Announcements

- HW 7 is due today
- HW 8 will be out tonight
- I will update Rainbow grades after class
  - HW 1-6
  - Exam 1-2
  - Quiz 1-7
- If you have any questions/concerns, let us know ASAP

Quiz 7 Question 2

A: All programs
S: All programs that run without errors
T: All programs accepted by a type-safe static type system

Quiz 7 Question 3

A: All programs
S: All programs that run without errors
T': All programs accepted by a type-unsafe type system

Last Class

- Type in C

Types
- Primitive types
- Composite types
  - Records (Structures), Variants (Unions), Arrays, Pointers

Today's Lecture Outline

- Control Abstraction
- Parameter Passing Mechanisms
  - Call by value
  - Call by reference
  - Call by value-result
  - Call by result
  - Call by name
  - Call by sharing

Control Abstraction and Parameter Passing

Read: Scott, Chapter 8.1-8.3
(lecture notes cover 8.3)
Abstraction

- Abstraction: hiding unnecessary low-level detail
- Data abstraction: types
  - Type integer is an abstraction
  - Type struct Person is an abstraction
- Control abstraction: subroutines
  - A subroutine abstracts away an algorithm
  - A subroutine provides an interface: name, argument types, return type: e.g., int binarySearch(int a[], int v)
- Classes/objects in OO, Abstract Data Types (ADTs) are a higher level of abstraction

Subroutines

- Other terms: procedures and functions
- Modularize program structure
- Argument: information passed from the caller to the callee (actual parameter or actual argument)
- Parameter: local variable in the callee, whose value is received from the caller (formal parameter)

Parameter Passing Mechanisms

- How does the caller pass information to callee?
- Call by value
  - C, Pascal, Ada, Algo68
- Call by reference
  - Fortran, C+, Pascal var params, sometimes Cobol
- Call by value-result (copy-in/copy-out)
  - Ada
- Call by result
  - Ada
- Call by name (outmoded)
  - Algo60
- Discussion applies to value model for variables

Parameter Passing Modes

- Most languages use a single parameter passing rule
  - E.g., Fortran, C
- Other languages allow different modes, in other words, programmer can choose different parameter passing rules in different contexts
  - E.g., C++ has two parameter passing mechanisms, Pascal too

Call by Value

- Value of argument is copied into parameter location

```pascal
m,n : integer;
procedure R(k,j : integer)
begin
  k := k + 1;
  j := j + 2;
end R;
...
m := 5;
n := 3;
R(m,n);
write m,n;
```

Output:
5 3

By Value:

```pascal
m,n : integer;
procedure R(k,j : integer)
begin
  k := k + 1;
  j := j + 2;
end R;
...
m := 5;
n := 3;
R(m,n);
write m,n;
```

Output:
5 3

Call by Reference

- Argument must be an l-value; l-value is passed to the parameter

```pascal
m,n : integer;
procedure R(k,j : integer)
begin
  k := k + 1;
  j := j + 2;
end R;
...
m := 5;
n := 3;
R(m,n);
write m,n;
```

Output:
6 5

Value update happens in storage of caller, while callee is executing

```pascal
m,n : integer;
procedure R(k,j : integer)
begin
  k := k + 1;
  j := j + 2;
end R;
...
m := 5;
n := 3;
R(m,n);
write m,n;
```

Output:
6 5
Call by Value vs. Call by Reference

- Call by value
  - Advantage: safe
  - Disadvantage: inefficient

- Call by reference
  - Advantage: more efficient
  - Disadvantage: potentially unsafe due to aliasing
  - Aliasing (memory aliasing) occurs when two or more different names refer to the same memory location
    - E.g., m in main, and k in R are aliases during the call to R

Aliasing: Call by Reference

```plaintext
y: integer;
procedure P(x: integer)
begin
  x := x + 1;
  x := x + y;
end P;
...
y := 2;
P(y);
write y;
```

Output: 6

No Aliasing: Call by Value

```plaintext
y: integer;
procedure P(x: integer)
begin
  x := x + 1;
  x := x + y;
end P;
...
y := 2;
P(y);
write y;
```

Output: 2

More Aliasing with Call by Reference

```plaintext
j,k,m : integer;
procedure Q(a,b : integer)
begin
  b := 3;
  a := m * a;
end Q;
...
s1: Q(m, k);
...
s2: Q(j, j);
```

Global-formal aliases: <m,a> <k,b> associations during call s1;
Formal-formal aliases: <a,b> during call s2.

Questions

- Aliasing is a powerful concept in programming languages
- Can you think of other examples of aliasing?
- Why memory aliasing is considered dangerous?
- Can you think of other ways for creating memory aliasing?

Variable Aliasing is Dangerous

- One part of the program can modify location through one alias, breaking invariants/expectations of other parts that use other aliases to same location

  In general, we cannot know whether x->f and y->f are aliases to the same location
  - We err on the safe side
  - Aliasing makes reasoning about code hard
  - Aliasing prevents compiler optimization
What are some defenses against unwanted modification through aliases?

- `const` parameters are an important paradigm in C/C++

```cpp
log(const huge_struct *r) {
  r->f = 0; // NOT OK
}
```

vs

```cpp
log(huge_struct * const r) {
  r->f = 0; // OK
}
```

---

```cpp
class C {
  int f;
public:
  int get() const {
    return f;
  }
  int set(int g) {
    f = g;
  }
};
```

---

What happens when someone uses an expression argument for a call by reference parameter?

- `(2*x)`?
Exercise

Write a program that produces different result when the parameter passing mechanism is call by reference and when it is call by value-result.

Call by Result

- Parameter is copied out into the actual argument when the subroutine exits.
- \( m, n \): integer;
- procedure \( R(k, j : \text{integer}) \)
  
  \[
  \begin{align*}
  k & := k + 1; \\
  j & := j + 2; \\
  \end{align*}
  \]

- Error in procedure \( R \): can’t use parameters which are uninitialized!

Call by Name

- An expression argument is not evaluated at call. It is evaluated within the callee, if needed.
- \( c \): array \([1..10]\) of integer;
- \( m \): integer;
- procedure \( R(k, j : \text{integer}) \)
  
  \[
  \begin{align*}
  k & := k + 1; \\
  j & := j + 2; \\
  c[m] & := c[m] + 2
  \end{align*}
  \]

- Error in procedure \( R \): can’t use parameters which are uninitialized!
Functions as Parameters

```pascal
m, k: integer;
procedure Q(x: integer;
  function f(): integer)
begin
  k, l: integer;
  l := f(); /* call to function param f */
end Q /* end of Q */
integer function F()
begin
  write k; /* which k is this? k or k? */
  ...
end F
Q(m, F);
```

Reference Model for Variables

- So far, discussion applied to the value model for variables
- What is the parameter passing mechanism in languages that use the reference model for variables? Neither call by value, nor call by reference make sense for languages with the reference model
  - Call by sharing

Immutability

- Immutability is a “defense” against unwanted mutation due to sharing
- In Scheme, methods are pure
- In Python, there are immutable datatypes
- In Java, `final` disallows re-assigning a variable. No `const`-like construct to protect the referenced object
  ```java
  final Point p = new Point();
  p = q; // NOT OK
  p.x = 0; r = p; r.y = 0; // ALL OK
  ```

Exercise

- Construct a program which prints different result when parameter passing mechanism is
  - Call by value
  - Call by reference
  - Call by value-result
  - Call by name