CS 230
Programming Languages

11 / 10 / 2022

Instructor: Michael Eckmann

Today’s Topics

- Questions? / Comments?
- Pointers

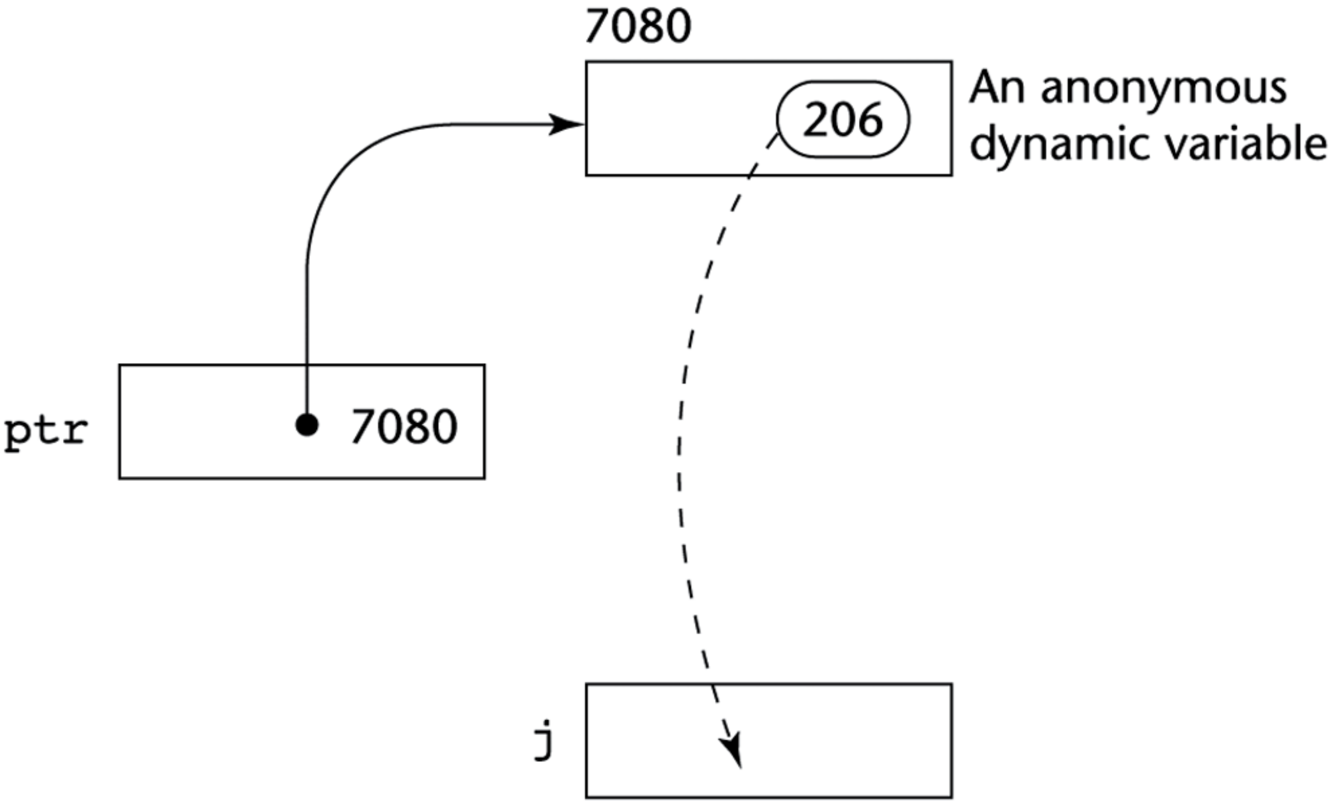
Pointers

- Pointer types
  - Store memory addresses or nil (no address.)
- For indirect addressing
- For dynamic storage management
  - Can access data at a particular memory address in the heap
- Provide a way to have data structures that shrink and grow during execution.
- Allocate space and deallocate space that the pointer points to

Pointers

- Using pointers
  - Pointer reference (the address)
  - Indirect reference (aka dereference) gives us what data is in the memory address that is stored in the pointer
    - Dereferencing is usually explicit (as in C, C++ with the * operator) but it is sometimes implicit (ala Ada)
- Dereferencing example
  - variable ptr is a pointer and contains the memory address 7080
  - Memory address 7080 contains the integer value 206
  - j is a variable of type integer.
  - j = *ptr (j is assigned the value that ptr is pointing to)

Pointers



Pointers

- Accessing fields of records via pointers to records
  - C++ (pointer to a struct)
  - (*ptr).field_name   (* dereferences, . accesses field)
  - ptr -> field_name   (-> does both)
    - Because this is a common operation, there's a shorthand

  - Ada
  - ptr.field_name
    - (because of implicit dereferencing acts just like -> in C++)

Pointers

- Problems
  - Dangling pointer (or dangling reference in those languages that don't have pointers)
  - Memory leakage
  - Did anyone read about these in the text or know what they are from prior knowledge?

Pointers

- Problems
  - Dangling pointer (or dangling reference in those languages that don't have pointers)
    - When a pointer contains the address of a variable that was deallocated already
    - example:
      - pointer p1 points to some heap dynamic variable
      - pointer p2 = p1 // make p2 also point there (an alias)
      - deallocate p1's heap dynamic variable
      - // now, p2 is a dangling pointer
  - If programmer cannot explicitly deallocate heap dynamic variables then there will be no possibility of dangling pointers.

Pointers

- Problems
  - Memory leakage
    - When a pointer's value changes (that is a different address is stored in it) and the address it was pointing to didn't have its memory deallocated
    - Garbage collection (frees deallocated memory for use)
      - C++ (done explicitly by programmer)
      - Java (done automagically)
    - Let me put some drawings on the board to represent dangling pointer and memory leakage problems.

Pointers

- Let's look at these problems in Ada, C/C++, Fortran 95, Java
- Ada
  - Pointers are of type **access**
  - Handles the dangling pointer problem by design
    - Implicit deallocation of memory at end of scope is done
    - Also allows explicit deallocation by the programmer via the deallocator: Unchecked_Deallocation

      (which can cause the dangling pointer problem)
  - Memory leakage
    - Nothing by design to prevent this

Pointers

- Let's look at these problems in Ada, C/C++, Fortran 95, Java
- C/C++
  - Both the dangling pointer problem and memory leakage exist in these languages
  - Can do pointer arithmetic
  - Can have pointers to any type and pointers to functions
  - One use of pointers is to pass variables by reference (so they may be changed within the function) --- contrast with pass by value, where the variable's value is copied to new temporary space in the function.
  - & is used to get the address of a variable
  - Examples of this stuff ...

Pointers

- C/C++

```
int *ptr;
int count=30, init=70;

ptr = &init;
count = *ptr;
```

What does this code do?

Pointers

- C/C++

```
int *ptr;     // declare a pointer to an int
int count=30, init=70;

ptr = &init;     // store the address of init in ptr
count = *ptr;
// dereference ptr and store what's in the address
// that ptr is pointing to

// so what value does count have?
```

Pointers

- C/C++ (pointer arithmetic)

```
int *ptr;     // declare a pointer to an int
int grades[ 100 ];
ptr = grades;     // grades is a “pointer” to grades[0] and now
// so is ptr
*(ptr + 1)
// here the + is pointer addition, so it actually adds
// the size of one int to ptr so that it dereferences
// (ptr + 1) to grades[1]
ptr[index]; // can use ptr like an array.
// What is the difference between ptr and grades?
```

Pointers

- Let's look at these problems in Ada, C/C++, Fortran 95, Java
- Fortran95
  - Has dangling pointer problem because allows explicit Deallocation command on a pointer
    - so how does that make dangling pointers possible?
  - Like Ada, pointers are implicitly dereferenced, but Fortran95 also provides a way to explicitly not dereference a pointer
    - What does this mean again?
    - Why might this be?

Pointers

- Let's look at these problems in Ada, C/C++, Fortran 95, Java
- Java
  - Anybody know whether we have dangling pointers/references or memory leakage problems in Java?

Pointers

- Reference types in C/C++ and Java and C#
  - C/C++ reference type is a special kind of pointer type
    - Used when declaring parameters to a function so that it is passed by reference
    - Formal parameter is specified with an &
    - But inside the function, it is implicitly dereferenced.
      - Makes code more readable and safer
    - Can get the same effect with regular pointers but code is less safe

Pointers

- Reference types in C/C++ and Java and C#
  - C/C++ reference type is a special kind of pointer type
    - Would there be any use for passing by reference using a constant reference parameter (that is, one that disallows its contents to be changed)?
  - Java
    - References replace C++'s pointers
      - Why?
    - Java references refer to class instances (objects).
    - No dangling references b/c implicit deallocation

Pointers

- Reference types in C/C++ and Java and C#
  - C#
    - Has both Java-like references and C++-like pointers.
    - Pointers are discouraged --- methods that use pointers need to be modified with the **unsafe** keyword.

Pointers

- Implementation of pointers
  - Pointers hold an address
- Solutions to dangling pointer problem
  - Tombstones
    - A tombstone points to (holds the address of) where the data is (a heap-dynamic variable.)
    - Pointers can only point to a tombstone (which in turn points to the actual data.)
    - What does this solve?
    - When deallocate, the tombstone is set to nil.

Pointers

- Solutions to dangling pointer problem (continued)
  - locks-and-keys
    - Pointers and variables need different implementation for this method
    - Pointers are ordered pairs of an integer key and an address.
    - Heap-dynamic variables
      - include a header cell that stores a lock value
      - and storage cell(s) for the variable itself
    - During allocation a lock value is calculated and placed in the key portion of the pointer AND the header cell of the variable.
    - Key and header cell are compared when the pointer is dereferenced and if they're the same it's a legal reference otherwise it is an illegal reference (which causes run-time error.)

Pointers

- Solutions to dangling pointer problem (continued)
  - Locks-and-keys (continued)
    - Multiple pointers may point to the variable but they all must have the same key.
    - When variable is deallocated (explicitly), the variable's header cell is changed to an illegal lock value (so no key will match it ever.)
  - Any other solutions you can think of to handle the dangling pointer problem?

Pointers

- Solutions to dangling pointer problem (continued)
  - Any other solutions you can think of to handle the dangling pointer problem?
    - Don't allow programmer to explicitly deallocate heap-dynamic variables. Like Java and LISP. Also like C#'s references (but not like C#'s pointers.)