Introduction to Compiler Design

Lesson 14:

Parameter Passing

Roadmap

- Last time
  - Discussed runtime environments
  - Described some conventions for assembly
    - Function call/return involve stack manipulations
    - Dynamic memory via a heap
- Now
  - Propagating values from one function to another

Outline

- Parameter Passing
  - Different styles
  - What they mean
  - How they look on the stack

Vocabulary

- Define several terms that are needed for talking about parameters
- We’ve already used some of them previously

L- and R- Values

- L-Value
  - A value with a place of storage
- R-Value
  - A value that may not have storage

```
b = 2;
a = 1;
a = b+b;
```

Memory references

- Pointer
  - A variable whose value is a memory address
- Aliasing
  - When two or more variables hold same address

Parameter Passing

- In the procedure definition:
- **void v(int a, int b, bool c) { … }**
  - Vocabulary
    - Formals / formal parameters / parameters
- At a call site:
- **v(a+b, 8, true);**
  - Vocabulary
    - Actuals / actual parameters / arguments

Types of Parameter Passing

- We'll talk about 4 different varieties
  - Some of these are more used than others
  - Each has its own advantages / uses

Pass by Value (aka Call by Value)

- When a function is called
  - *Values* of actuals are copied into the formals
  - C and Java always use pass by value

```
void fun(int a){
    a = 1;
}
void main(){
   int i = 0;
   fun(i);
   print(i);
}
```

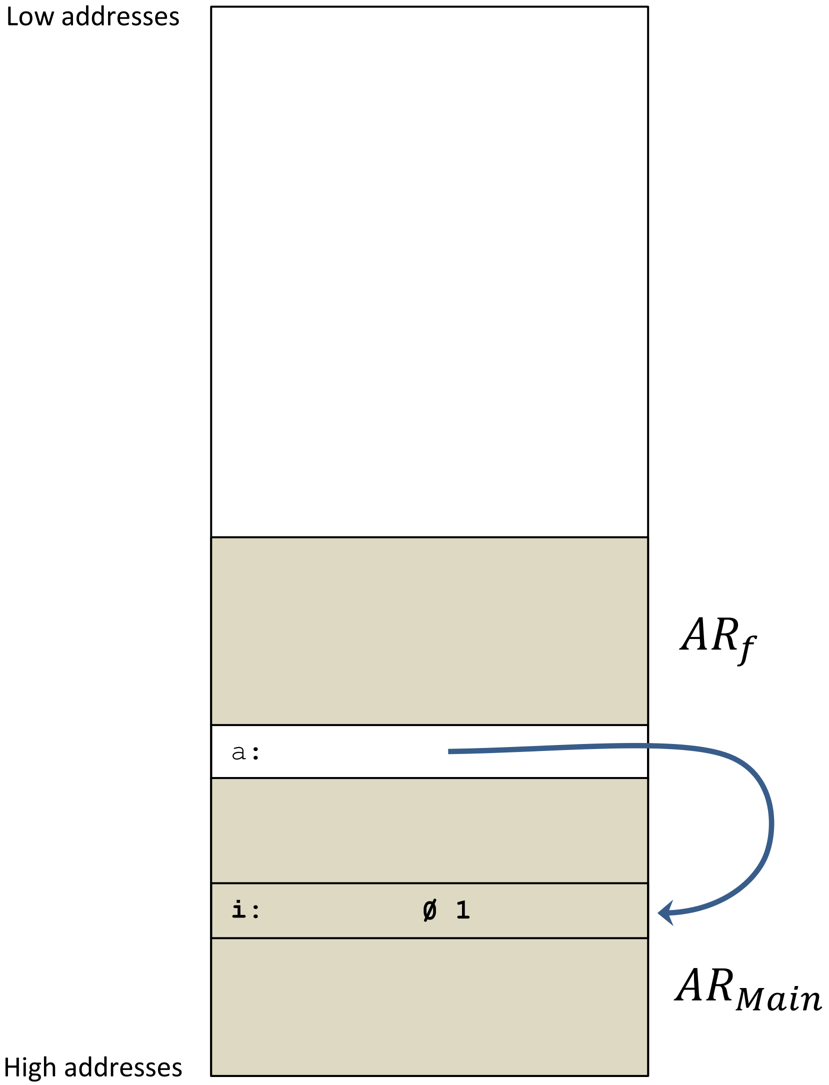
Pass by Reference (aka Call by Reference)

- When a function is called
  - The address of the actuals are *implicitly* copied

```
void f(int a){
   a = 1;
}
void main(){
   int i = 0;
   f(i);
   print(i);
}
```

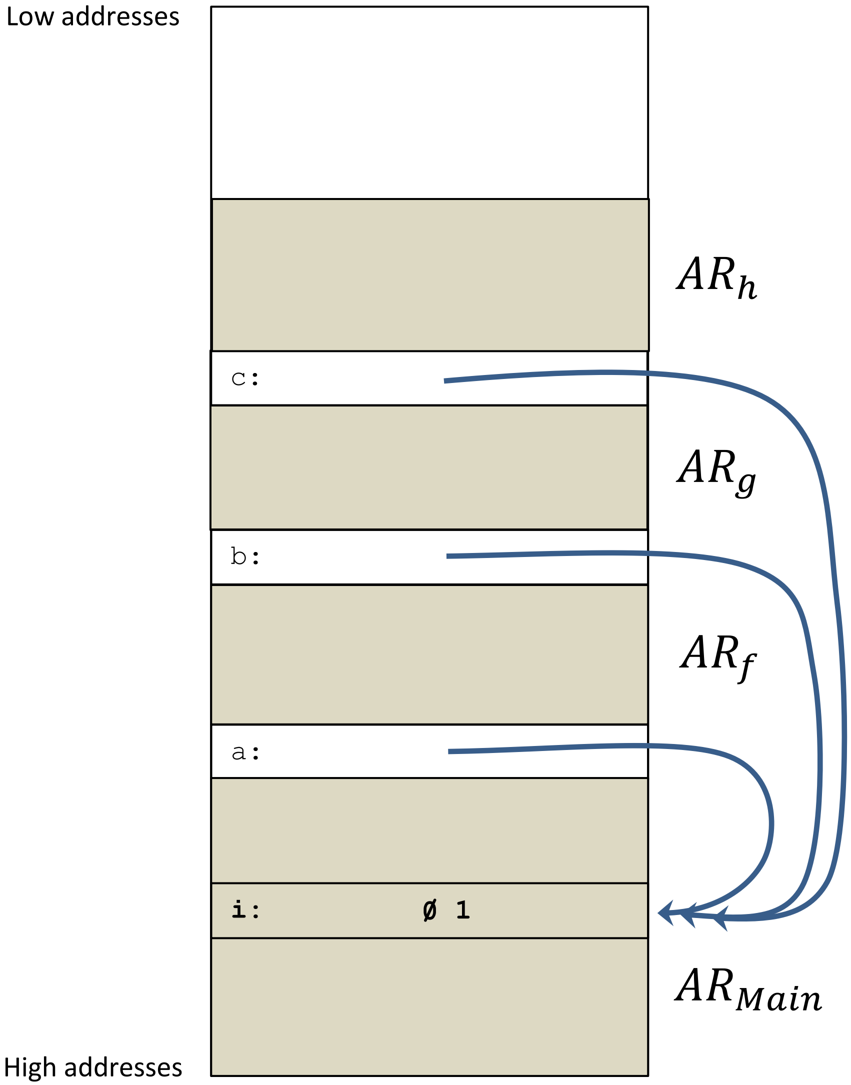
Pass by Reference (aka Call by Reference)

```
void f(int a){
    a = 1;
}
void main(){
    int i = 0;
    f(i);
    print(i);
}
```



Pass by Reference (aka Call by Reference)

```
void h(int c){
    c = 1;
}
void g(int b){
    h(b);
}

void f(int a){
    g(a);
}

void main(){
    int i = 0;
    f(i);
    print(i);
}
```



Language Examples

- Pass by value
  - C and Java
- Pass by reference
  - Allowed in C++ and Pascal
  - In C, can be simulated using pointers (address-valued parameters)

```
void fun(int& a){
    a = 1;
}
void main(){
    int i = 0;
    fun(i);
    print(i);
}
```

```
void fun(int* a){
    *a = 1;
}
void main(){
    int i = 0;
    fun(&i);
    print(i);
}
```

Wait, *Java* is Pass by Value?

- All non-primitive L-values are pointers

```
void fun(int a, Point p){
    int a = 0;
    p.x = 5;
}
void main(){
    int i = 0;
    Point k = new Point(1, 2);
    fun(i,k);
}
```

Java: Pass by Value

```
public static void main( String[] args ){
  Dog aDog = new Dog("Max");
  foo(aDog);

  if (aDog.getName().equals("Max")) {
       System.out.println( "Java passes by value." );
  } else if (aDog.getName().equals("Fifi")) {
        System.out.println( "Java passes by reference." );
  }
}

public static void foo(Dog d) {
    d.getName().equals("Max");
    d = new Dog("Fifi");
    d.getName().equals("Fifi");
}
```

For aDog in main, aDog.getName() equals “Max”. Execution goes down the then branch.

Changes value of d in foo, but leaves aDog in main unchanged

Pass by Value-Result

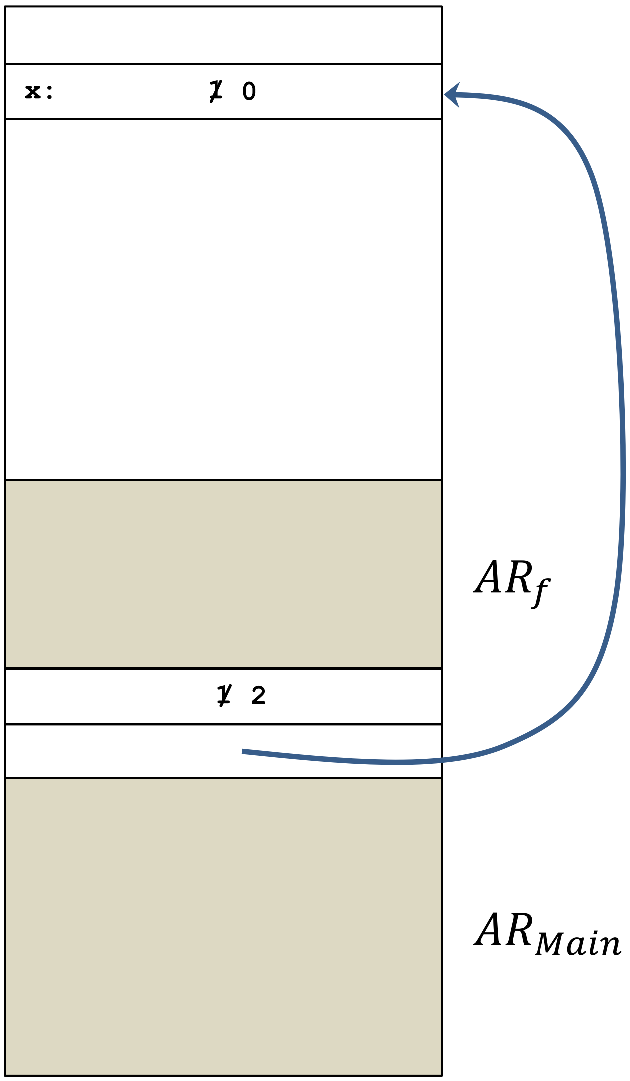
- When a function is called
  - Value of actual is passed
- When the function returns
  - Final values are copied back to the actuals
  - The actual must be a variable, not an arbitrary expression
- Used by Fortran IV, Ada
  - As the language examples show, not very modern

Pass by Value-Result – Example 1

```
int x = 1;      // a global variable

void f(int & a)
{
  a = 2;
  // when f is called from main, a and x are aliases
  x = 0;
}

main()
{
  f(x);
  cout << x; // 0 with call by value and call by reference
             // 2 with call by value-result
}
```

Pass by Value-Result – Example 1

```
int x = 1;        // a global variable

void f(int & a)
{
   a = 2;
   // when f is called from main
   // a and x are aliases
   x = 0;
}

main()
{
   f(x);
   cout << x;   // 0 with call by value
               // and call by reference
              // 2 with call by value-result
}
```



Pass by Value-Result – Example 2

```
void f(int &a, int &b)
{
   a = 2;
   b = 4;
}

main()
{
   int x;
   f(x, x);
   cout << x; // Undefined: different output with
              // different compilers
}
```

Pass by Name (aka Call by Name)

- Conceptually works as follows:
  - When a function is called
    - Body of the callee is **rewritten** with the **text** of the argument
  - Like macros in C/C++, but conceptually the rewriting occurs at runtime

Call by Need (aka Lazy Evaluation)

```
int f(x,y)
  { return x+y;}

main()
{
  int x = f(5, 6); // x=“5+6”
  cout << x;       // x is now evaluated
}
```

Implementing Parameter Passing

- Let’s talk about how this is actually going to work in memory

Bad Uses of R-Values

- Can prevent programs that are valid in pass by value from working in pass by reference
  - Or when a C++ formal is changed from **int** to **int&**

```
void f(int a){…} ⇒ void f(int& a){…}
f(x);        // OK
f(3);        // not OK
f(x + 3);    // not OK
```

  - Literals and non-trivial expressions do not have locations in memory
- The type checker would catch bad uses of R-values

Efficiency Considerations
[Calls, Accesses by Callee, Return]

- Pass by value
  - Copy values into AR (slow)
  - Access storage directly in function (fast)
- Pass by reference
  - Copy address into AR (fast)
  - Access storage via indirection (slow)
- Pass by value-result
  - Strictly slower than pass by value
  - Also need to know where to copy locations back

Object Handling

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

void main(){
   Position loc;
   Point dot;
   // … initialize loc with
   // x=1,y=2
   // … initialize dot with loc
   alter(dot, loc);
}
```

```
•  class Point{
•      Position p;
   }

•  class Position{
       int x, y;
•  }
```

- In Java, loc and dot hold the addresses of objects (addresses in the heap)
- In C++, loc and dot are objects in the stack; no (extra) indirection needed

Roadmap

- We learned about parameter-passing conventions
  - Semantics of by-value, by-reference, by-value-result, by-name
  - How the code must traverse the stack for each of the conventions

- Next
  - Runtime access to variables in different scopes