

Parameter Passing

Roadmap

Last time

- Discussed runtime environments
- Described some conventions for assembly
 - Functions via stack
 - Dynamic memory via a heap

Today

- Propagating values from one function to another

Outline

Parameter Passing

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

Vocabulary

Define a couple of terms that are helpful to talk about parameters

We've already obliquely talked about some of these



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 definition:

void v(int a, int b, bool c) { ... }

– Terms

- Formals / formal parameters / parameters

In call:

v(a+b,8,true);

– Terms

- 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 it's own advantages / uses

Pass by Value

On function call

- *Values* of actuals are copied into the formals
- C and java always pass by value

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

Pass by Reference

On function call

- The address of the actuals are *implicitly* copied

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

Language Examples

Pass by value

- C and Java

Pass by reference

- Allowed in C++ and Pascal

Wait, *Java* is Pass by Value?

All non-primitive L-values are pointers

```
void fun(int a, Point p){  
    a = 1;  
    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");
}
```

Demo

Pass by Value-Result

When function is called

- Value of actual is passed

When function returns

- Final values are copied back to the actuals

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 ref, 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

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++

Call-by-need / 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 actually is going to work in memory



Let's draw out the memory

```
int g;  
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);  
}
```

Consider pass-by-value and
pass-by reference

Bad use of R-Values

Can prevent programs that are valid in pass by value from working in pass by reference

- Literals (for example) do not have locations in memory

We will rely on the type checker to catch bad use of R-values

Let's draw out the memory again

```
int g;  
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, g);  
    f(a+b, 7, 8);  
}
```

Consider pass by value-result
and pass by name

Efficiency Considerations

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 are pointers to objects (on the heap)

In C++, loc and dot are objects with no indirection (on the stack)

Roadmap

We learned about parameter passing

- By-value, by-reference, by-value-result, by-name
- How values traverse the stack

Next time

- Allocating variables