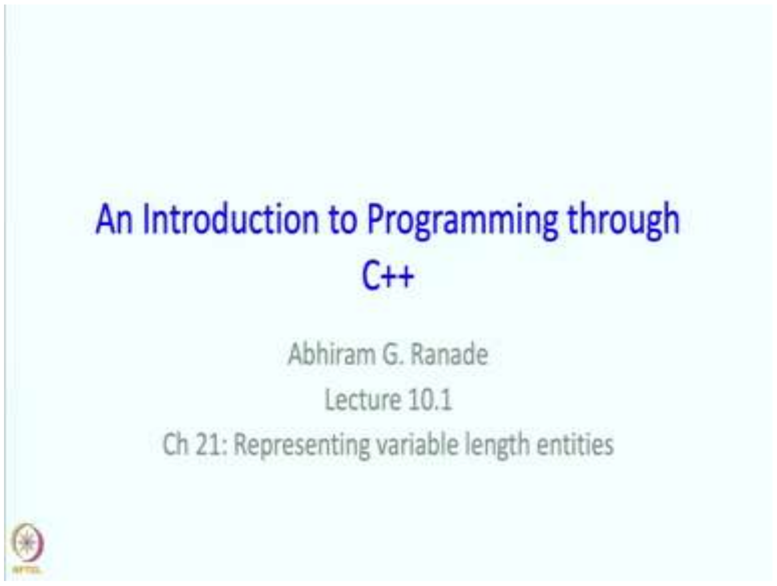
An Introduction to Programming through C++

Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture-22 Part-01
Representing variable length entities
Introduction

Hello, and welcome to the NPTEL course on an introduction to programming through C++.

(Refer Slide Time: 0:31)

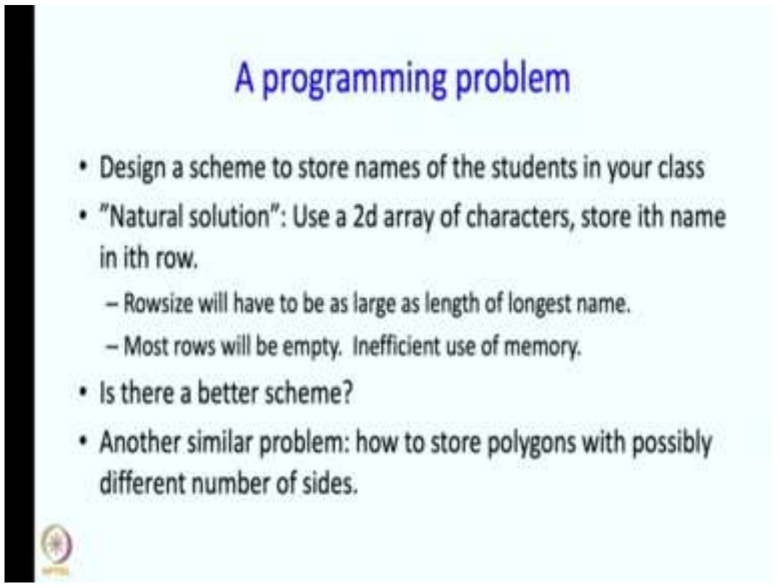


An Introduction to Programming through C++

Abhiram G. Ranade
Lecture 10.1
Ch 21: Representing variable length entities

I am Abhiram Ranade and this lecture is about representing variable-length entities. And the reading for it is chapter 21 of the textbook. So, let me begin with the programming problem.

(Refer Slide Time: 00:43)



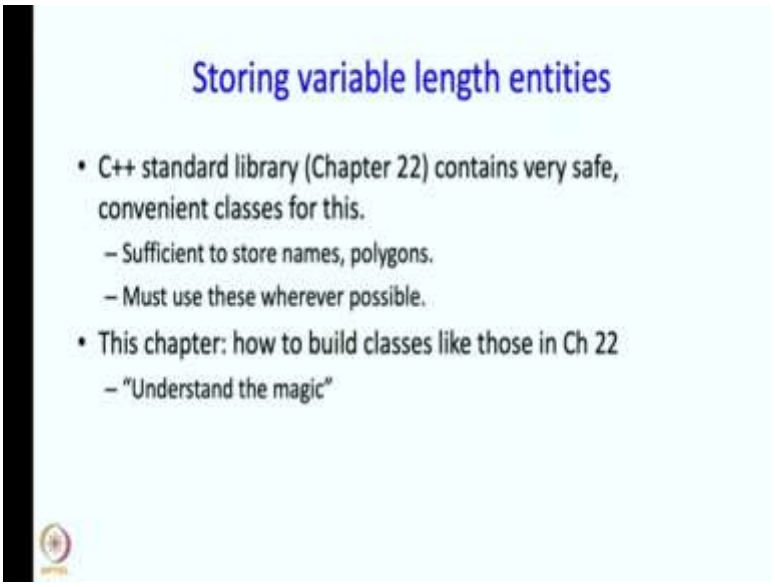
A programming problem

- Design a scheme to store names of the students in your class
- "Natural solution": Use a 2d array of characters, store ith name in ith row.
  - Rowsize will have to be as large as length of longest name.
  - Most rows will be empty. Inefficient use of memory.
- Is there a better scheme?
- Another similar problem: how to store polygons with possibly different number of sides.

So, we would like to design a scheme to store the names of the students in your class. Now there is a natural solution to this use a two-dimensional array like we talked very recently store ith name in ith row. Now, what will the row size have to be? So, the row size will have to be as large as the length of the longest name. And therefore, it seems that most rows will be mostly empty. And therefore, we are using the memory quite inefficiently. So, you might wonder. Is there is a better scheme?

And this is not the only problem that we are looking at today. Indirectly, there are also other problems which are very similar. So, for example, for some reason, I might want to store a whole bunch of polygons in my program. And say the polygons have varying number of sides. So, again the provision that I might have to make might be for the largest polygon whereas I might end up having a lot of small polygons. So, again I will end up wasting a lot of space.

(Refer Slide Time: 02:05)

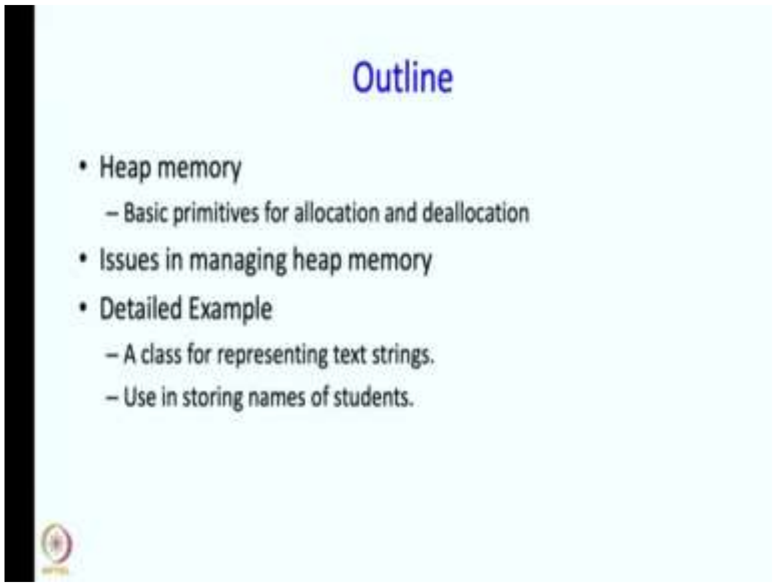


Storing variable length entities

- C++ standard library (Chapter 22) contains very safe, convenient classes for this.
  - Sufficient to store names, polygons.
  - Must use these wherever possible.
- This chapter: how to build classes like those in Ch 22
  - "Understand the magic"

So, the C++ standard library which is discussed in chapter 22 of the book and which we will take up later on contains very safe and convenient classes for this. And they will be sufficient to store names, polygons and other such entities and you should use them wherever possible. But in this lecture or in this chapter of the book, we are going to talk about how to build the classes like those in chapter 22. So, essentially we are going to understand what is the mechanism behind these classes? So, sort of what is the magic on which these classes run?

(Refer Slide Time: 2:46)

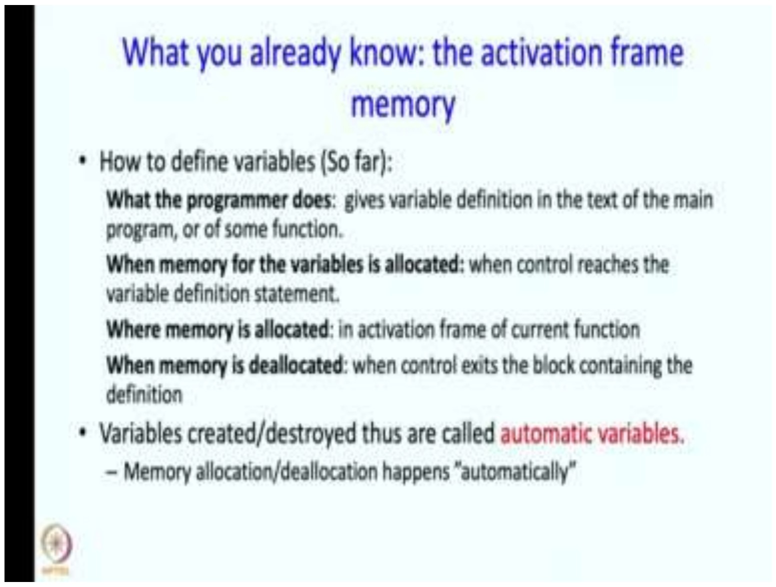


Outline

- Heap memory
  - Basic primitives for allocation and deallocation
- Issues in managing heap memory
- Detailed Example
  - A class for representing text strings.
  - Use in storing names of students.

So, here is the outline for today's lecture. So, I am going to talk about something called the heap memory. Which is something that you have not seen so far. And there will be a discussion of the primitives for allocation and deallocation of memory from this heap memory. Then we will talk about how do you manage this heap memory and we will do a detailed example. And in this example, we will talk about a class for representing text strings and this will be useful in storing the names of students, the problem with which we started this lecture.

(Refer Slide Time: 3:26)



What you already know: the activation frame memory

- How to define variables (So far):

  What the programmer does: gives variable definition in the text of the main program, or of some function.

  When memory for the variables is allocated: when control reaches the variable definition statement.

  Where memory is allocated: in activation frame of current function

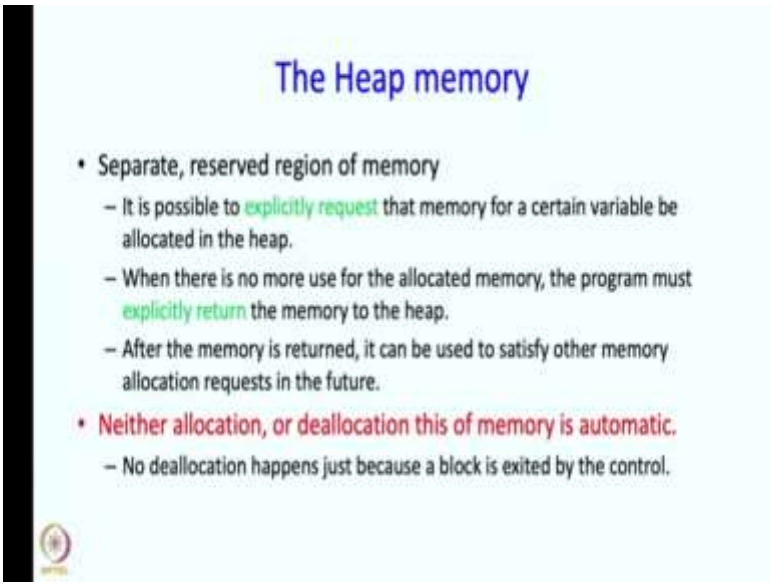
  When memory is deallocated: when control exits the block containing the definition
- Variables created/destroyed thus are called automatic variables.
  - Memory allocation/deallocation happens "automatically"

So, let me begin with what you already know. So, you already know the activation frame memory. So, how do we define variables, well so far the idea has been that the programmer will give the variable definitions in the text of the main program, or of some function as well. And the memory for these variables will be allocated when control reaches the variable definition statements.

And the memory will be allocated in the activation frame of the current function. And it will also be deallocated, well it will be deallocated when the control exits the block containing the definition. So, it could be exiting the function, it could be a ‘for statement’ whose control variables might be getting deallocated. But yes the variable will be deallocated as well and there is a very clear point where the variable is going to get deallocated. Now, variables which are created and destroyed in this way are called automatic variables.

Automatic because the memory allocation and deallocation sort of happens automatically, well, I mean you could say that the user is saying, user is defining the variable and therefore, it is getting created, yes. But at least the deallocation is the sort of. I mean it is indicated by the end of the block. But anyway so because the user is not explicitly asking for memory, at least the user is not explicitly giving up memory these things are called automatic variables.

(Refer Slide Time: 05:31)



The Heap memory

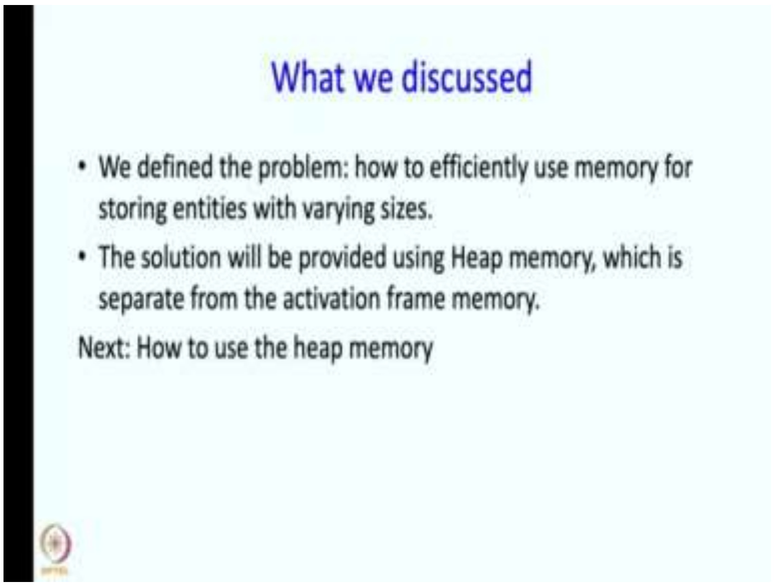
- Separate, reserved region of memory
  - It is possible to explicitly request that memory for a certain variable be allocated in the heap.
  - When there is no more use for the allocated memory, the program must explicitly return the memory to the heap.
  - After the memory is returned, it can be used to satisfy other memory allocation requests in the future.
- Neither allocation, or deallocation this of memory is automatic.
  - No deallocation happens just because a block is exited by the control.

In addition to the activation frames, we also have this so called heap memory. So, this is a completely separate reserved region of memory. Completely different from the activation frames and in this memory, it is possible to explicitly request, say, give me some memory for a certain variable and that memory will be given from the heap and we will be saying that this variable has been allocated to the variable on the heap, whatever.

And when there is no more use for the allocated memory the program or maybe the programmer must explicitly return the memory to the heap. So, the program there must be a statement in the program which will say okay now I do not need this memory take it back, put it back and maybe give it to me later if I ask for it again. So, after the memory is returned it can be used to satisfy other memory allocation requests that might come up with the future.

Now, as you can see, there are explicit statements which say that give me this memory or there will be an explicit statement which says, “No! No! Take this back.” And so, therefore, this memory is not automatic in any sense. And even the deallocation is explicit, it does not happens just because of block is exited by the control.

(Refer Slide Time: 7:01)



What we discussed

- We defined the problem: how to efficiently use memory for storing entities with varying sizes.
- The solution will be provided using Heap memory, which is separate from the activation frame memory.

Next: How to use the heap memory

All right, what had we discussed so far? Well, we have defined the problem being how to efficiently use memory for storing entities with varying sizes. And we had said the solution will be provided using heap memory, which is separate from the activation frame memory. And next we are going to talk about how to use the heap memory but we will take a quick break.