§12.1-12.5: Pointers

16 Nov 2007
CMPT14x
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Quiz08

- Contrast: alias, shallow copy, deep copy.
  - Draw or describe an example highlighting the differences
- What name does Python expect for the initializer (constructor) method in a user-defined class?
- Create a Python dictionary with three entries.
- Name at least three methods special to dictionaries.
- Write a Python code snippet that throws and catches an exception.

Review last time (Py tut §9.2)

- Namespaces
- Scope
  - New names add to local scope
  - Names outside local scope are read-only
    - Assigning to them makes a local copy
  - global command
- Backtracking: Knight's Tour

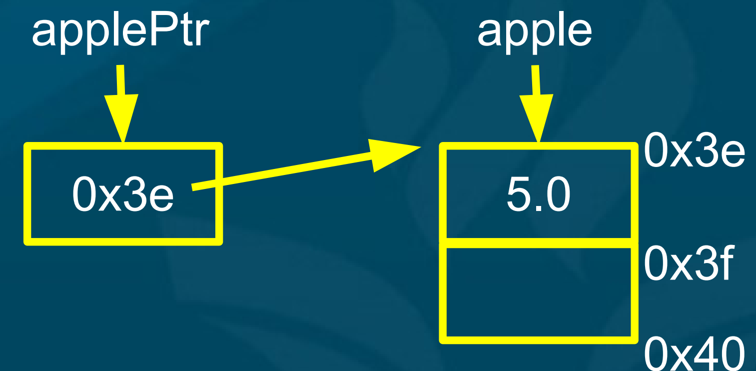
What's on for today (12.1-12.5)

- Pointers (in Modula-2 and C)
  - Creating pointers, dereferencing pointers
  - Assignment compatibility
  - Pointer arithmetic
  - NIL (in C: NULL)
- Static vs. dynamic allocation of memory
  - Activation records
  - Stack, stack pointer
- Dynamic variables: NEW(), DISPOSE()

Pointers

- Values are stored in locations in memory
- These locations are accessed by their addresses, which point to a spot in memory
- 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);
```

In C:

```
float apple;
float* applePtr;
apple = 5.0;
applePtr = &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

- Different pointer types are not compatible
  - But can always cast from one type to another:

```
float* applePtr;
int* pearPtr;
applePtr = (float*) pearPtr;
```

- NIL points to nothing at all
  - Handy for initializing pointers: ptr1 := NIL;
  - Dereferencing NIL raises sysException
  - In C, use NULL (which is just 0)

Pointers and C arrays

- An array in C is really just a pointer to a location in memory that stores consecutive entries of the array:

```
float appleSizes[4];
appleSizes[0] = 2.5;
```

  - Indexing into the array is really done by adding to the pointer to the head of the array:

```
appleSizes[2]
```

    - Is the same as:

```
*(appleSizes + 2)
```

Pointers and call-by-reference

- Pointers are how call-by-reference is done in C:

```
int increment (int* x) {          /* takes a pointer to an int */
    *x = *x + 1;
    return *x;
}
int x;
x = 5;
increment (&x);                   /* pass a pointer to x */
```

- In C++, can specify in the function definition:

```
int increment (int &x) {          /* call-by-reference */
    x = x + 1; ....
increment (x);
```

Static vs. dynamic memory

- Static variables are allocated at the beginning of the program run
  - Their size in memory is fixed at compile-time
  - Variables 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

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 (in Modula-2 and C)
  - Creating pointers, dereferencing pointers
  - Assignment compatibility
  - Pointer arithmetic
  - NIL (in C: NULL)
- Static vs. dynamic allocation of memory
  - Activation records
  - Stack, stack pointer
- Dynamic variables: NEW(), DISPOSE()

TODO

- No lab next week
- Midterm next Wed 21Nov: