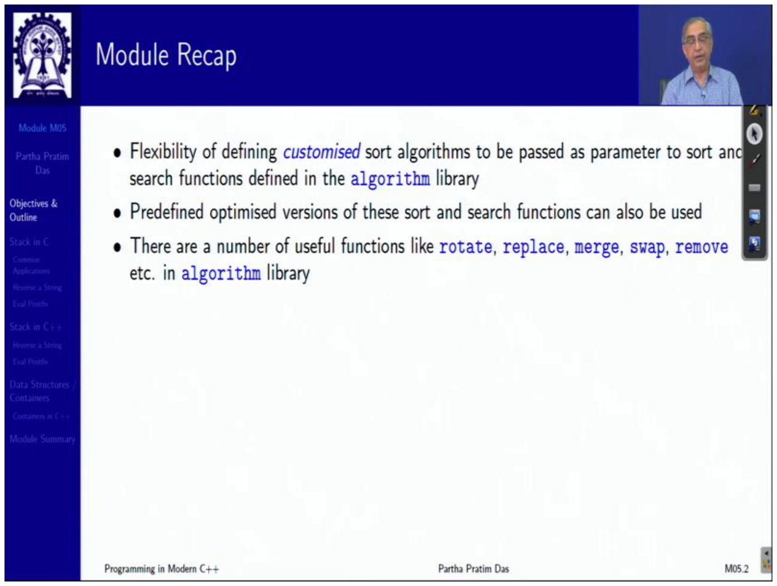
Programming in Modern C++
Professor. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No. 08
Stack and Common Data Structures/ Containers

Welcome to programming in modern C++. We are in week 1 and we are going to discuss module 5.

(Refer Slide Time: 00:36)

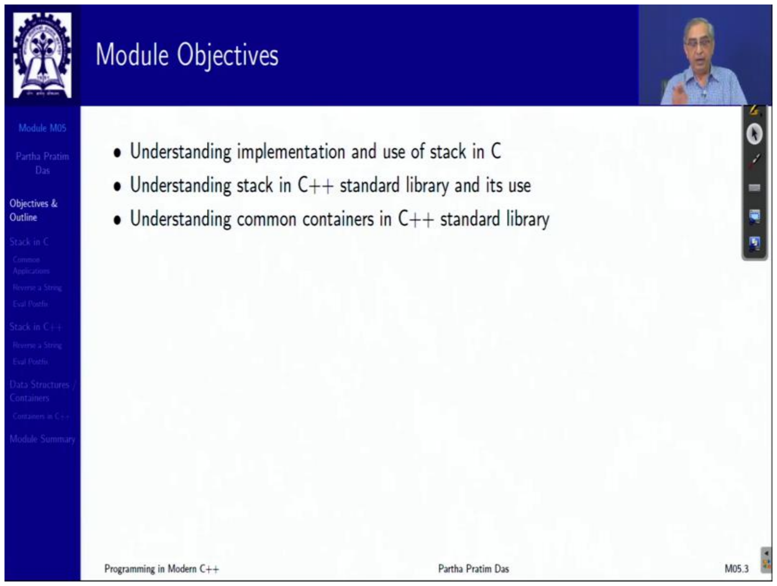


Module Recap

- Flexibility of defining *customised* sort algorithms to be passed as parameter to sort and search functions defined in the `algorithm` library
- Predefined optimised versions of these sort and search functions can also be used
- There are a number of useful functions like `rotate`, `replace`, `merge`, `swap`, `remove` etc. in `algorithm` library

In the last module, we have seen some of the common algorithms being available as a part of the C++ standard library and we have also seen how to use sorting and searching particularly to great benefit.

(Refer Slide Time: 00:55)


Module Objectives

Module M05

Partha Pratim Das

Objectives & Outline

Stack in C

Common Applications

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

Data Structures / Containers

Containers in C++

Module Summary

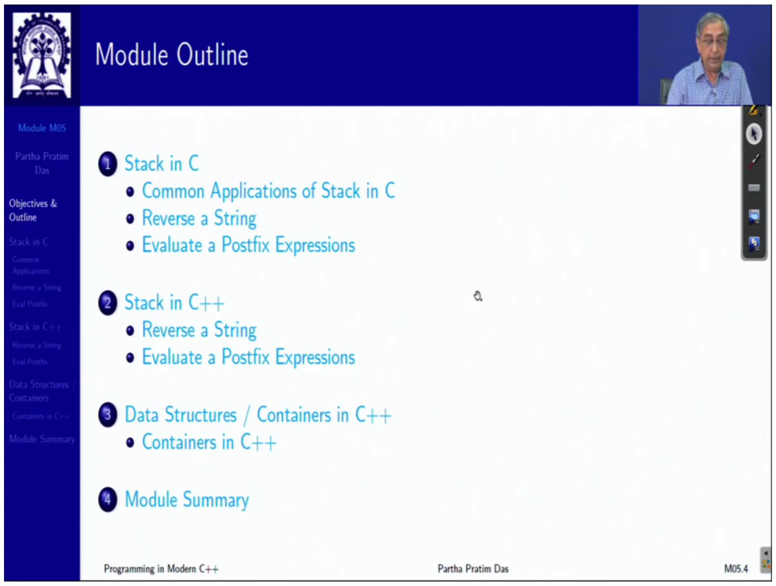
- Understanding implementation and use of stack in C
- Understanding stack in C++ standard library and its use
- Understanding common containers in C++ standard library

Programming in Modern C++ Partha Pratim Das M05.3

Now, as you all would know that in programming algorithm what comes closely with the algorithm are data structures, that is what we say that algorithm and data structure together form the foundation for problem solving. So, as some common algorithms are available, in this module, we will try to we will take a look at what are the common data structures that are also available as a part of the C++ standard library.

We will use stack as our vehicle to understand that and see the difference between a stack written by the user in C and a stack provided by the C++ standard library and then see what are the other data structures that are also available in C++ standard library.

(Refer Slide Time: 01:47)



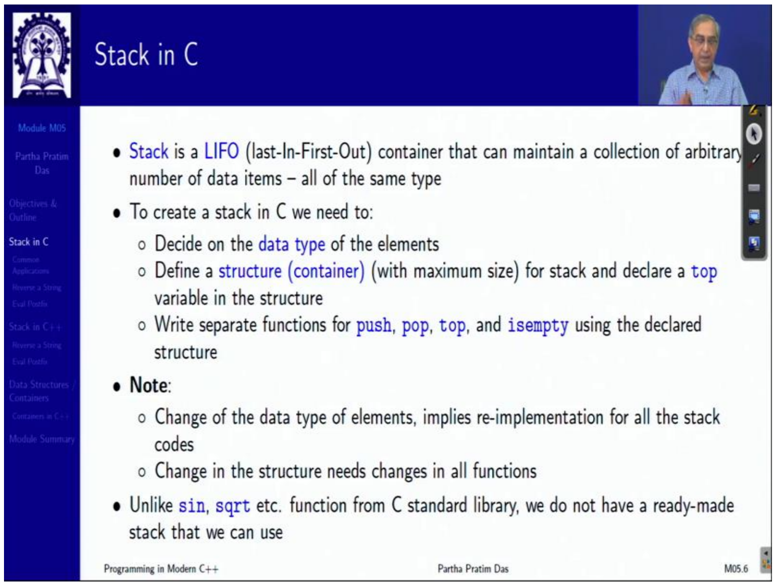
Module Outline

1. Stack in C
   - Common Applications of Stack in C
   - Reverse a String
   - Evaluate a Postfix Expressions
2. Stack in C++
   - Reverse a String
   - Evaluate a Postfix Expressions
3. Data Structures / Containers in C++
   - Containers in C++
4. Module Summary

Programming in Modern C++ Partha Pratim Das M05.4

So, this is our outline, which you will see on the left all the time.

(Refer Slide Time: 01:54)



Stack in C

- 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

Programming in Modern C++ Partha Pratim Das M05.6

So, first let us look at it stack in C. So, before I get into that, just for completeness, let me quickly go over the concept of stack. Stack as we know is a container which maintains the data element in a last in first out order called that LIFO that is the element that is added last to the collection will be removed first. And we conceptualize it as if it is a vertical structure stack of books the one who put on the top is the first one you have to remove that is it.

So, create a for creating a stack we need to decide on these things we need to decide on the data type of every element whether it is an in stack of integers stack of double stack of strings, stack of user defined structures and so on. We have to define a structure a container structure to maintain the stack. Theoretically a stack is infinite, but in reality, it cannot be so you will have a maximum size also associated with the stack.

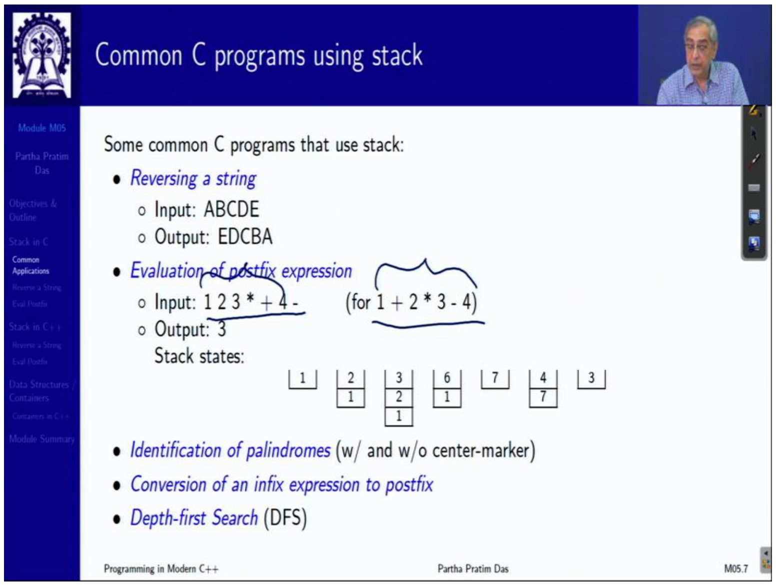
So, you would be familiar with stack being implemented with array stack being implemented as list or stack being implemented as some other DQ structure and so on. Now, we also need to declare a top variable which is the marker of the topmost element, so that if I want to add an element, it goes on top of that, if I want to remove the element, the last in element which will be first out I need to move the top in the other direction that is the purpose of top.

And the whole functionality of stack the data structure is defined in terms of four operations as we know push that is adding an element, pop that is removing an element, top that is what is the current topmost element if you want to check that several algorithms will need that and the fourth is to say whether the data structure at all of the stack at all has any element or not that is is-empty.

With these four functions, we have a total conceptual model for a stack, the last in first out. Now in C if you are writing a stack, then you know that if you change the data type of the element, then you have to re-implement the stack change the code that is the stack code for an integer and stack code for double elements would be different. And the changes needed in the structure which subsequently changes all the function code so the entire implementation has to be redone, retested all that.

And unlike sin, tan square root these kinds of functions in C standard library, C standard library has not provided with readymade stack for the simple reason that if the data type changes, the whole implementation has to be different. So, you have seen in qsort or in bsearch, how complicated it was to work with the algorithm without actually knowing the type of the data, here, you will have to do it for all these functions and so on very complicated. So, the designers decided not to provide any readymade stack in the C standard library.

(Refer Slide Time: 05:39)



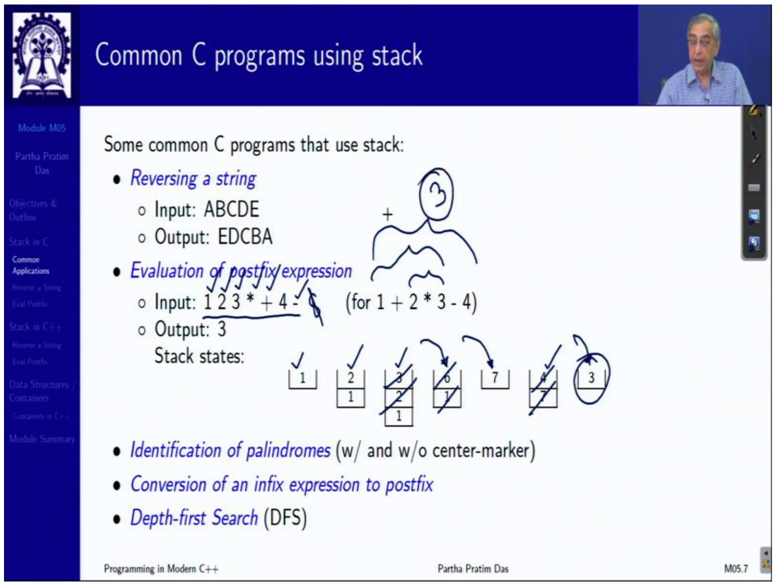
Common C programs using stack

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)

Programming in Modern C++ Partha Pratim Das M05.7



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)

Programming in Modern C++ Partha Pratim Das M05.7

Now, we all know this is just a quick recap of what you know, we all know that stack is a very important data structure, because it is very common to solve a whole number of problems. For example, you can use a stack to reverse a string, all that you need to do put the elements one after the other in the stack, and then you just take them out one after the other, push, push, push, push, push, push pop, pop, pop, pop will reverse the string, very simple last in first out reverses.

We can evaluate postfix expressions for example, here, I have an expression this where the postfix is 123 start plus 4 minus. Now I think you know postfix notation, if you do not, please read it up. Actually, stack can be used to convert an infix notation like this to the postfix notation also. This conversion itself can be done using a stack. But here what we are focusing at is the evaluation of it.

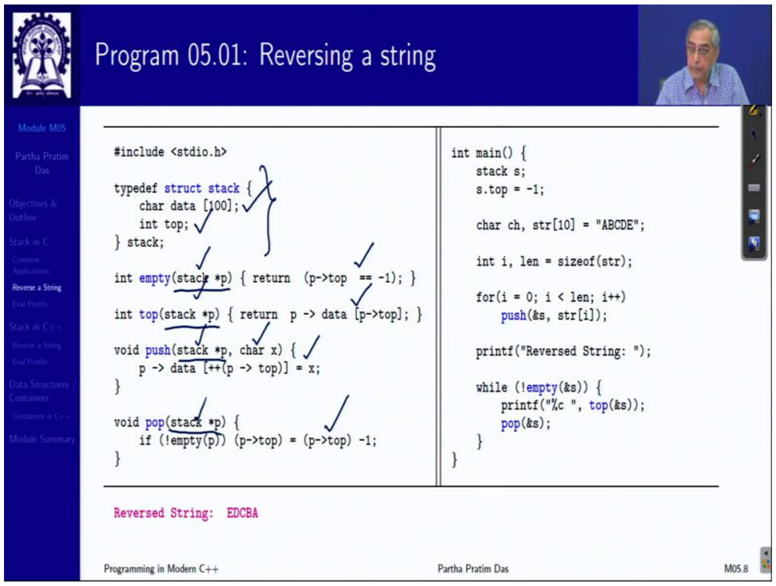
So, if the expression is given in postfix, then how do we evaluate take a stack, if you keep on scanning the input from left to right, if you have a value, put it in the stack. So, you put 1 a second value, put it in the stack push, third value push, then you get an operator, the moment you get an operator, you know what is the arity of the operator, that is how many operands we literally need. So, you take out those many elements from the stack one after the other.

So, as you have got star, which is multiplication, which is binary, you take out 3, you take out 2 and do that operation. As you do that operation, you will get 6 put it back. So, 2, 3 is gone 6 now comes here, then you have a plus. So, again you do the same thing you take out plus as binary take out two elements, 6 1, do the 6 plus 1, 7, that goes in then you get 4. So, you push it in, then you get minus which is a binary minus here. So, you take out 4 takeout, 7, do 7 minus 4, get 3 put it in, you are at the end of the input.

So, your process is over what remains in the top of the stack, it should contain actually only one value. If everything is correct, that is your evaluation result, you can check it out. If you look into this expression, there is the highest precedence for 2 times 3 is 6, then these have equal precedence, but left associativity of, 6 plus 1 is 7 and 7 minus 4 is 3, we have got the correct result. So, stack has a wide application in terms of these kinds of expression evaluation and as I said it can be used to convert an infix to postfix also.

It can be used to identify palindromes. Palindromes are which read the same from both sides like Malayalam is a palindrome you read it from left to right you read it from right to left it reads the same. So, this can be used this detection can be used by stack you can as I said convert infix to postfix you can do depth first search so many different things post order traversal all of these things require stack highly useful.

(Refer Slide Time: 09:30)



Program 05.01: Reversing a string

Module M05

Partha Pratim Das

Objectives & Outline

Stack in C

Common Applications

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

Data Structures / Containers

Containers in C++

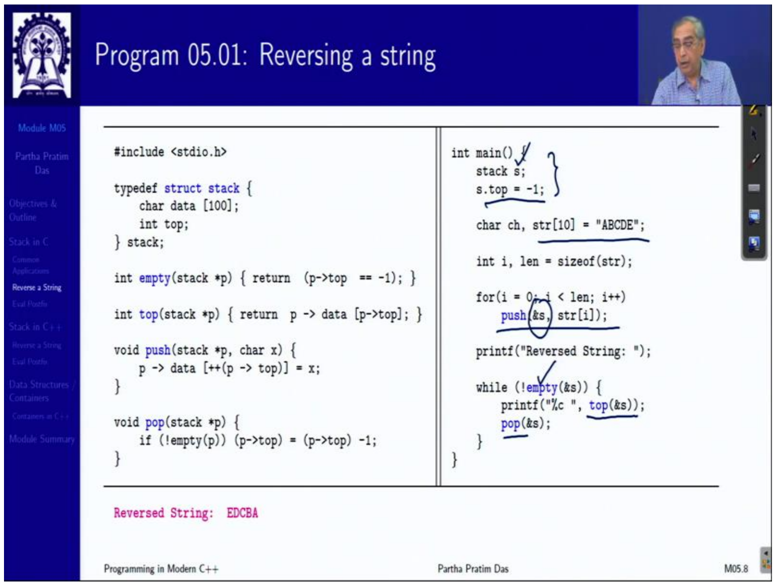
Module Summary

```
#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;
}
```

```
int 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
```

Programming in Modern C++ Partha Pratim Das M05.8



Program 05.01: Reversing a string

Module M05

Partha Pratim Das

Objectives & Outline

Stack in C

Common Applications

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

Data Structures / Containers

Containers in C++

Module Summary

```
#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;
}
```

```
int 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
```

Programming in Modern C++ Partha Pratim Das M05.8

So, let us try to recapitulate writing a stack in C. So, this is the structure I define where I have use an array as a container and I use an array index variable top as a marker. So, each one of these functions are global. The stack functions are global, because in C all functions are global. So, I need to pass the stack as a parameter to each one of these. So, each one of them takes a pointer to the stack.

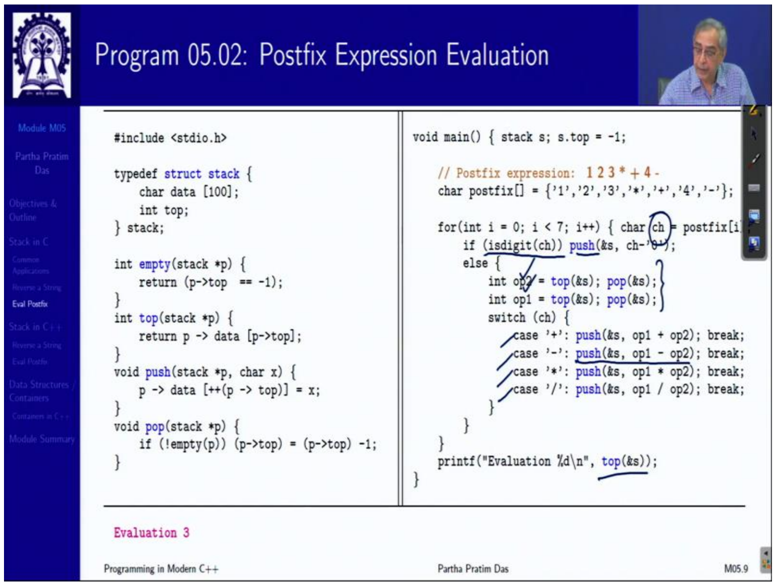
Push in additionally, will need the element that is to be pushed to the stack of characters. So, it takes a character, everything else does not need another element. So, here you check for whether your top is equal to the initial value. Here you provide the current top value here you increment the top, get the next empty location in the array and put that element there. Here, you decremented the top 2 logically remove the element.

For simplicity, where what here we have used the protocol that pop just pops, it does not return the value that it gets that it has. So, to get that value you should first do top and then do to pop to remove it. So, that is it. That is a simple way of doing it. So, now what I will have to do, I will have to declare a stack and also initialize it with the value of the marker, which shows it is empty. Now, since I have used an array, so the index starts from 0.

So, I say that the index which is invalid empty index is minus 1, one less than that. So, this is the initialization that I need to do, then I have an array. Now I am trying to array of characters, I am trying to reverse that string. So, I keep on taking character one by one, keep on pushing them in the stack that I have, I am using that particular stack. And once I am done then I keep on finding the top and popping it top pop (it top pop) it.

So, I am taking out in the reverse order till the stack gets empty. So, using these four functions, I can easily do this kind of programming, which I am sure most of you have done already. Now, let us see.

(Refer Slide Time: 12:24)



Program 05.02: Postfix Expression Evaluation

```
#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','-'};

    for(int i = 0; i < 7; i++) { char ch = postfix[i];
        if (isdigit(ch)) push(&s, ch-'0');
        else {
            int op2 = top(&s); pop(&s);
            int 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
```

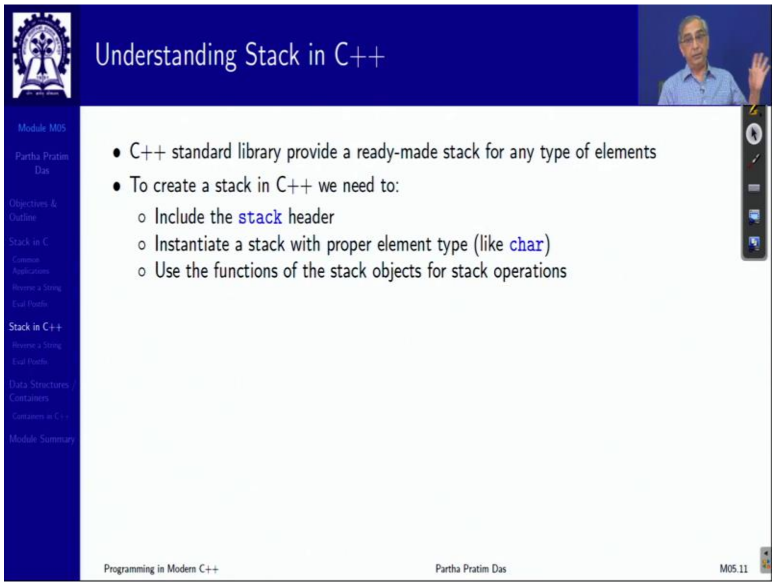
Programming in Modern C++ Partha Pratim Das M05.9

So, another example postfix evaluation, I will not go through the details, but I can just highlight the stack creation per definition part of is, is the same again I have to provide that and then for doing this evaluation, I have to check if something is a digit, because then I have to push it or if it is an operator, then I have to take out the top two and operate. So, if it is a digit I push otherwise, it will have to be an operator because that is only two things that are available in a postfix expression.

So, I take out the top two elements one after the other. And I already know what the operator is which is in ch. So, I find do a case of possible operators and whichever it matches, I will take the two operands op1 and op2 that I have got operate them according to the operator and the result I put back to the stack. I keep on doing this till I come to the end of the postfix expression and things going correct at that time, I will have only one element left which is the result in the stack which I take out as top.

Postfix expression evaluation very smoothly done. So, stack, this kind of programming is very, very common. So, if we now step ahead and look at what happens in C++.

(Refer Slide Time: 14:08)

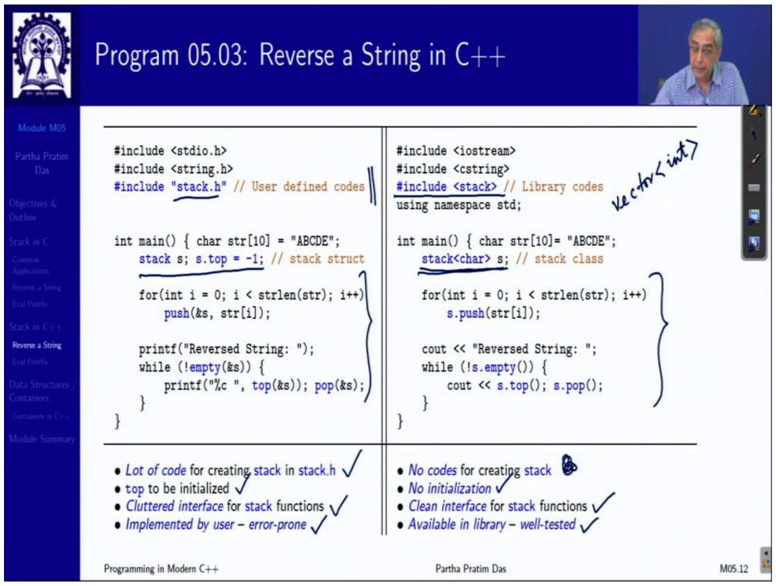


Understanding Stack in C++

- 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

In C++, the standard library provides a separate header, a separate component called stack. And the interesting part is that is a stack of any type as you want, again uses that magic of templates, meta programming, we will come to that later on.

(Refer Slide Time: 14:31)



Program 05.03: Reverse a String in C++

```
#include <stdio.h>
#include <string.h>
#include "stack.h" // User defined codes

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

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

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

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

```
#include <iostream>
#include <cstring>
#include <stack> // Library codes
using namespace std;

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

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

    cout << "Reversed String: ";
    while (!s.empty()) {
        cout << s.top(); s.pop();
    }
}
```

vector< int>

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

Programming in Modern C++ Partha Pratim Das M05.12

So, let us look at on the left is your C code. So, all these are scripted here all that you wrote for the stack function stack structure has to be given which is given in say a hidden stack dot h that you have written. And then, you have to define and initialize and then do this algorithm to reverse.

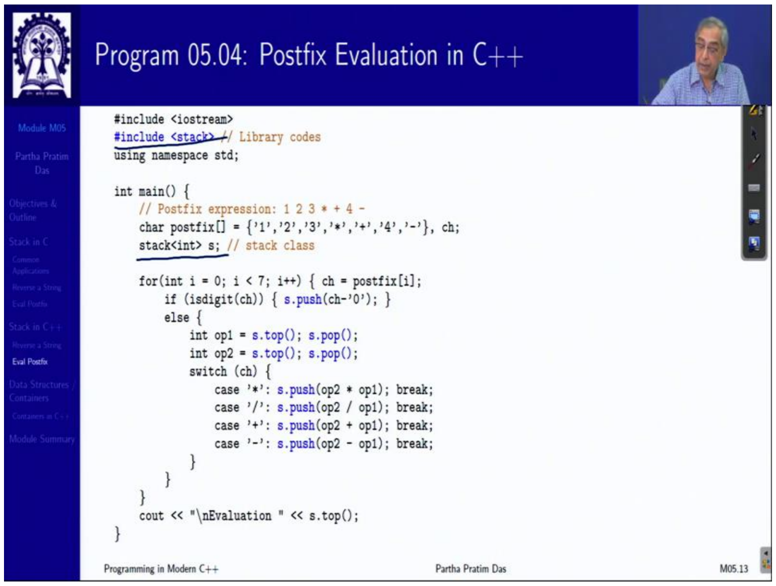
In contrast, in C++ all that you need to do is to this header, it gives you the entire stack, correct one. And when you want to create all that, you do is, you remember this tile was used in vector also we did vector int. This is that template style where you say that the stack is written assuming that the data type could be anything, it is a variable. And now you are saying that assume that the data type is char.

So, compiler will take that and create the set of codes that are required for a stack of chars and compile them. So, you have your readymade step, through writing just this couple of characters. And once that has been done, rest of the code works exactly in the same way. So, you, you have a lot of advantage now, because a lot of code that you had to write in terms of stack.h, is not required, there is no code that needs to be written here.

The you needed a separate initialization, which if you forget, or do erroneously, everything will fall apart. There is no initialization required, in this case, the whole cluttered interface of stack functions which require every time the stack pointer to be pointed to the stack to be passed are not required, you are just calling it as, as functions of that particular stack object it is a clean interface.

And here you have to write it for every type you have to write separately tested, it is quite, not only laborious, but it is error prone. Here, there is just only one code which is well tested and rest of it, the automatic generator is doing. So, it is available in the library very less amount of work. So, that is the advantage of having data structures as a part of your library, which C++ strongly provides.

(Refer Slide Time: 17:17)



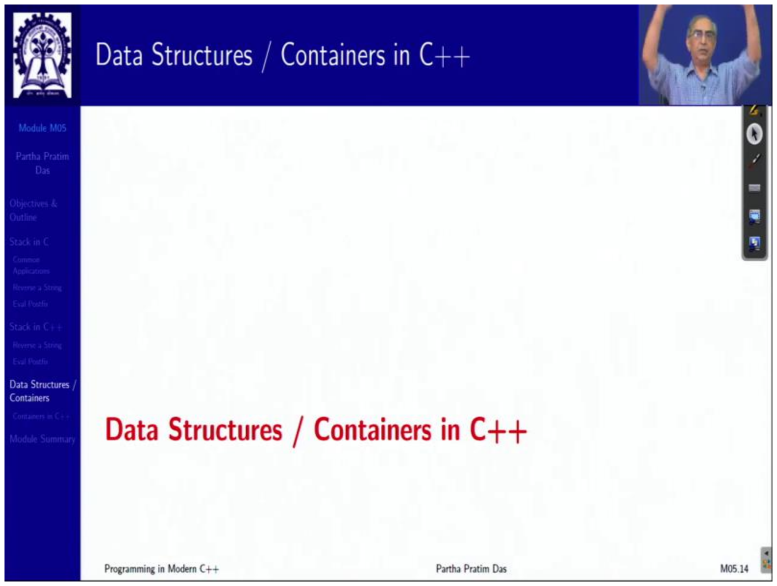
Program 05.04: Postfix Evaluation in C++

```
#include <iostream>
#include <stack> // Library codes
using namespace std;

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

    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();
}
```

Programming in Modern C++ Partha Pratim Das M05.13

So, this is just for your, completeness, this is the postfix evaluation coordinates are written in C++, which is exactly same as the code that you wrote in C except for the definition of the stack. And this particular component from where the stack codes are coming, everything else is the same. And it is just a matter of a couple of minutes in which you can easily write such code.

(Refer Slide Time: 17:51)



Data Structures / Containers in C++

Module M05

Partha Pratim Das

Objectives & Outline

Stack in C

Common Applications

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

Data Structures / Containers

Containers in C++

Module Summary

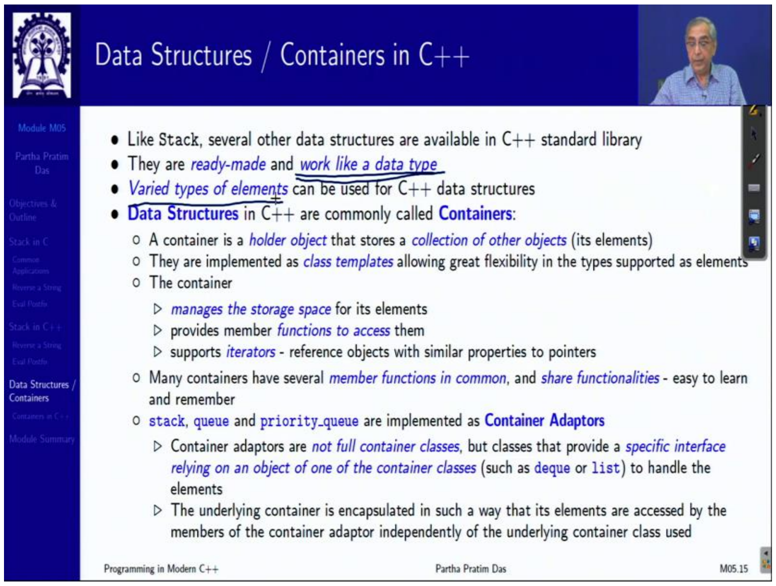
Data Structures / Containers in C++

Programming in Modern C++ Partha Pratim Das M05.14

Now, having said that, you understand that stack is a data structure and in generality C++ or I should more specifically say C++ standard library calls the data structures as containers, because, what is a data structure? Data structure is, is basically a way to keep the data keep a collection of data and then a certain protocol of operations that you can do on it these two together forms a data structure.

Stack also maintains data array also maintains data queue also maintains data, everybody every data structure, think of it has to store certain amount of data. And the different in terms of the operations that you stand as a LIFO, queue as a FIFO array as a random access and so on, so forth.

(Refer Slide Time: 18:46)



Data Structures / Containers in C++

Module M05

Partha Pratim Das

Objectives & Outline

Stack in C

Common Applications

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

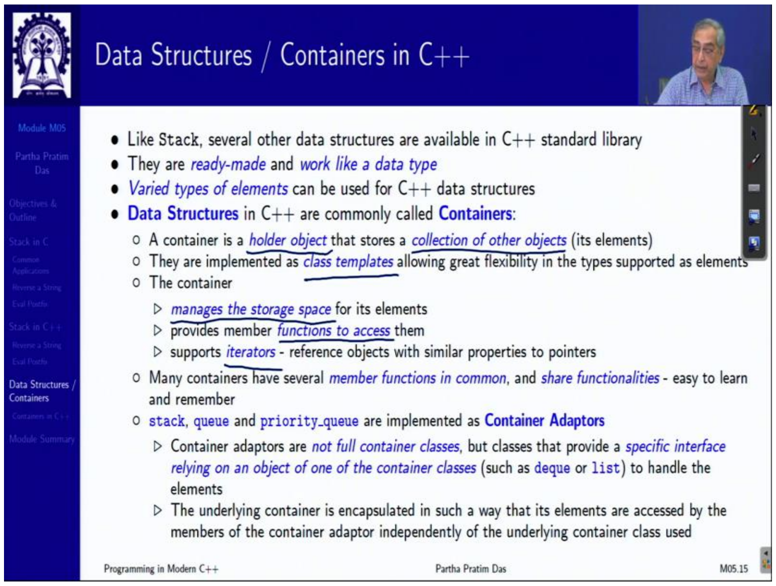
Data Structures / Containers

Containers in C++

Module Summary

- Like `Stack`, several other data structures are available in C++ standard library
- They are *ready-made* and *work like a data type*
- *Varied types of elements* can be used for C++ data structures
- **Data Structures** in C++ are commonly called **Containers**:
  - A container is a *holder object* that stores a *collection of other objects* (its elements)
  - They are implemented as *class templates* allowing great flexibility in the types supported as elements
  - The container
    - ▷ *manages the storage space* for its elements
    - ▷ provides member *functions to access* them
    - ▷ supports *iterators* - reference objects with similar properties to pointers
  - Many containers have several *member functions in common*, and *share functionalities* - easy to learn and remember
  - `stack`, `queue` and `priority_queue` are implemented as **Container Adaptors**
    - ▷ Container adaptors are *not full container classes*, but classes that provide a *specific interface relying on an object of one of the container classes* (such as `deque` or `list`) to handle the elements
    - ▷ The underlying container is encapsulated in such a way that its elements are accessed by the members of the container adaptor independently of the underlying container class used

Programming in Modern C++ Partha Pratim Das M05.15



Data Structures / Containers in C++

Module M05

Partha Pratim Das

Objectives & Outline

Stack in C

Common Applications

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

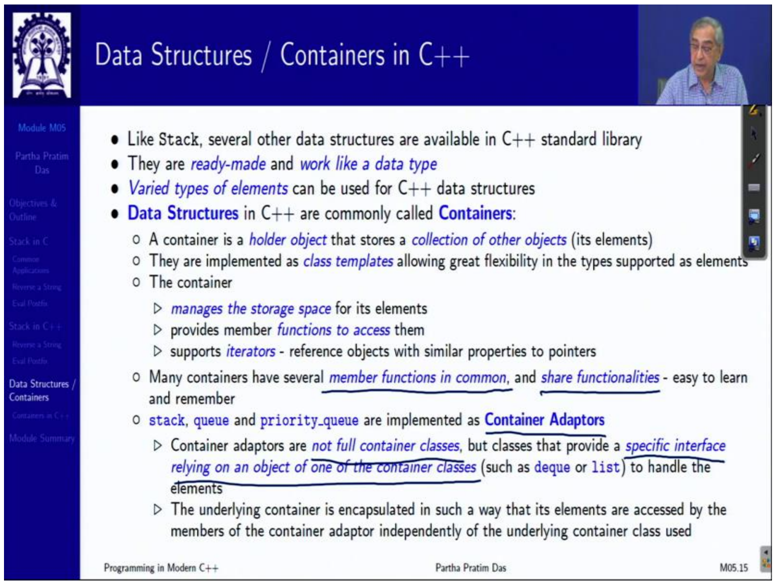
Data Structures / Containers

Containers in C++

Module Summary

- Like `Stack`, several other data structures are available in C++ standard library
- They are *ready-made* and *work like a data type*
- *Varied types of elements* can be used for C++ data structures
- **Data Structures** in C++ are commonly called **Containers**:
  - A container is a *holder object* that stores a *collection of other objects* (its elements)
  - They are implemented as *class templates* allowing great flexibility in the types supported as elements
  - The container
    - ▷ *manages the storage space* for its elements
    - ▷ provides member *functions to access* them
    - ▷ supports *iterators* - reference objects with similar properties to pointers
  - Many containers have several *member functions in common*, and *share functionalities* - easy to learn and remember
  - `stack`, `queue` and `priority_queue` are implemented as **Container Adaptors**
    - ▷ Container adaptors are *not full container classes*, but classes that provide a *specific interface relying on an object of one of the container classes* (such as `deque` or `list`) to handle the elements
    - ▷ The underlying container is encapsulated in such a way that its elements are accessed by the members of the container adaptor independently of the underlying container class used

Programming in Modern C++ Partha Pratim Das M05.15



Data Structures / Containers in C++

- Like Stack, several other data structures are available in C++ standard library
- They are *ready-made* and *work like a data type*
- *Varied types of elements* can be used for C++ data structures
- **Data Structures** in C++ are commonly called **Containers**:
  - A container is a *holder object* that stores a *collection of other objects* (its elements)
  - They are implemented as *class templates* allowing great flexibility in the types supported as elements
  - The container
    - ▷ *manages the storage space* for its elements
    - ▷ provides member *functions to access* them
    - ▷ supports *iterators* - reference objects with similar properties to pointers
  - Many containers have several *member functions in common*, and *share functionalities* - easy to learn and remember
  - stack, queue and priority_queue are implemented as **Container Adaptors**
    - ▷ Container adaptors are *not full container classes*, but classes that provide a *specific interface relying on an object of one of the container classes* (such as deque or list) to handle the elements
    - ▷ The underlying container is encapsulated in such a way that its elements are accessed by the members of the container adaptor independently of the underlying container class used

Programming in Modern C++ Partha Pratim Das M05.15

So, C++ prefer to call them as containers. So, there are in C++ standard library, there are several components for different containers, they are readymade. And the most important point to be noted, which is what we will keep on learning all through this course. So, that even we can write similar libraries in future if we need to, is the fact that every container works like a data type. Every container works like a data type. And that is the beauty of the whole thing.

And we will slowly understand that. There can be the containers can have elements of varied type, including user defined types and are defined in that appropriate way. So, in C++, these containers are called holder objects that can contain the collection of other objects or its elements. As I said, it is implemented using class template you will learn that later on, but just consider that it is a kind of imagination.

As you write the template of the code, the basic structure of the code without having to write, without using the knowledge of the element type that is involved, you take the element type as if as a variable. And then when it is used, the element type is known, the compiler generates a code for that element type, and compiles it. So, that is the basic. So, it manages, of course, it has to manage the storage space, because it is a container and it provides functions to access them, which makes it a data structure a collection to store data plus the operations like push pop top and empty makes something a stack.

Similarly, so the functions to access them will be provided. It also supports iterators, I briefly mentioned in earlier module iterators are nothing but, generalized for loops. So, it can you can provide on your data structure, if you can provide a range, then you can go from one end to the other end of the range doing something, you can go in the forward direction, you can go in the reverse direction.

Now, having structured in this way, some of the byproducts or design advantages that you get is several containers have member functions in common. For example, in general, let us think of if you have a data structure, what are the common things you will always have? First of all, there will be a way to create it, and there will be a way to destroy it. Of course, that has to be there. That is true for everybody. In most cases, there will be a way to add an element, there will be a way to remove an element.

And in many, there could be a way to find an element, and so on, so forth. So, there are a lot of member functionalities, which are common or are shared. And C++ standard library makes use of that. So, once you understand the basic philosophy, common philosophy of this data structures, you learn all the data structures together, you do not have to remember everything that is that is the beauty. And we will see that more.

Now some of the containers are known as container adapters. Because they are not new containers, there is some container and they just been adapted to behave differently, giving us a different set of functions. For example, let us say if I use implement a stack using an array, then the basic container is the array. But I do not want to see that as an array, I want to see it as a stack. If I see it as an array, I will do random access. But I do not want to do that.

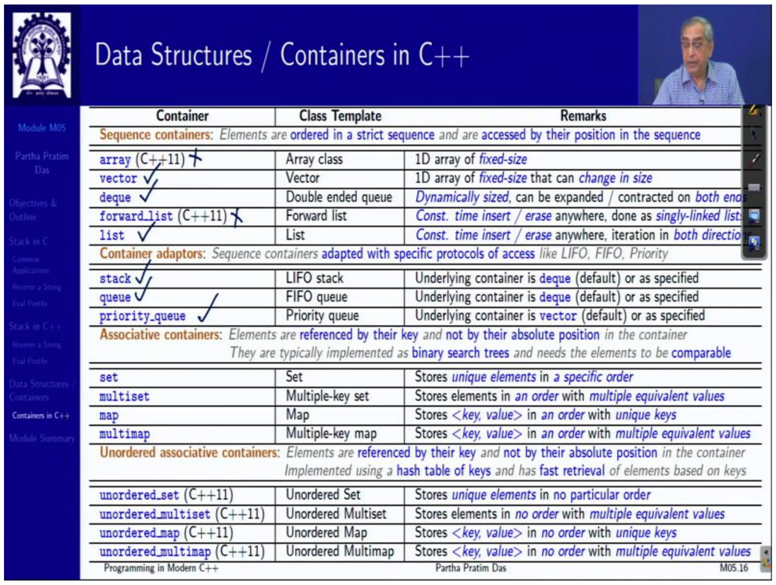
I want to look at it as a stack that which gives me push pop is-empty and top. So, I am adapting the original container array with a set of functions, which make it behave like a stack. So, it is not a fundamentally different whereas if I talk about a linked list, it is a fundamentally different container than an array.

Because it does not have random access. But it has the ability to random insert random delete. If I am talking about a tree, it is a fundamentally different container. But when I am talking about array and stack or say list and stack, stack is not a different container, it is an adaption of existing convenient container.

The similar thing is for the queue or for the priority queue which some of you may know as a heap, it is also called the heap data structure. The heap sort is based on that and so on. So, containers are not therefore are not full container classes. They use some underlying container and provide a specific interface relying on the object of one of the container classes. So, stack is one kind of interface maybe on an array.

Queue is another kind of interface, maybe on an array, or stack is one kind of interface on the list queue is another kind of interface on the list, but the underlying container is a full container. And these are just adapters and these adapters are very important for writing several good algorithms.

(Refer Slide Time: 24:58)



Data Structures / Containers in C++

Module M05

Partha Pratim Das

Objectives & Outline

Stack in C

Common Applications

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

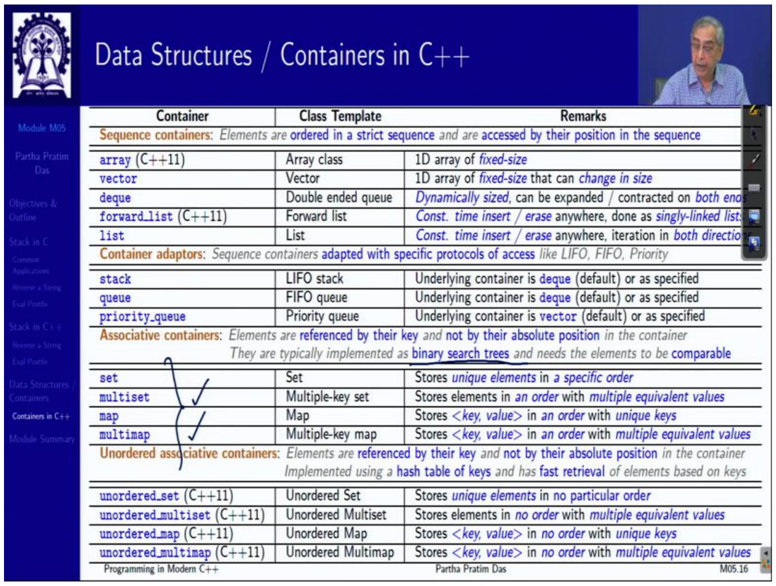
Data Structures / Containers

Containers in C++

Module Summary

Container	Class Template	Remarks
Sequence containers: *Elements are ordered in a strict sequence and are accessed by their position in the sequence*		
`array` (C++11) ✗	Array class	1D array of *fixed-size*
`vector` ✓	Vector	1D array of *fixed-size* that can *change in size*
`deque` ✓	Double ended queue	*Dynamically sized*, can be expanded / contracted on *both ends*
`forward_list` (C++11) ✗	Forward list	*Const. time insert / erase* anywhere, done as *singly-linked lists*
`list` ✓	List	*Const. time insert / erase* anywhere, iteration in *both directions*
Container adaptors: *Sequence containers adapted with specific protocols of access like LIFO, FIFO, Priority*		
`stack` ✓	LIFO stack	Underlying container is `deque` (default) or as specified
`queue` ✓	FIFO queue	Underlying container is `deque` (default) or as specified
`priority_queue` ✓	Priority queue	Underlying container is `vector` (default) or as specified
Associative containers: *Elements are referenced by their key and not by their absolute position in the container. They are typically implemented as binary search trees and needs the elements to be comparable*		
`set`	Set	Stores *unique elements* in *a specific order*
`multiset`	Multiple-key set	Stores elements in *an order* with *multiple equivalent values*
`map`	Map	Stores *<key, value>* in *an order* with *unique keys*
`multimap`	Multiple-key map	Stores *<key, value>* in *an order* with *multiple equivalent values*
Unordered associative containers: *Elements are referenced by their key and not by their absolute position in the container. Implemented using a hash table of keys and has fast retrieval of elements based on keys*		
`unordered_set` (C++11)	Unordered Set	Stores *unique elements* in no particular order
`unordered_multiset` (C++11)	Unordered Multiset	Stores elements in *no order* with *multiple equivalent values*
`unordered_map` (C++11)	Unordered Map	Stores *<key, value>* in *no order* with *unique keys*
`unordered_multimap` (C++11)	Unordered Multimap	Stores *<key, value>* in *no order* with *multiple equivalent values*

Programming in Modern C++ Partha Pratim Das M05.16



Data Structures / Containers in C++

Container	Class Template	Remarks
Sequence containers: *Elements are ordered in a strict sequence and are accessed by their position in the sequence*		
`array` (C++11)	Array class	1D array of *fixed-size*
`vector`	Vector	1D array of *fixed-size* that can *change in size*
`deque`	Double ended queue	*Dynamically sized*, can be expanded / contracted on *both ends*
`forward_list` (C++11)	Forward list	*Const. time insert / erase* anywhere, done as *singly-linked lists*
`list`	List	*Const. time insert / erase* anywhere, iteration in *both directions*
Container adaptors: *Sequence containers adapted with specific protocols of access like LIFO, FIFO, Priority*		
`stack`	LIFO stack	Underlying container is `deque` (default) or as specified
`queue`	FIFO queue	Underlying container is `deque` (default) or as specified
`priority_queue`	Priority queue	Underlying container is `vector` (default) or as specified
Associative containers: *Elements are referenced by their key and not by their absolute position in the container. They are typically implemented as binary search trees and needs the elements to be comparable*		
`set`	Set	Stores *unique elements* in *a specific order*
`multiset` ✓	Multiple-key set	Stores elements in *an order* with *multiple equivalent values*
`map`	Map	Stores *<key, value>* in *an order* with *unique keys*
`multimap` ✓	Multiple-key map	Stores *<key, value>* in *an order* with *multiple equivalent values*
Unordered associative containers: *Elements are referenced by their key and not by their absolute position in the container. Implemented using a hash table of keys and has fast retrieval of elements based on keys*		
`unordered_set` (C++11)	Unordered Set	Stores *unique elements* in no particular order
`unordered_multiset` (C++11)	Unordered Multiset	Stores elements in *no order* with *multiple equivalent values*
`unordered_map` (C++11)	Unordered Map	Stores *<key, value>* in *no order* with *unique keys*
`unordered_multimap` (C++11)	Unordered Multimap	Stores *<key, value>* in *no order* with *multiple equivalent values*

Programming in Modern C++ Partha Pratim Das M05.16

So, do not get shocked by looking at this list, obviously, on the on day one, nobody expects you to remember this, but you have to with us get accustomed to this set of data structures because they are available in the C++ standard library. So, you do not have to do anything just instantiate and you are ready to go. And these data structures are designed in a way so that most of the common problems that you have to solve using data structures are available here as readymade.

So, if we quickly take a look forget about those which are marked as C++ 11. We will talk about them when you come to the modern part of it. We have already seen what is a vector, it is an array with a lot more of power we have a list which is bi-directional list, doubly linked list what we call. So, it can be expanded and contracted at both ends and works in that way. We have a dequeue which is Double Ended queue, which means that you can add and remove elements at both ends.

What is stack is where you add remove elements at one end queue where you add element at one end remove element from the other end. And dequeue is a generalization where you can add and remove elements from both ends. So, you have all of that ready, then you have adapters like stack queue priority queue with the expected member functions that are their exact member function name you have to know but the beauty is the function name to add is same between stack and the queue and the priority queue.

So, once you learn one, you know the rest. Then you have a set of containers which are associative. What does the associative mean in array stack queue priority queue and so on. You are you find out an element based on either its position or its priority, or its order of arrival, and so on, not by the value of the element. You never need to know the value of the element to do these operations. To do push pop, top empty, you are not concerned with the value of the element that you are dealing with.

All that you are dealing with is the order of the arrival and the order of departure of the elements to the data structure. There are several containers where you need to do the other way you are concerned about what is the value for example, you are creating a set. Now set does not have any ordering as you know it is a structure mathematically it is a collection where the elements are unique and are have no specific order.

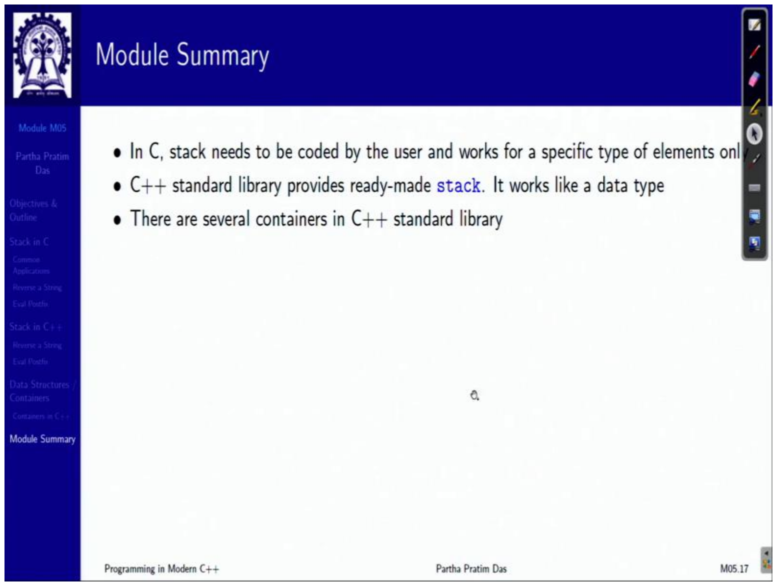
So, here these kinds of containers are called associative. There is a map, map is nothing but the name value pair you have a key and a value the keys are assumed to be unique. So, that given the value of the key you can find out what is the value of the associated with that particular key which is very required. This can be for example map can be used to represent a graph where you have associations between two nodes by an age which gives you the graph.

So, set and map, map is also commonly you will commonly know as a hash operation, but here it is a little different. Hash directly is not available. So, these are associated with where the value is the important part in keeping in the way we will keep it in the data structure. So, set and map are the fundamental ones, which in an underlying way uses binary search tree structure. The underlying structure is a binary search tree.

Now binary search tree per se is not given as a component but the set and map they use binary search tree internally and obviously need the elements to be comparable. Now, so which is a little bit maybe for a set, it may be a little bit uncomfortable because you want to make a set of anything. And if you have to build a binary search tree based on the elements of a set, you have to make them comparable and which may not be a little unnatural.

So, there are some naturality in that. It also has multi-set and multi-map, multi-set is a set where duplication is allowed multi-map is one where duplication in terms of the key value is allowed. And this need to be compatible is done away with in C++ where you have unordered versions of them, but as I said, we will not talk about this right now.

Refer Slide Time: 30:37)



Module Summary

Module M05

Partha Pratim Das

Objectives & Outline

Stack in C

Common Applications

Reverse a String

Eval Postfix

Stack in C++

Reverse a String

Eval Postfix

Data Structures / Containers

Containers in C++

Module Summary

- In C, stack needs to be coded by the user and works for a specific type of elements onl
- C++ standard library provides ready-made stack. It works like a data type
- There are several containers in C++ standard library

Programming in Modern C++ Partha Pratim Das M05.17

So, we have overall given a view of the availability of data structures in C++ as a standard library using stack as a primary regular example, and this will be very useful for you to develop C++ programs very easily. Thank you so much for your attention and we will meet in the next module.