

# CS212

## "C Practicum"

Heap  
Pointers  
Fall 2005

1

## Announcements

- P2b style grades tonight
- P3 up next!

2

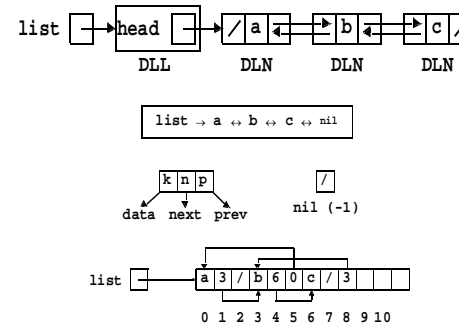
## Overview

- The Heap:
  - Simulating Objects
  - JVM and The Heap
  - SaM's Heap
- Pointers:
  - definition
  - declaration
  - addressing
  - dereferencing
  - examples

3

## Simulating Objects

An example:

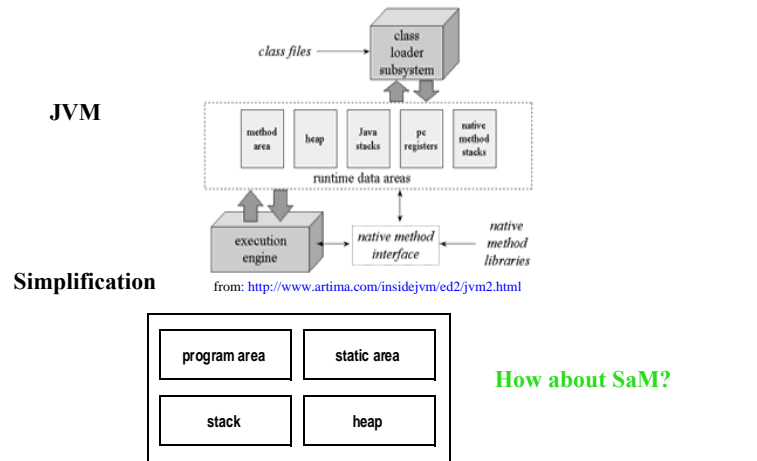


Example demonstrates needs to dynamically allocate memory that we can access and remove as needed.

So, where do the objects in memory go....?

4

## JVM and OOP



5

## SaM's Heap

- Allocation of heap in SaM:
  - **object** = chunk of memory in heap
  - address = beginning of chunk
  - no functions in Heap!
- Object starts at first cell
- Fields in cells
- Can use some cells for other info...

6

## Stack ↔ Heap?

- How to connect object to Stack?
  - when allocating object, store first cell address on Stack
  - refer to fields of object with that address
- Picture:

7

## Example Samcode for Heap

- Use memory allocation Samcode instruction: **MALLOC!**
  - pops top of Stack
  - allocates that **number of cells** in heap
  - pushes the address of the first cell onto stack
  - SP++
- Example (**heap.sam**):

```

PUSHIMM 1 // 1 cell to allocate
MALLOC // pop 1 and allocate 1 cell in heap
PUSHIMM 3 // 3 cells to allocate
MALLOC // pop 3 and allocate 3 cells in heap
PUSHIMM 0 // no cells to allocate
MALLOC // pop 0 and allocate no cells in heap
FREE // deallocate last "object"
FREE // deallocate second "object"
FREE // deallocate first "object"
PUSHIMM 0 // push dummy return value
STOP // cease execution
    
```

8

## Bali and C

- Borrowing from C!
  - Syntax roughly equivalent to Java
  - <http://computer.howstuffworks.com/c.htm> (and others)
  - See Tools lecture... Cygwin and C compiler
- Need to deal with pointers...why?
  - pointers part of grammar
  - interested in *dynamically allocated memory*
    - what is it? (see example below)
    - why bother?

```
Example) Java objects
Person p = new Person();
Person q = p;
q.name = "Dimmu Borgir"
```

9

## C Pointers

- Declare variables as pointer types
  - *Type\* p*
  - *p* is *pointer to type Type*
  - how does SaM represent "memory type"?
- Example) *int\* p*
  - *p* is pointer to an *int*
  - sometimes you see *int \*p* and *int p\**
- What does *p* actually store?
  - When initialized, *p* points to unknown location
  - After assigned, *p* points to another variable
- Picture of *p*:

10

## Address Operator (&)

- How to store value in *p*?
  - Address operator *&*:
    - *address of ...*
    - *location of ...*
  - Yields an integer value:
    - the address of some variable is somewhere in memory
    - think of Stack cell addresses
- Example)

```
/* address.c */
int main(int argc, char* argv[]) {
    int* p; // declare pointer p to int
    int i; // declare int i
    p = &i; // store address of i in p

    printf("%s%i\n", "&i: ", &i);
    printf("%s%i\n", "p: ", p);
    printf("%s%i\n", "&p: ", &p);
}
Output: &i: -4196924
p: -4196924
&p: -4196920
```

Picture to relate *p* and *i*?

11

## Dereferencing/Indirection Operator (\*)

- Reminders:
  - *Type\* var*
  - *var* points to an address of *Type*
- *Dereferencing operator: \**
  - *\*p* dereferences *p* to access location to which *p* points
  - Since value stored in *p* is an address (memory location), *\*p* is the indirect value of *p* (the value stored at that address).
- Complication:
  - *\*p* depends on "side" of assignment: *L-value, R-value*
  - brief examples:
    - \*p = 10;*
    - q = \*p + 10;*

12

## L-Value (**\*p = RHS**)

- **L-value**:
  - dereference the address that **p** stores, and put the RHS of the assignment there
  - abbreviated: **put** RHS where **\*p** points

- Example:

```
int* p; // declare pointer p to int
int v; // declare int v
p = &v; // store address of v in p
// or say, "p points to v"
*p = 10; // dereference p and put 10 in that address
// or say, "store 10 where p points"
```

- Picture:

13

## R-Value (**LHS = \*p**)

- **R-value**:
  - dereference the address that **p** stores, and retrieve the value stored there
  - abbreviated: **get** the value from where **\*p** points

- Example:

```
int* p; // declare pointer p to int
int v; // declare int v
p = &v; // store address of v in p
*p = 10; // store 10 where p points
int x; // declare int x
x = *p+2; // retrieve value at address pointed to by p
// and add 2
printf("%i",*p); // is this L- or R-value?
```

- Picture:

14

## Aliases

- Akin to OOP:
  - multiple pointers pointing to same location
  - changing the stored value affects both "aliased" pointers

```
/* alias example */
int main(int argc, char* argv[]) {
    int i, *p, *q;

    p = &i ;
    *p = 30 ;
    q = p ; /* what happens here? */
    *q = 40 ;

    printf("%s%i\n",**p: ",*p);
    printf("%s%i\n",**q: ",*q);
    printf("%s%i\n",i: ", i);
}
/* Output? */
```

Picture?

15

## Two More Details

- Dereferencing and addressing are inverse operations:

```
/* identity example */
int main(int argc, char* argv[]) {
    int i, x;
    x = 50;
    i = *x;
    printf("%s%i\n", "p: ",i);
    printf("%s%i\n", "p: ",x);
}

```

- **NULL** pointer:

```
/* NULL example */
#include "stddef.h"
int main(int argc, char* argv[]) {
    int *p, x;
    printf("%s%i\n", "p:",p);
    p = NULL;
    printf("%s%i\n", "p:",p);
}

```

16