§12.1-12.5: Pointers

24 Nov 2005
CMPT14x
Dr. Sean Ho
Trinity Western University

Reminders:

- ***journals in folder***
- ***Midterms will be back tomorrow***

Review of last time (11.4-11.9)

- Constructors: Type { list }
  - Set constructors
  - Array constructors
  - Record constructors
- Variant records

- Read on your own:
  - CASE statement
  - Pragmas
  - Tips for program efficiency

What's on for today (12.1-12.5)

- Pointers
  - Creating pointers, dereferencing pointers
  - Assignment compatibility
  - Pointer arithmetic
  - NIL
- Static vs. dynamic allocation of memory
  - Activation records
  - Stack, stack pointer
- Dynamic variables: NEW(), DISPOSE()

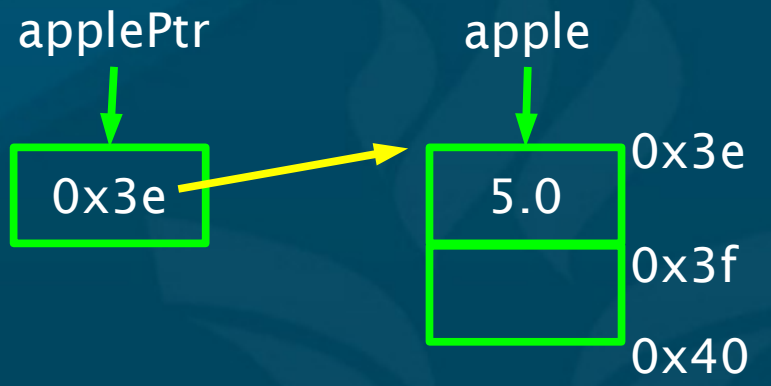
Pointers

- Values are stored in locations in memory (LOC)
- These locations are accessed by their ADDRESSes, which point to a spot in memory

```
TYPE ADDRESS = POINTER TO LOC;
```

- A pointer is a variable whose value is a memory address:

```
VAR
    applePtr : POINTER TO REAL;
    apple : REAL;
BEGIN
    apple := 5.0;
    applePtr := SYSTEM.ADR (apple);
```



Dereferencing pointers

- The last example shows how to make a pointer:

```
VAR
    applePtr : POINTER TO REAL;
    apple : REAL;
BEGIN
    apple := 5.0;
    applePtr := SYSTEM.ADR (apple);
```

- How do we get at the memory pointed to?

```
    applePtr^ := 4.0;     (* same as apple := 4.0 *)
```

  - (C syntax: *applePtr)
  - The "hat" operator ^ is called the dereferencing operator

Operations on pointers

- Pointers are compatible with SYSTEM.ADDRESS
  - Otherwise diff. pointer types not compatible
  - But can always use CAST
- Cannot compare (=, <, etc.) pointers
- Arithmetic operators (+, -, *, /) don't work
  - But in SYSTEM: ADDADR, SUBADR, DIFADR

```
ptr2 := ADDADR (ptr1, 10);
```

- NIL points to nothing at all
  - Handy for initializing pointers: ptr1 := NIL;
  - Dereferencing NIL raises sysException

Pointers and VAR parameters

- Passing a large data structure to a procedure can be wasteful:

```
PROCEDURE P1 (data : BigArray);
```

  - Call-by-value makes a local copy of the array
- We could pass a pointer to the array instead:

```
PROCEDURE P1 (dataPtr : POINTER TO BigArray);
```

  - Invoke the procedure with:

```
P1 ( SYSTEM.ADR (myData) );
```

- This is essentially how VAR parameters work:

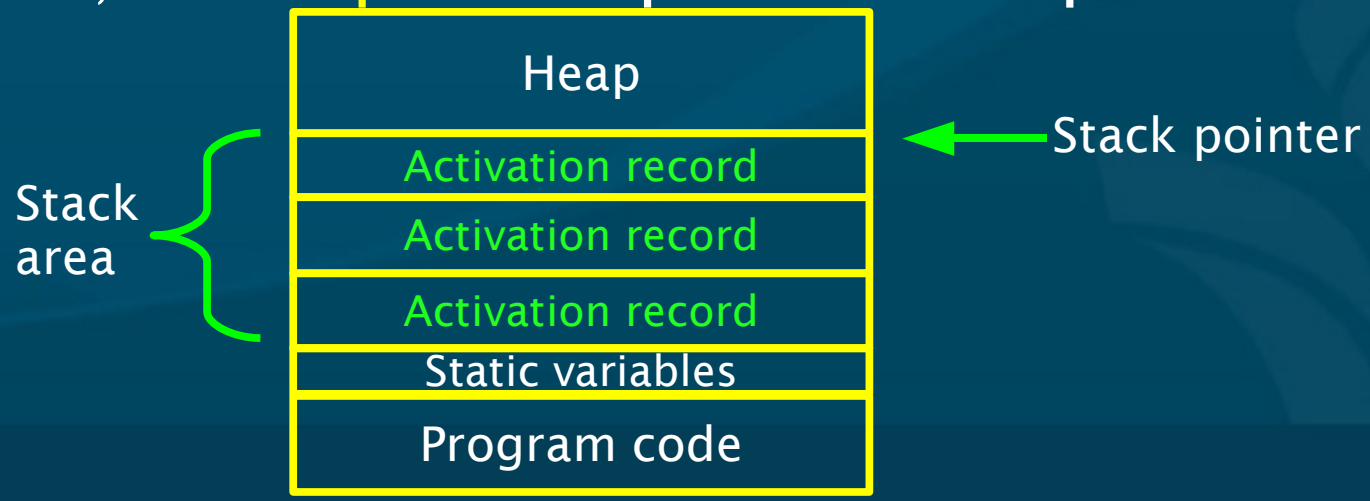
```
PROCEDURE P1 (VAR data : BigArray);
```

Static vs. dynamic memory

- Static variables are allocated at the beginning of the program run
  - Their size in memory is fixed at compile-time
  - VARs named in declaration section
- Dynamic variables are allocated during the running of a program
  - May also be deallocated during program
  - Size need not be predetermined
  - Reference them via pointers

Procedure activation records

- Whenever a procedure is invoked, memory is allocated for its static variables and parameters
  - This memory is called the activation record
  - One activation record for each invocation, not for each procedure declaration
- The stack is the area of memory for all activation records; stack pointer points to top of stack



Dynamic variables

- You can make your own dynamically allocated variables, using NEW() and DISPOSE():

```
VAR
    applePtr : POINTER TO REAL;
BEGIN
    NEW (applePtr);
```

  - Allocates memory for a REAL, and stores the address in applePtr

```
    DISPOSE (applePtr);
```

  - Deallocates the memory, and sets applePtr to NIL
- Dynamic variables are in the heap:
  - Open space for program to allocate/deallocate
- If heap is full, NEW sets pointer to NIL

A caution about pointers

- Pointers are a powerful tool and a quick way to shoot yourself in the foot:

```
VAR
    applePtr : POINTER TO REAL;
BEGIN
    applePtr^ := 5.0;          (* yipes! *)
```

  - Uninitialized pointer could point to anywhere in memory: dereferencing it can potentially modify any accessible memory!
    - Can crash older Windows; core dump in Unix

Review of today (12.1-12.5)

- Pointers
  - Creating pointers, dereferencing pointers
  - Assignment compatibility
  - Pointer arithmetic
  - NIL
- Static vs. dynamic allocation of memory
  - Activation records
  - Stack, stack pointer
- Dynamic variables: NEW(), DISPOSE()

TODO items

- Reading: through §12.7 for tomorrow
- Homework due tomorrow: 11.10 #10, 15, 16, 22
- No lab next week!

- Get cracking on your paper!