§12.6–12.7: Pointer Applications

25 Nov 2005
CMPT14x
Dr. Sean Ho
Trinity Western University

Reminders:

- ***journals** in folder*
- ***Homework** due*

Review of last time (12.1–12.5)

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

What's on for today (12.6–12.7)

- Endianness
- Pointer applications
  - Sorting using pointers
  - Resize-able dynamic array ADT
    - Type definition
    - Indexing the array
    - Creating a new array, resizing an existing one

A note about endianness

- Recall: CPU works on data one word at a time
  - 32-bit CPU: 1word = 4bytes
- 1 CARDINAL on a 32-bit machine takes up 1 word
  - 63_{10} = 00....0111111$_2$ = 00 00 00 3F$_{16}$
- But what order are the bytes within a word?
  - Big-endian (big end first): 00 00 00 3F
  - Little-endian (little end first): 3F 00 00 00
- Different CPUs choose different endianness
  - => byte-ordering "holy wars"

Big-endian vs. little-endian

- Big-endian (MSB: most significant byte first)
  - How we write numbers: 4,902
  - Can sort numbers lexicographically like strings
  - CPUs: Sun Sparc, IBM mainframes, SGI MIPS/IRIX, most PowerPC
  - "Network byte order" for IP (Internet)
- Little-endian (LSB: least significant byte first)
  - How we do arithmetic: 236 + 105 (carry)
  - CPUs: Intel x86, AMD, IA64/Linux
- No "one true way", just be aware + byte-swap

ALLOCATE and DEALLOCATE

- NEW() and DISPOSE() work on predeclared vars:

```
VAR
    myStudent : POINTER TO StudentRecord;
BEGIN
    NEW (myStudent);
    DISPOSE (myStudent);
```

- They use Storage.ALLOCATE(), DEALLOCATE():

```
NEW (myStudent);            (* is the same as *)
ALLOCATE (myStudent, SIZE (StudentRecord));
```

- ALLOCATE/DEALLOCATE work on ADDRESSes (pointer to any type); you specify how many LOCs

Pointer apps: sorting big records

- Bubble sort on array of REALs:

```
FOR surface := HIGH (list) TO 0 BY -1 DO
    FOR bubble := 0 TO surface-1 DO
        IF list [bubble] > list [bubble+1] THEN
            Swap (list [bubble], list [bubble+1]);
    END; END; END;
```

  - (other sorts are faster, but this is simple to code)
- Sorting involves lots of swaps:
  - Easy for array of reals, but
  - Wasteful for array of big records
- Solution: sort array of pointers to records

Bubble sort via pointers

```
PROCEDURE BubbleSort (
    VAR list : ARRAY OF POINTER TO bigRecord);
VAR
    surface, bubble : CARDINAL;
    tmp : POINTER TO bigRecord;
BEGIN
    FOR surface := HIGH (list) TO 0 BY -1 DO
        FOR bubble := 0 TO surface-1 DO
            IF Greater (list [bubble]^, list [bubble+1]^)
            THEN
                tmp := list [bubble];
                list [bubble] := list [bubble+1];
                list [bubble+1] := list [bubble];
            END; END; END;
END BubbleSort;
```

Define comparison of bigRecords

Swap pointers

- Swapping pointers easier than swap big records

Resize-able dynamic array ADT

- Normal arrays in M2 are statically allocated:
  - Need to know size at compile-time
  - Usually hard-code max length of arrays
    - Bigger than needed; wasteful
- Using pointers, we can make an array ADT that allows the user to resize it as needed:
  - Create (length: CARDINAL): DynArray
    - Make a new dynamic array of the given length
  - Resize (VAR list : DynArray, length)
    - Copy contents into a new array of given length
    - Throw away anything that doesn't fit

DynArray ADT

- Under the covers, a DynArray is just a record storing the length and a pointer to the start:

```
VAR DynArray = RECORD
        length : CARDINAL;
        start : ADDRESS;
```

- Index (access) the array via pointer arithmetic:

```
PROCEDURE Access (list: DynArray, idx: CARDINAL):
    POINTER TO ElementType;
VAR eltPtr : POINTER TO ElementType;
BEGIN
    RETURN CAST (POINTER TO ElementType,
        SYSTEM.ADDADR (list.start, idx * SIZE (ElementType));
END Access;
```

DynArray.Create()

- Create() by ALLOCATE()-ing a chunk of memory of given size:

```
PROCEDURE Create (length: CARDINAL): DynArray;
VAR
    newlist : DynArray;
BEGIN
    newlist.length := length;
    ALLOCATE (newlist.start, length * SIZE (ElementType);
    RETURN newlist;
END Create;
```

DynArray.Resize()

- Resize() by ALLOCATE()-ing a new array and copying contents into the new array.
  - Remember to DEALLOCATE() the old array

```
PROCEDURE Resize (
    VAR list : DynArray, newlength : CARDINAL);
VAR newptr : ADDRESS;
BEGIN
    ALLOCATE (newptr, newlength * SIZE (ElementType));
    Copy (list.start, newptr, MIN (list.length, newlength));
    DEALLOCATE (list.start, list.length * SIZE (ElementType);
    list.start := newptr;
    list.length := newlength;
END Resize;
```

Internal helper function: Copy()

- Copy contents of one block of memory into another block, one LOC at a time:

```
PROCEDURE Copy
    (src, dst: ADDRESS, len: CARDINAL);
VAR
    offset : CARDINAL;
    srcPtr, dstPtr : ADDRESS;
BEGIN
    FOR offset := 0 TO len-1 DO
        srcPtr := ADDADR (src, offset);
        dstPtr := ADDADR (dst, offset);
        dstPtr^ := srcPtr^;
    END;
END Copy;
```

Review of today (12.6-12.7)

- Endianness
- Pointer applications
  - Sorting using pointers (why?)
  - Resize-able dynamic array ADT
    - TYPE definition
    - Access (DynArray, idx)
    - Create (length): DynArray
    - Resize (VAR DynArray, newlength)
    - Other procedures needed to complete ADT?

TODO items

- Reading: through §12.10 for Mon
- Quiz ch11 on Mon
- No lab next week!

- Get cracking on your paper!