CS212

Pointers and SaM

Fall 2005

Announcements

- P3
- FAQ:
  - change partners for P3? ***Yes***
  - change partners during P3? ***No***
  - allowed to our solution? ***yes (not mandatory)***
- Dirty Harry and `Math.random()`

Pointer Recap

- Pointer: (see video)
  - variable type
  - points to another location in memory
- Operators:
  - **&**: *get address*
  - *****: *dereference* (get value: two kinds...***left***, ***right***)
- Pointer values:
  - **left** (as expression, LHS of assignment)
    - get address pointed to
    - ex)
```
int* p; int v; p = &v; print(p);
*p = 10; // value of v?
```
  - **right** (RHS of assignment)
    - get value from address pointed to
    - ex)
```
int k; k = *p;
print(k); print(*p); // output?
```


Variables and SaM

- Scope:
  - Global variables
  - Local variables
  - Parameters
- Concerns:
  - what goes on heap?
  - what goes on stack?
    - relative to FBR?
    - relative to program?
- Organization:
  - stack
    - absolute addressing
    - relative addressing
  - heap
    - allocation
    - accessing
    - deallocation

Absolute Address (Stack)

- Global variables in program:
  - To **store** a value ***v*** at location ***i***:
    - **PUSHIMM *v***: Stack[SP] ← ***v***; SP++
    - **STOREABS *i***: Stack[***i***] ← Stack[SP-1]; SP--
  - To **retrieve** a value ***v*** from location ***k***:
    - **PUSHABS *k***: Stack[SP] ← Stack[***k***]; SP++
- Java-style example:

```
int rv;
int x;
int y;
x = 10;
y = 20;
rv = x + y;
return rv;
```

```
ADDSP 3
PUSHIMM 10
STOREABS 1
PUSHIMM 20
STOREABS 2
PUSHABS 1
PUSHABS 2
ADD
STOREABS 0
ADDSP -2
STOP
```

Note: Bali assignments are *expressions*, which means...? (templates are different!)

Relative Address (Stack)

- Local variables in functions
- Instructions:
  - To store a value ***v*** at location ***i***:
    - **PUSHIMM *v***: Stack[SP] ← ***v***; SP++
    - **STOREOFF *i***: Stack[***i***+FBR] ← Stack[SP-1]; SP--
  - To retrieve a value ***v*** from location ***k***:
    - **PUSHOFF *k***: Stack[SP] ← Stack[***k***+FBR]; SP++

Heap

- Allocation:
  - **malloc(*int*)**
  - returns **void*** pointer to first cell in heap
    - what does that mean for stack?
    - what does that look like?

```
{ int* p; }
{ p = <int*> malloc(3); ... }
```

Stack Heap

Heap (continued)

- Accessing:
  - ***(*address*)**
    - the * deferences the location
    - the address can be an arithmetic expression (*pointer*↔*pointer*, *pointer*↔*int*)
    - ***pointer + int*** is common ***int*** called ***offset***
  - examples:

```
*(p+0) *(p+1) *(p+2)
```

  - L-value: ***(*address*) = *expr***
  - R-value: ***(*address*)** and ***name=*(*address*)***

Heap (continued)

- Deallocation:
  - Must free allocated space
    - Stack?
    - Heap?
  - Heap uses **free**
    - **free(*voidpointer*)**
    - returns 0

```
free(<void*>p);
```


Samcode for Pointers and Heap

- Pointer declaration:
  - declaration ***type* var***
  - example) **int* p;**
    - **ADDSP 1**
  - Could use **PUSHIMMMA 0** for "default" memory address
    - why bother with **PUSHIMMMA**?
    - assumes all pointers have default of **null**
    - what's **null**?
    - do Bali variables get defaults?

Samcode P&H (continued)

- Storing an address:

```
p = &a ;
```

  - Global:

```
PUSHIMMMA a
DUP
STOREABS p
ADDSP -1
```

  - Local:

```
PUSHFBR
PUSHIMM a
ADD
DUP
STOREOFF p
ADDSP -1
```

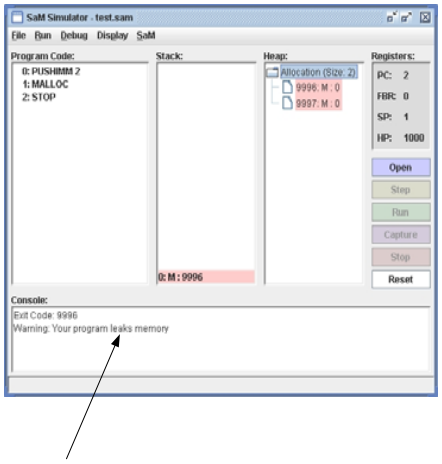

Samcode P&H (continued)

- Allocate heap:

```
malloc(2);
```

  - Push space for 2 cells
  - Push address of 1st cell on Stack
  - Code:

```
PUSHIMM 2
MALLOC
```



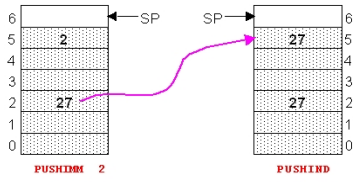
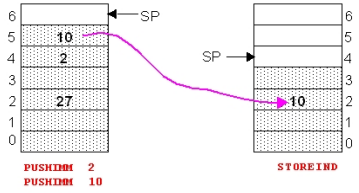
Samcode P&H (continued)

- Freeing memory:
  - ***Memory leak***:
    - allocated memory that remains on heap after program ends
    - bad! memory "leaks" away from total available
    - responsibility of programmer to free heap allocated memory
      `free(voidpointer)`
  - Side note:
    - what's automatic garbage collection?
  - Samcode:
    - push address of object
    - **`FREE`**
    - **`PUSHIMM 0`**

Samcode P&H (continued)

- Accessing values on heap:
  - operations:
    - putting (L-value or R-value?)
    - getting (L-value or R-value?)
  - to access heap cells:
    - need to get to address of heap: **`*p`**
    - need to add to address of heap: **`*(p+2)`**
  - questions to answer with Samcode:
    - where/how do you get heap address?
      Answer: _____
    - once you have address, how to put/get?
      (see next slide)

Samcode P&H (continued)

Instruction	Example	Demonstration
PUSHIND $V_{SP-1} \leftarrow V_{V_{SP-1}}$	PUSHIMM 2 PUSHIND	
STOREIND $V_{V_{SP-2}} \leftarrow V_{SP-1}$ $SP \leftarrow SP-2$	PUSHIMM 2 PUSHIMM 10 STOREIND	

Samcode P&H (continued)

- Store in heap (L-value, **`*(ep)=e1`**):

```
code_for_e1   // push e1
DUP           // duplicate for result of assignment
code_for_ep   // push ep (could involve arith expr)
SWAP          // swap order of one copy of ep and e1
STOREIND      // V[V[below]] <- V[top]
ADDSP -1      // deallocate result of assignment
```

- Get from in heap (R-value, **`*(e+o)`**):

```
code_for_e              // heap
PUSHIMM offset_of_cell  // offset
ADD                     // h+o
PUSHIND                 // V[top] <- V[V[top]]
```