Lecture C5: Access Types

Response to 'Muddiest Part of the Lecture Cards'

(17 respondents)

1) Since arithmetic operations aren't defined on pointers, what is the best way to increment through the pointers to a list?

You keep track of the pointers by chaining the nodes together.

2) Are ListNode, List, .Head, and .Next already implemented in Ada?

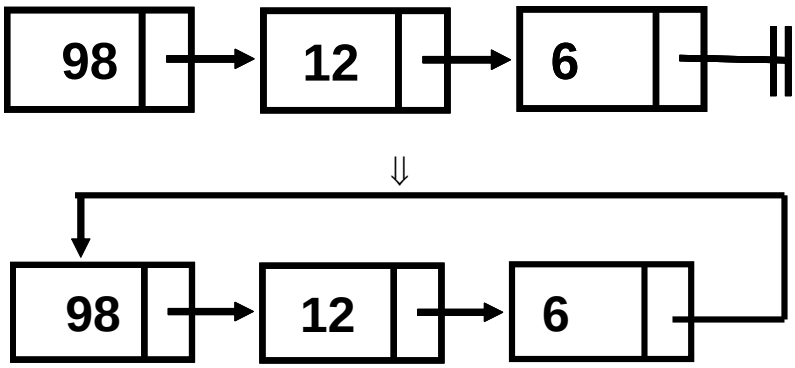
They are not predefined in the Ada language. The operations are defined in Linked_List.ads and Linked_List.adb

3) How would one keep track of all the pointers if removing them manually?

You do not de-allocate the pointer. You de-allocate the node pointed to by the pointer.

4) Are we going to be dealing with circular lists?

You should be able to extend the code for the singly linked list into a circular list, by making the last node point to the first node.



5) What does (IKL,5) imply?

The my_record:= **new** contact'("IKL", 5) statement allocates a record from the heap and initializes the fields to IKL and 5.

6) How do you print an entire list?

(From Linked_List.Adb)

```
 -- set the pointer to the head of the node
Temp:= L.Head;
while Temp /= null loop
   Put(Temp.Element);
   Put(" , ");
   -- move pointer to the next node
   Temp :=Temp.Next;
end loop;
```

Given the head of the list, you just have to traverse the list till you reach the null. In each node, print the contents.

7) Why is the head in the stack (slide 16-19)?

(From List_Test.Adb)

```
procedure List_Test is
  My_List : List;
```

My_List is defined among the variables in the program hence it is defined in the stack. If you were to use new to allocate memory, then the heap is used.

8) I do not understand what UNCHECKED_DEALLOCATION is.

Unchecked de-allocation is used to return dynamically allocated memory back to the heap. It is programmer's responsibility to ensure that there are no dangling pointers or that he/she is not trying to de-allocate memory that has already been de-allocated.

9) What was the function of the 'Head' pointer being a record? Was it just a pointer?

Head is of type Listptr within the type List. You can just define Head as a Listptr.

(From Linked_List.Ads)

```
type Listnode;
 type Listptr is access Listnode;
 type Listnode is
   record
     Element : Elementtype;
     Next    : Listptr;
   end record;

 type List is
   record
     Head : Listptr;
   end record;
```

The advantage of using a record within the type definition of List, is that it can be easily expanded to contain more than one pointer (as you will see when we talk about trees). Another advantage is that by defining distinct types for the list and the nodes of the list, they are not mixed up.

10) Can you create a pointer to an existing record or do you have to create a new one every time?

You can create a pointer to an existing record, provided the record is aliased. See section 3.10.2 in the LRM

11) Are all of the 'dangers' (as in C++) taken out of Ada because of the fact that it was designed differently?

The advantage of eliminating operations on pointers, is that it enforces better design.

12) PC3.All := PC2.All. What happens if PC3 has more memory than PC2.

They types will not match in that case, and you will not be able to compile the program.

13) Can a single record point to multiple nodes?

A single record can be used to point to multiple nodes, provided there are an equal number of fields to point to those nodes.

14) No mud (5 students)