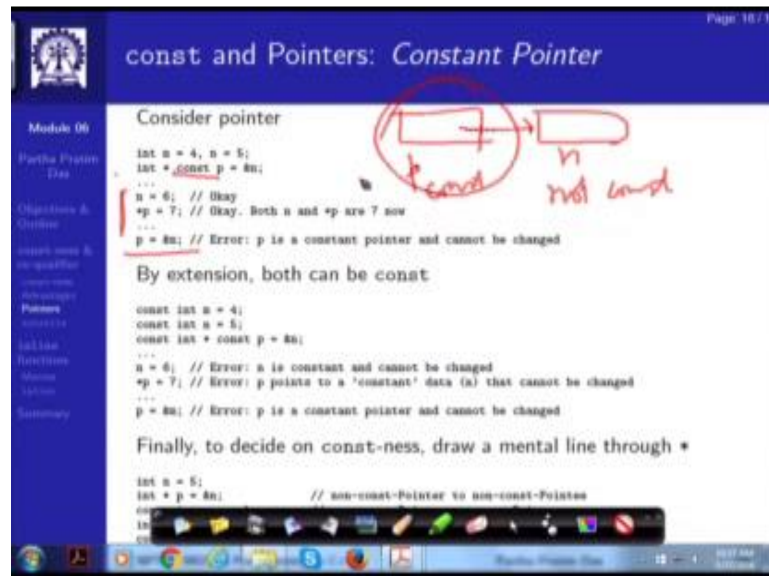
(Refer slide Time: 21:46)

So, if we just try to illustrate then the last int case here, so we are talking about this case if I have a p which points to n, where this is constant and star p is not constant then we have a error quite validly, because n is a constant. If star p is not constant, star p is trying to point to n then I can always make use of star p here to change the value of n which am not supposed to do.

So what we learned here is a basic notion that if a value is a not a constant I can still use it pointed to a constant to view that, but I will not be able to change it to that pointer. But

if a value is constant then I cannot use a pointer which is a pointed to a non constant value I will not even be allowed to initialize that const pointer with the address of this constant variable because that would violate the basic principle of const-ness.

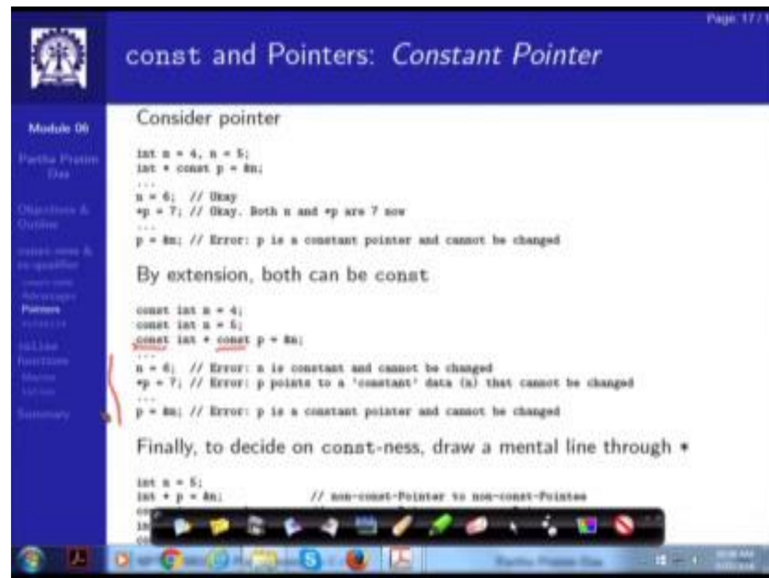
(Refer slide Time: 23:20)



Next, let us look at the const-ness of the other side, what if the pointer is constant? So, if you look in here this is where we are, we have slightly shifted the position where we had written the const. Earlier the const was written here at this point, now the const is written after the star symbol this says that the pointer is constant, but the value that it is pointing to is not constant. If I draw it p is const n not const.

So, what it means that if I can easily write this n is not const, so I can change its value, since n is not const I can use p dereference it assign seven to star p that would change the value of n which is valid because I am not violating anything, but I will not be able to do is the last one that is I cannot change the address that is stored in p I cannot make p now point to a new variable m, because I have said that the pointer itself is constant this side is constant now that is earlier the other side was constant. Naturally, if we have this then by extension we can also combine both of this that both the pointer and the data it is pointing to can be constant.

(Refer slide Time: 24:55)



So, we here we are showing an example where I write const on both sides which means that p is a constant pointer to a constant data, which means neither p can be made to point to any other variable other than n nor I can use p to change the value of n. So, all of this will now become error.

Now, at the end certainly since we are writing since we are writing since we are writing a const-ness on the pointed data or the pointer itself it is confusing at times as to where should I write the const and what will become const by putting the const keyword. The thumb rule is very simple, that when you have this declaration look at the star symbol in the whole declaration.

(Refer slide Time: 26:09)

Mentally draw a vertical line so if you are trying to do this you are saying const int star p etcetera, draw a vertical line through this star symbol and see which side the const keyword appears, this is your data side and this is your pointed side. So if the const-ness is on the data side then whatever you are pointing to is constant. In contrast if you have int star const p etcetera, so this is this const is on the pointer side so the pointer is constant. So that is the basic thumb rule by which you can decide which of them is a constant.

(Refer slide Time: 26:57)

The examples are given below. So, you can use this and for string this is an example that I have worked out you could read it carefully and try to understand a string is given and if we just have a string then you can two ways change that either you can edit the string or you change the string itself. On the code on top we show the effect of editing the string or changing the whole string altogether.

(Refer slide Time: 27:34)

But you can stop that if you do something like, put a const here, if you put a const here then the string itself becomes constant, so you cannot change any character of the string. So, you cannot like in here you could write assign n to the first symbol you cannot do that anymore. Whereas, if you put the const on this side then you can now change every any of the symbols in the string, but you cannot change the string as a whole.

Here you could change the string now you cannot change the string because that means changing the pointer. And certainly you could protect both the edit as well as the changing of the name if you put const on both sides of the pointer that is, if you have a constant char star pointer pointing to a constant array of characters then neither it can be edited nor it can be changed. This is an example to show how const-ness applies on both sides. So, we have discussed the basic notion of const-ness and illustrated how const-ness applies in terms of the pointers.