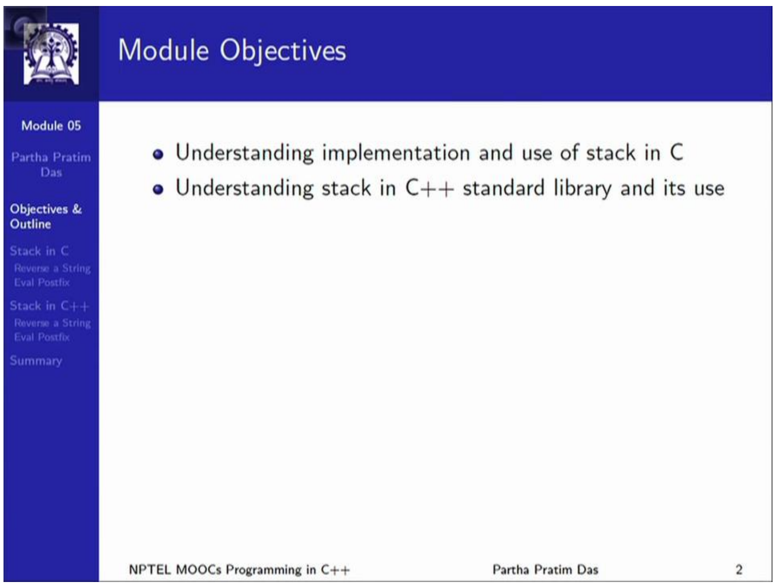
Programming in C++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 07
Stack and Its Applications

Welcome to module 5 of Programming in C++. We have been discussing various example programs that typically you have written in C, and now we are showing how they can be written equivalently in C++. How often the C++ standard library can be used for it and with that, we would like to show, how C++ enhances the ease of programming. So, in this module we will talk about stacks and its applications.

(Refer Slide Time: 00:58)



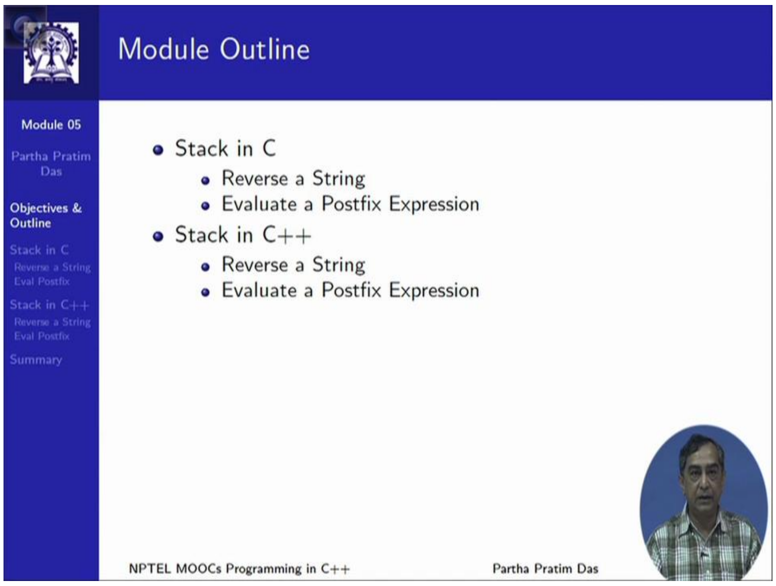
Module Objectives

- Understanding implementation and use of stack in C
- Understanding stack in C++ standard library and its use

Module 05
Partha Pratim Das
Objectives & Outline
Stack in C
Reverse a String
Eval Postfix
Stack in C++
Reverse a String
Eval Postfix
Summary
NPTEL MOOCs Programming in C++
Partha Pratim Das
2

We will try to understand implementation and use of stack in C, which I presume all of you know and then we will show, how in C++ the standard library can be used for doing stacks.

(Refer Slide Time: 01:10)



Module Outline

Module 05

Partha Pratim Das

Objectives & Outline

Stack in C

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

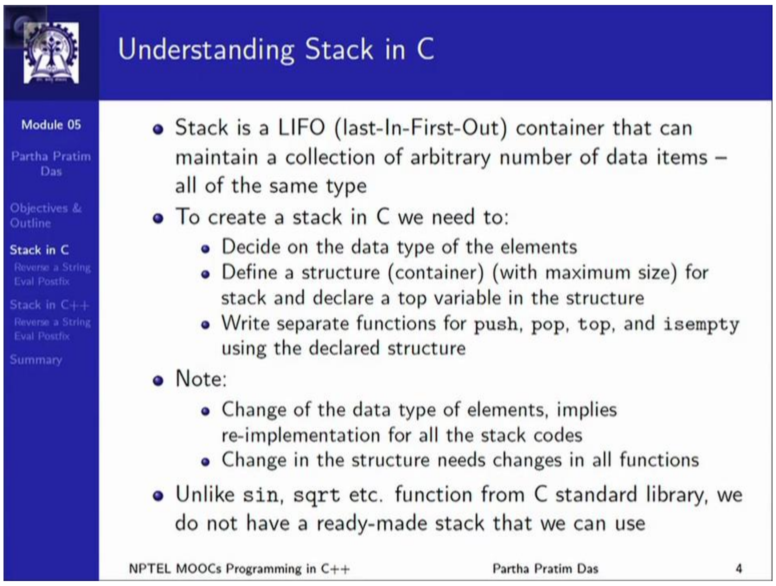
Summary

- Stack in C
  - Reverse a String
  - Evaluate a Postfix Expression
- Stack in C++
  - Reverse a String
  - Evaluate a Postfix Expression

NPTEL MOOCs Programming in C++ Partha Pratim Das

So, these are the topics we will primarily use reversing a sting and evaluating a postfix expression as examples.

(Refer Slide Time: 01:22)



Understanding Stack in C

Module 05

Partha Pratim Das

Objectives & Outline

Stack in C

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

Summary

- Stack is a LIFO (last-In-First-Out) container that can maintain a collection of arbitrary number of data items – all of the same type
- To create a stack in C we need to:
  - Decide on the data type of the elements
  - Define a structure (container) (with maximum size) for stack and declare a top variable in the structure
  - Write separate functions for `push`, `pop`, `top`, and `isempty` using the declared structure
- Note:
  - Change of the data type of elements, implies re-implementation for all the stack codes
  - Change in the structure needs changes in all functions
- Unlike `sin`, `sqrt` etc. function from C standard library, we do not have a ready-made stack that we can use

NPTEL MOOCs Programming in C++ Partha Pratim Das 4

Just to recap, on the Introduction of Stack. Stack is a LIFO structure; last-in-first-out container that maintain a collection of arbitrary number of data items. There is no restriction like in array; there is a restriction of how many elements I can keep depending on the size of the arrays. Stack by definition unbounded, all elements data elements that keep in a stack have to be of the same type, but I can keep any number of elements. So, to create a stack in C, we need to do; these are the common steps that we will need to follow.

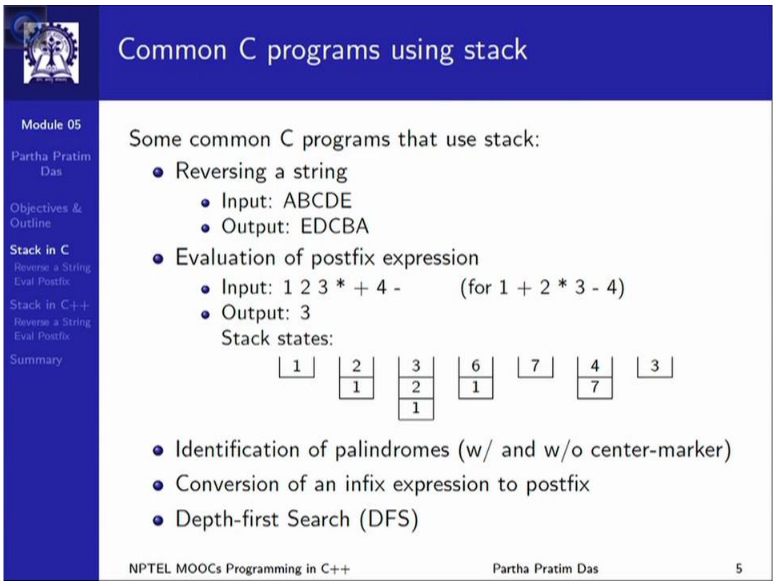
First we need to decide on the data type of the element, define a structure or container, for actually defining that in C we will need to use some maximum size, otherwise the C compiler is not allow us to define it, declare a top variable which will maintain the top of the stack and then we will need to write the four functions that are typically required for the operations of that stack.

The function push, which adds an element to the stack; function pop, which removes the top most elements, the element last added; function top, which gives up me the top most element without removing it and finally, we will need to have a function isempty or empty, which will tell me whether there is at all any elements in the stack. So, if isempty is true then operations like pop and top will be invalid because there is no top element to remove or to return.

So, we can note that as we change the type of elements for a stack, we need to re-implement, rewrite the whole code of the stack that is the code of the stack for stack of integers, and the code of the stack for stack of strings would be different, though the basic notion of the LIFO will be valid in both cases. So, unlike say sine, square root all these functions that we have in C standard library, we have taken at look earlier in math dot h, we have a lot of library functions. So, if I want to do compute sine or compute square root or compute arctan, I do not need to write that function I can just use it from the library.

In contrast, C standard library does not give me a mechanism to use a stack. It does not give me a stack which am I can use right away. So, this is the current state of affairs with the C programming. So, in this context, we would try to take a look as to how things can change if you use C++.

(Refer Slide Time: 04:15)



Module 05

Partha Pratim Das

Objectives & Outline

Stack in C
Reverse a String
Eval Postfix

Stack in C++
Reverse a String
Eval Postfix

Summary

Common C programs using stack

Some common C programs that use stack:

- Reversing a string
  - Input: ABCDE
  - Output: EDCBA
- Evaluation of postfix expression
  - Input: 1 2 3 * + 4 - (for 1 + 2 * 3 - 4)
  - Output: 3

  Stack states:
- Identification of palindromes (w/ and w/o center-marker)
- Conversion of an infix expression to postfix
- Depth-first Search (DFS)

NPTEL MOOCs Programming in C++ Partha Pratim Das 5

So, let us also take a look in some of the common problems that we tried to solve using a stack, given a string. I would like to reverse the string that is the left to right order I would like to change to right to left order is a typical program. Another is evaluation of a postfix expression as you all know typically we write expressions with infix notation where the operator happens in between the operands. So, I am showing an example 1 plus 2 star 3 minus 4, if I have to evaluate this expression though we read the expression from left to right, when we go to evaluate this we will not strictly be able to evaluate left to right.

We know that star the multiplication has a higher precedence compared to addition and subtraction. So, first we will have to multiply 2 with 3, get the result 6 and then we have a plus and a minus which both of which has a same precedence, but they are left associative. So, we will have to proceed from left to right and compute that in the result will turn out to be 3.

So, infix notation is very good in terms of how we write an expression. We have been taught from our school days to write these expression and follow the so called BODMAS rules to evaluate it, but it gets very difficult for evaluating a postfix expression because the order of the operators to be executed where is in between depending on their precedence associativity. So, we take resort to a different notation known as a postfix notation.

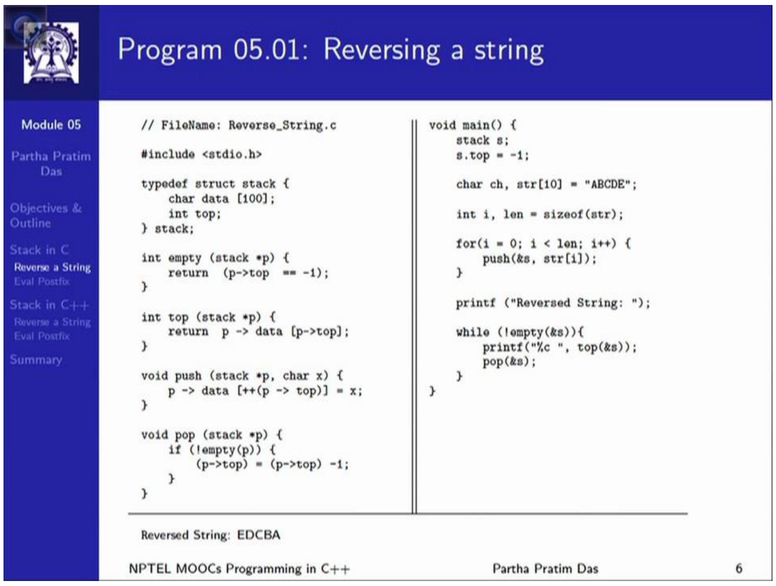
In the postfix notation, it is, first the operand has specified and then the operator comes that is why it is postfix. So, always if I have C an operator then the operands for that have already occurred. So, if the operator has an arity 2, then I know that 2 immediately preceding operands are to be applied for that operator. So, we show an example here where you can see that if I look into say this expression which is converted from the infix expression as given here, my expression turns out to be 1 2 3 star plus 4 minus and this also a problem as to how to convert the infix expression to a postfix expression. That also can be solved by using a stack, but here we are just showing that, if the postfix expression is given, how do we evaluate it using a stack.

So, these are the different stacks states all that we can keep on doing is we take; scan the expression from left to right, take a free operands, put it into the stack, keep on stacking till we come across an operator. So, here we push 1, then we push 2, then we push 3 and then we come across the operator star which by definition has arity 2. So, I know that it will require the last 2 operands, we will pop these 2 and 3 apply the operated to get 6 and push it back.

Then we get plus the next operator. So, we know that also needs two operands. So, we pop 6 and 1, the stack becomes empty, we operate it with plus to get 7 and push it back and we proceed in this way to get the result. So, this is a nice example to see that we need to proceed using the different functions of the stack to actually compute the postfix expression that is available to us.

Similarly, several other problems like identification of palindromes, which read the same from both sides, with or without a center marker that is a special character at the middle can be identified by using stack, infix expression can be convert it to postfix depth first search can be done and an examples are several. So, stack plays a significant role in the programming of any software system and therefore, if you can have convenient, reliable, robust mechanisms to use stack we will be strongly benefitted and then this is what C++ gives us as a readymade solution.

(Refer Slide Time: 08:57)



Program 05.01: Reversing a string

Module 05

Partha Pratim Das

Objectives & Outline

Stack in C

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

Summary

```
// FileName: Reverse_String.c

#include <stdio.h>

typedef struct stack {
    char data [100];
    int top;
} stack;

int empty (stack *p) {
    return  (p->top  == -1);
}

int top (stack *p) {
    return  p -> data [p->top];
}

void push (stack *p, char x) {
    p -> data [++(p -> top)] = x;
}

void pop (stack *p) {
    if (!empty(p)) {
        (p->top) = (p->top) -1;
    }
}
```

```
void main() {
    stack s;
    s.top = -1;

    char ch, str[10] = "ABCDE";

    int i, len = sizeof(str);

    for(i = 0; i < len; i++) {
        push(&s, str[i]);
    }

    printf ("Reversed String: ");

    while (!empty(&s)){
        printf("%c ", top(&s));
        pop(&s);
    }
}
```

```
Reversed String: EDCBA
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 6

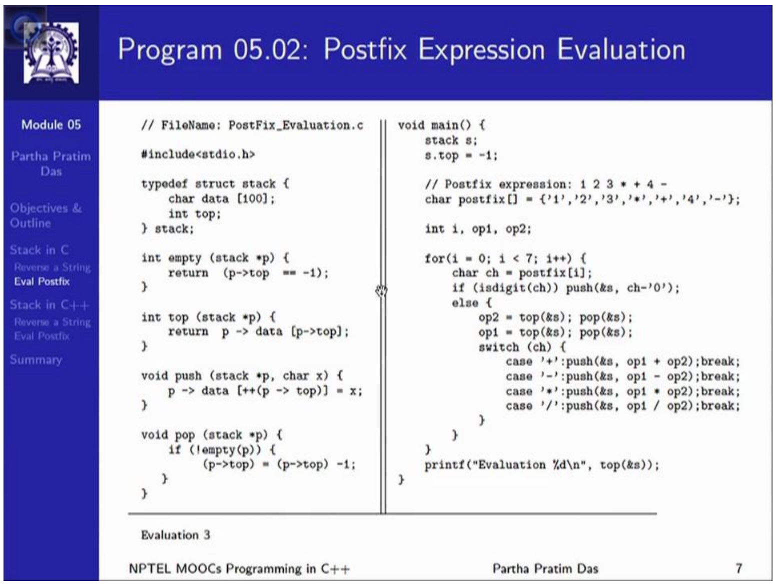
So, let us take a look at reversing a string. This is more of a recap for you, this is a C program which is trying to reverse the string as you can see the stack is defined as a structure, which has an array which is the container of the elements and a marker which is a top index which will keep the current index of the top most element, and these are the implementation of the four functions empty, top, bush and pop; empty just checks if the top value is minus 1, minus 1 designates that there is no element because are the minimum index value in the array could be 0. So, if it is minus 1 then we designate that the stack is empty.

The top simply returns the element from the top position; push increments the top position and puts the new element which is been given to be added to the stack on to the stack and pop simply decrements the top point at or the top index. So, that the element that was considered to be the topmost is not considered topmost anymore, element just below it is considered to be the topmost and since we cannot do this operation if the stack is already empty, it will be good to check if the stack is empty before we actually perform the pop operation.

Now, this is; we all know this will here using that if you have a string A B C D E here, we can just go in a for loop, add all these characters A B C D and E, one after the other into the stack and then if we keep on finding out the top element and popping that top element then certainly e has been added last.

So, that will come out first, D has been added just before that, so it will come out next and in result we will get the string that we are showing here, which is E D C B A and we all are familiar with this. So, this is the way if you have to do the reverse string program, then is not only that you have to write this main function or the function which uses the stack to reverse the string, you also need to write the code that is given on the left column that is the scenario in C.

(Refer Slide Time: 11:25)



Program 05.02: Postfix Expression Evaluation

Module 05

Partha Pratim Das

Objectives & Outline

Stack in C

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

Summary

```
// FileName: PostFix_Evaluation.c

#include<stdio.h>

typedef struct stack {
    char data [100];
    int top;
} stack;

int empty (stack *p) {
    return  (p->top  == -1);
}

int top (stack *p) {
    return  p -> data [p->top];
}

void push (stack *p, char x) {
    p -> data [++(p -> top)] = x;
}

void pop (stack *p) {
    if (!empty(p)) {
         (p->top) = (p->top) -1;
    }
}
```

```
void main() {
    stack s;
    s.top = -1;

    // Postfix expression: 1 2 3 * + 4 -
    char postfix[] = {'1','2','3','*','+','4','-'};

    int i, op1, op2;

    for(i = 0; i < 7; i++) {
        char ch = postfix[i];
        if (isdigit(ch)) push(&s, ch-'0');
        else {
            op2 = top(&s); pop(&s);
            op1 = top(&s); pop(&s);
            switch (ch) {
                case '+':push(&s, op1 + op2);break;
                case '-':push(&s, op1 - op2);break;
                case '*':push(&s, op1 * op2);break;
                case '/':push(&s, op1 / op2);break;
            }
        }
    }
    printf("Evaluation %d\n", top(&s));
}
```

Evaluation 3

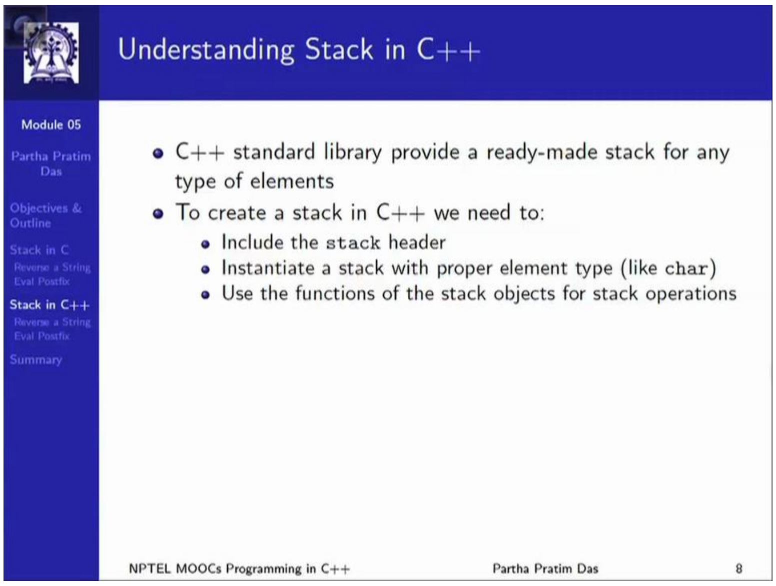
NPTEL MOOCs Programming in C++ Partha Pratim Das 7

A similar scenario is given for evaluating postfix expression, again in C left column is identical to the previous example, where the implementation of the stack is given and on the right column we show how we can use this to actually do the postfix evaluation. This is just the code of the algorithm I explained couple of slides back; you can go through it and understand that will actually, will do the job for us. It does the operation of finding if something is an operand which is done at this point, if you just look at particularly focused at this point.

Then you see that we are identifying if some the corrected that is given in the expression is operand or not. So, it is considered to be an operand if it is a numeral. So, in this expression we found out if each digit is ch, that is whether it is a digit can be an operand, single digit operand and then we push it otherwise we consider.

We know that this is an operator and the according to what that operator is it finds out and takes the topmost elements, pops them out and then computes the result and pushes it back. The general code of postfix expression would be much more complicated because you will have to also consider the arity of the operator and then do things based on that here we know all four operators, expected operators are binary. So, all those stacks are not here.

(Refer Slide Time: 13:04)



Understanding Stack in C++

Module 05

Partha Pratim Das

Objectives & Outline

Stack in C
Reverse a String
Eval Postfix

Stack in C++
Reverse a String
Eval Postfix

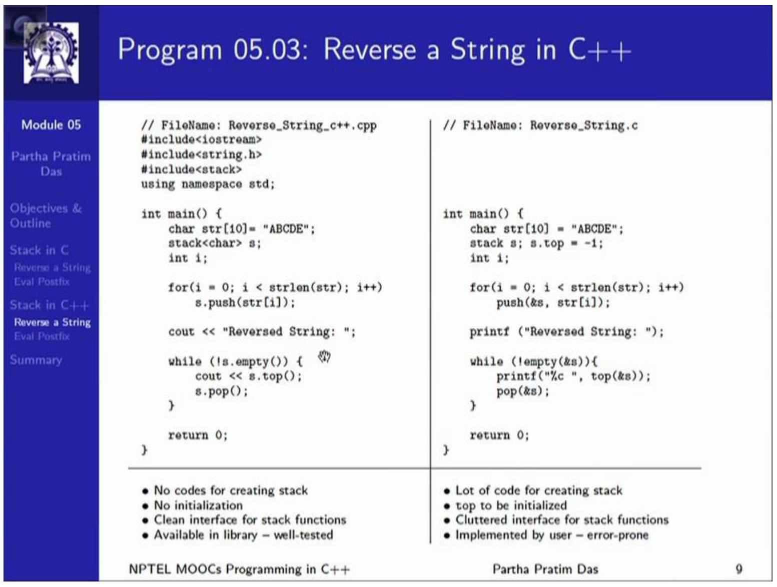
Summary

- C++ standard library provide a ready-made stack for any type of elements
- To create a stack in C++ we need to:
  - Include the `stack` header
  - Instantiate a stack with proper element type (like `char`)
  - Use the functions of the stack objects for stack operations

NPTEL MOOCs Programming in C++ Partha Pratim Das 8

Now, let us move on, so what will happen in C++. In C++, the good thing is the C++ standard library provides us with a readymade stack. So, to create that stack or to use that stack all we need to do is to include a new header called the stack header. Instantiate the stacks, which is defined in the stack header and then just start using the functions of that stack object and we will have the stack ready for us.

(Refer Slide Time: 13:37)



Program 05.03: Reverse a String in C++

Module 05

Partha Pratim Das

Objectives & Outline

Stack in C
Reverse a String
Eval Postfix

Stack in C++
Reverse a String
Eval Postfix

Summary

```
// FileName: Reverse_String_c++.cpp
#include<iostream>
#include<string.h>
#include<stack>
using namespace std;

int main() {
    char str[10]= "ABCDE";
    stack<char> s;
    int i;

    for(i = 0; i < strlen(str); i++)
        s.push(str[i]);

    cout << "Reversed String: ";

    while (!s.empty()) {
        cout << s.top();
        s.pop();
    }

    return 0;
}
```

```
// FileName: Reverse_String.c




int main() {
    char str[10] = "ABCDE";
    stack s; s.top = -1;
    int i;

    for(i = 0; i < strlen(str); i++)
        push(&s, str[i]);

    printf ("Reversed String: ");

    while (!empty(&s)){
        printf("%c ", top(&s));
        pop(&s);
    }

    return 0;
}
```

- No codes for creating stack
- No initialization
- Clean interface for stack functions
- Available in library – well-tested

- Lot of code for creating stack
- top to be initialized
- Cluttered interface for stack functions
- Implemented by user – error-prone

NPTEL MOOCs Programming in C++ Partha Pratim Das 9

So, here we are showing the reverse string example. Again on the right is the C program of course, the stack code which we have already discussed and not shown here is just the use of the stack in terms of doing the reversing of the string is shown, and an equivalent code is written on the left hand side. Here, please note that in the C++ code, we do not need to write the stack codes instead all that we are trying to do is we include a specific header. This is the header includes stack, this includes stack header. We have included, which includes the stack definitions and this is how we instantiate stack.

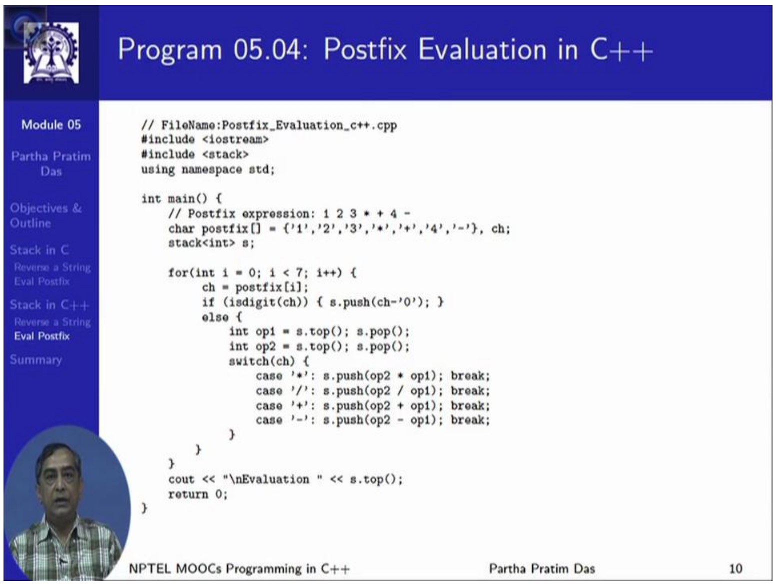
You had seen this notation earlier when we talked about vectors in contrasts to add is you had seen, if I have it have a vector then I could put the element type within the corner bracket similarly, if I have a stack within this angle brackets we are showing what is the type of the element that the stack will be made of and stack objects name will be s.

Now, we look into this, how we use this. Here in C, if you have to push we have to pass two parameters, we have say what is the stack which is the address ampersand s and what is the element that we want to push. Here, we simply do s dot, you had seen this notation also briefly in the context of vector, but we will talk about this lot more, but this is a typical notation; in the notation like what we use in structures, but somewhat differently interpreted s dot push which says that in stock s, I want to push and what do I want to push? I want push the s t r, that particular element.

Similarly, if I want to do this here, I want to do check for emptiness I do empty ampersand s. In contrast, here I will do s dot is empty as we can see here. So, you can see that, in terms of the code that use the stack the code is very similar and actually as we will understand the code is simpler to write because I do not have unnecessary parameters as in C, where I have to pass the parameter, the pointer to the stack at every point and I can just keep that outside of the function call and just pass the parameters that are needed which in case of push is one parameter; in case of the rest no parameters are required.

Finally, also note in C, while we define the stack; we also needed to initialize the top marker to minus 1. Here, somehow the system takes care of it the compiler takes care of it. So, I will not need to bother about initializing this top marker. So, in brief if we use the stack from the standard library then we do have a lot of advantage of not having to rewrite the stack codes every time we have a new type of data to deal with. These points are highlighted at the below here and instead, we have a well tested readymade stack available for this.

(Refer Slide Time: 17:07)



Program 05.04: Postfix Evaluation in C++

Module 05

Partha Pratim Das

Objectives & Outline

Stack in C

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

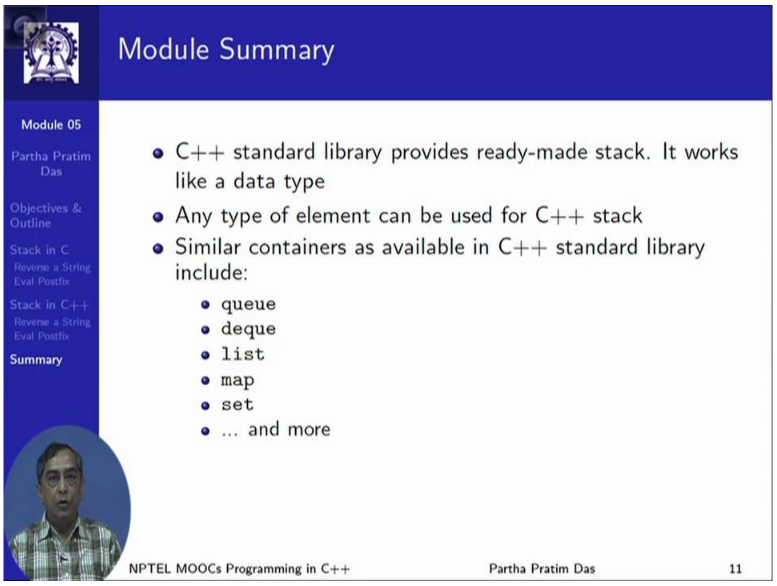
Summary

```
// FileName:Postfix_Evaluation_c++.cpp
#include <iostream>
#include <stack>
using namespace std;

int main() {
    // Postfix expression: 1 2 3 * + 4 -
    char postfix[] = {'1','2','3','*','+','4','-'}, ch;
    stack<int> s;

    for(int i = 0; i < 7; i++) {
         ch = postfix[i];
         if (isdigit(ch)) { s.push(ch-'0'); }
         else {
             int op1 = s.top(); s.pop();
             int op2 = s.top(); s.pop();
             switch(ch) {
                 case '*': s.push(op2 * op1); break;
                 case '/': s.push(op2 / op1); break;
                 case '+': s.push(op2 + op1); break;
                 case '-': s.push(op2 - op1); break;
             }
         }
    }
    cout << "\nEvaluation " << s.top();
    return 0;
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 10

I will just proceed, if you look into the next example. This is again similar to what we did in C, but now we are doing it in C++ for the postfix expression evaluation in postfix notation and this code is again very similar to the C code that we had written with the exception of having to include the stack header, define the stack with this notation and then all the stack functions are similarly written in a little bit different format, but the code otherwise is, as it were in C and with the advantage that we do not have to do anything in terms of the C stack code implementation.

(Refer Slide Time: 17:58)



Module Summary

Module 05

Partha Pratim Das

Objectives & Outline

Stack in C
Reverse a String
Eval Postfix

Stack in C++
Reverse a String
Eval Postfix

Summary

- C++ standard library provides ready-made stack. It works like a data type
- Any type of element can be used for C++ stack
- Similar containers as available in C++ standard library include:
  - `queue`
  - `deque`
  - `list`
  - `map`
  - `set`
  - ... and more

NPTEL MOOCs Programming in C++ Partha Pratim Das 11

So, in this module what we have shown is, if you are programming in C++ then you get a major advantage in terms of using stack. You do not need to implement the stack code; you do not need to even be concerned about, what should be the maximum size of the container for the stack. The C++ standard library stack automatically takes care of that. It takes care of how to initialize that top and it can be used for any type of element that I want the stack for, and the very interesting and exciting part of the whole C++ programming and standard library is that the story does not stop just with stack.

All common data structures which are frequently required like queue, which is first-in-first-out; deque, which is double ended queue, where you can add and remove at both ends of the sequence. The list; the singly linked list, the map, which is like name value pair; set, where the ordering is not important, you can just do union intersection in that. All of these data structures are available readymade in the C++ standard library. So, as we start programming in C++ even before we have understood the detailed nuances of the programming language, we would like to frequently take a look at the C++ standard library and write programs making use of all these data structures and make our programming easier and more robust to use.