CS 536 Announcements for Wednesday, April 3, 2024

Last Time
- runtime environments
- runtime storage layout
- static vs stack allocation
- activation records
- what happens on function call, entry, return

Today
- parameter passing
- terminology
- different styles
  - what they mean
  - how they look on the stack
  - compare and contrast

Next Time
- runtime access to variables in different scopes

Example

#1 integer summation{integer max} [ #2 integer sum.
#3 integer k.
#4 sum = 0.
#5 k = 1.
#6 while k <= max [ #7 sum = sum + k.
#8 k++.
#9 ] #10 return sum.
#11 ] #12 void main{} [ #13 integer x.
#14 x = summation(4).
#15 write << x.
#16 ]
Parameter passing: terminology

R-value – value of an expression
L-value – value with with a location

pointer – a variable whose value is a memory address
aliasing – when two (or more) variables hold the same address

In **definition** of function/method/procedure

```cpp
void f(int x, int y, bool b) { . . . }
```

In **call** to function/method/procedure

```cpp
f(x + y, 7, true)
```

Types of parameter passing

**pass by value**
- when a procedure is called, the *values* of the actuals are copied into the formals

**pass by reference**
- when a procedure is called, the *address* of the actuals are copied into the formals

**pass by value-result**
- when a procedure is called, the values of actuals are passed
- when procedure is ready to return, final values of formals are copied back to the actuals

**pass by name**
- (conceptually) each time a procedure is called, the body of the procedure (the callee) is rewritten with the actual text of the actual parameters
- like macros in C/C++, but conceptually the rewriting occurs at runtime
Example: pass by value

```c
void f(int x, int y, int z) {
    x = 3;
    y = 4;
    z = y;
}

void main() {
    int a = 1, b = 2, c = 3;
    f(a, b, c);
    f(a+b, 7, 8);
}
```

Example: pass by reference

```c
void f(int x, int y, int z) {
    x = 3;
    y = 4;
    z = y;
}

void main() {
    int a = 1, b = 2, c = 3;
    f(a, b, c);
    f(a+b, 7, 8);
}
```
Example: pass by value-result

```c
void f(int x, int y, int z) {
    x = 3;
    y = 4;
    z = y;
}

void main() {
    int a = 1, b = 2, c = 3;
    f(a, b, c);
    f(a+b, 7, 8);
}
```

Parameter passing example

```c
class Point {
    Position p;
    ...
}

class Position {
    int x, y;
    ...
}

void doIt(Point pt, Position pos) {
    pos = pt.p;
    pos.x++;
    pos.y++;
}

void main() {
    Position loc;
    Point dot;
    // code to initialize Point dot with position (1, 2)
    // code to initialize Position loc at (3, 4)
    doIt(dot, loc);
}
```

In Java, `loc` & `dot` are references to objects (in the heap)
In C++, `loc` & `dot` are objects (in the AR of `main`)
Parameter passing example (cont.)

Pass by value in Java

Pass by value in C++  Pass by reference in C++

What are the \((x,y)\) coordinates of \(\text{dot}\) and \(\text{loc}\) after the call to \(\text{doIt}\)?

<table>
<thead>
<tr>
<th></th>
<th>Pass by value (Java)</th>
<th>Pass by value (C++)</th>
<th>Pass by reference (C++)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{dot})</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{loc})</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Aliasing and parameter passing

How aliasing can happen

• via pointers (in pass by value) – aliasing of actuals and formals
  
  ```java
  doiT(dot, loc); // in Java
  ```

• when a global variable is passed by reference
  
  ```java
  int t = 0;

  void h(int x) {
    x = 7;
    t = 4;
  }

  void main() {
    h(t);
  }
  ```

• when a parameter is passed by reference more than once
  
  ```java
  void f(int x, int y, int z) {
    x = 3;
    y = 4;
    z = y;
  }

  void main() {
    int a = 1, b = 2, c = 3;
    f(a, a, b);
  }
  ```

What happens in pass by value-result?
Code generation and parameter passing

Efficiency considerations (calls, accesses by callee, return)

Pass by value
- copy values into callee's AR
- callee directly accesses AR locations

Pass by reference
- copy addresses into callee's AR
- access in callee via indirection

Pass by value-result

Handling objects

```java
class Point {  
    Position p;  
    int x, y;  
    // ...  
}

class Position {  
    int x, y;  
    // ...  
}

void doIt(Point pt, Position pos) {  
    pos = pt.p;  
    pos.x++;  
    pos.y++;  
}

void main() {  
    Position loc;  
    Point dot;  
    // ... initialize dot with position (1, 2)  
    // ... initialize loc at (3, 4)  
    doIt(dot, loc);  
}
```

In Java, `loc` and `dot` hold the addresses of objects

In C++, `loc` and `dot` are objects in the stack
Compare and contrast

Pass by value
- no aliasing
- easier for static analysis
- called function (callee) is faster

Pass by reference
- more efficient when passing large objects
- can modify actuals

Pass by value-result
- more efficient than pass by reference for small objects
- if no aliasing, can be implemented as pass by reference for large objects