Announcements

Course policies:

http://cs.illinois.edu/class/cs225

For general assistance:

http://piazza.com/class#spring2013/cs225

MP1 grading run for late adders tonight, 11:59p.

MP2 available, due 2/5, 11:59p. EC: 1/29, 11:59p.

Stack vs. Heap memory:

```
void fun() {
  string s = "hello!";
  cout << s << endl;
}
int main() {
  fun();
  return 0;
}
```

System allocates space for s and takes care of freeing it when s goes out of scope.

Data can be accessed directly, rather than via a pointer.

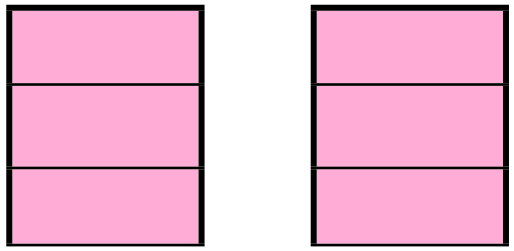
```
void fun() {
  string * s = new string;
  *s = "hello?";
  cout << *s << endl;
  delete s;
}
int main() {
  fun();
  return 0;
}
```

Allocated memory must be deleted programmatically.

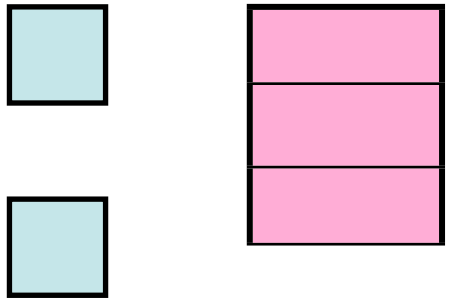
Data must be accessed by a pointer.

Pointers and objects:

```
face a, b;
… // init b
a = b;
a.setName("ann");
string s = b.getName();
```



```
face * c, * d;
… // init *d
c = d;
(*c).setName("carlos");
string t = d->getName();
```

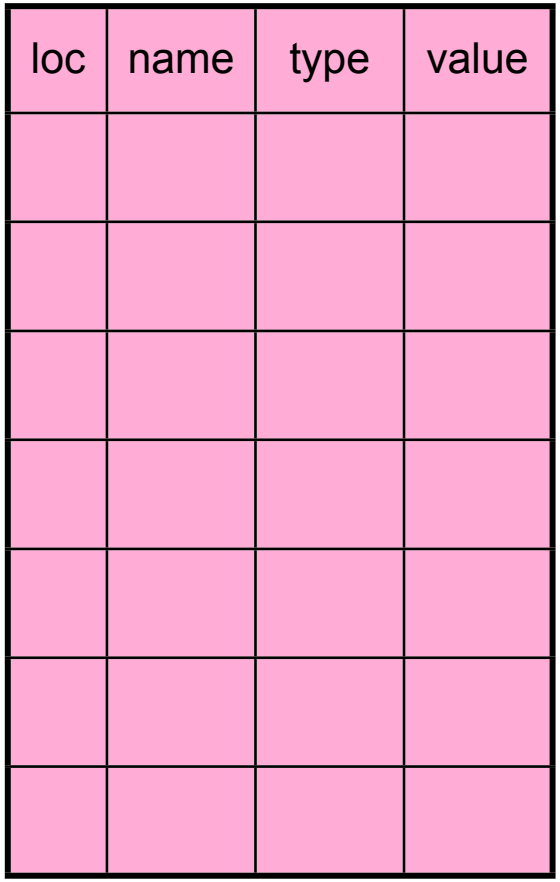


```
class face {
public:
     void setName(string n);
     string getName();
     …
private:
     string name;
     PNG pic;
     boolean done;
};
```

Arrays: static (stackic)

```
int x[5];
```

Stack memory



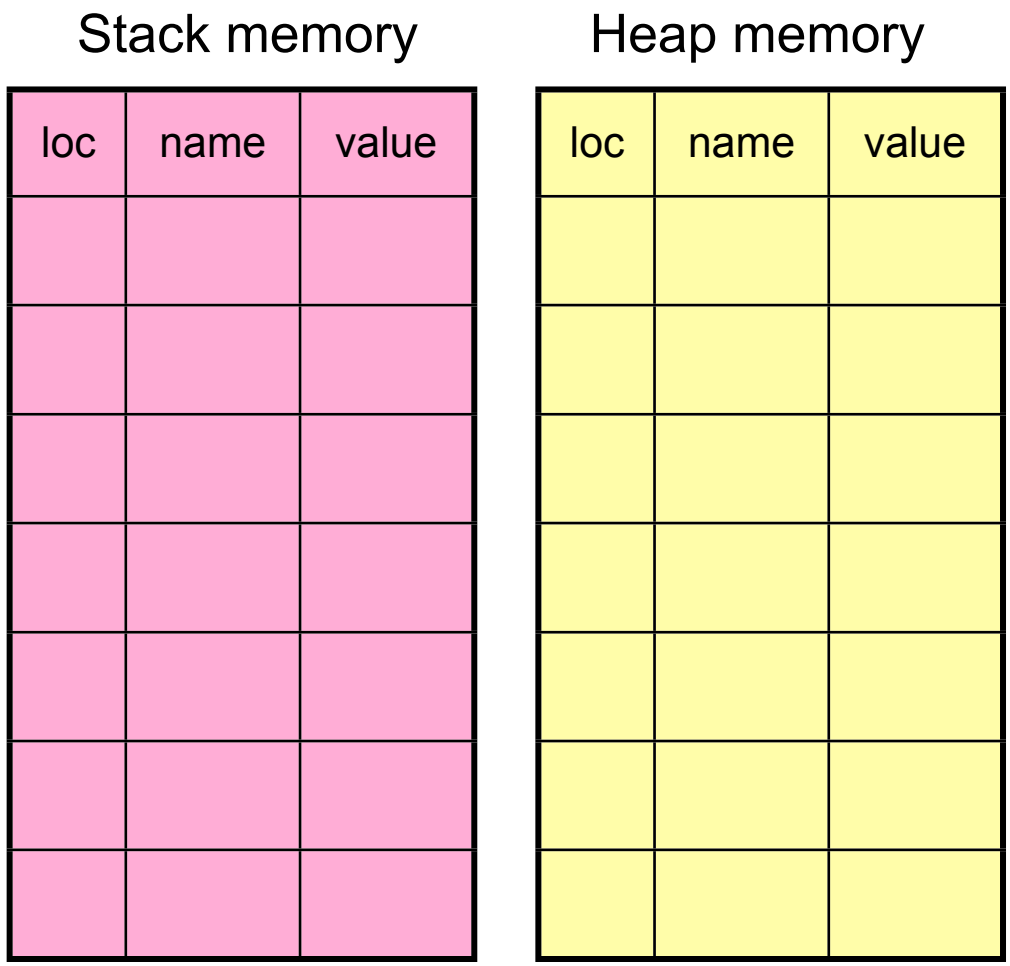
Arrays: dynamic (heap)

```
int * x;

int size = 3;
x = new int[size];

for(int i=0, i<size, i++)
    x[i] = i + 3;

delete [] x;
```



A point to ponder: How is my `garden` implemented?

```
class garden{
public:
…
// all the public members
…
private:
   flower ** plot;
   // other stuff
};
```

Option 1:

Option 2:

Option 3:

Option 4:

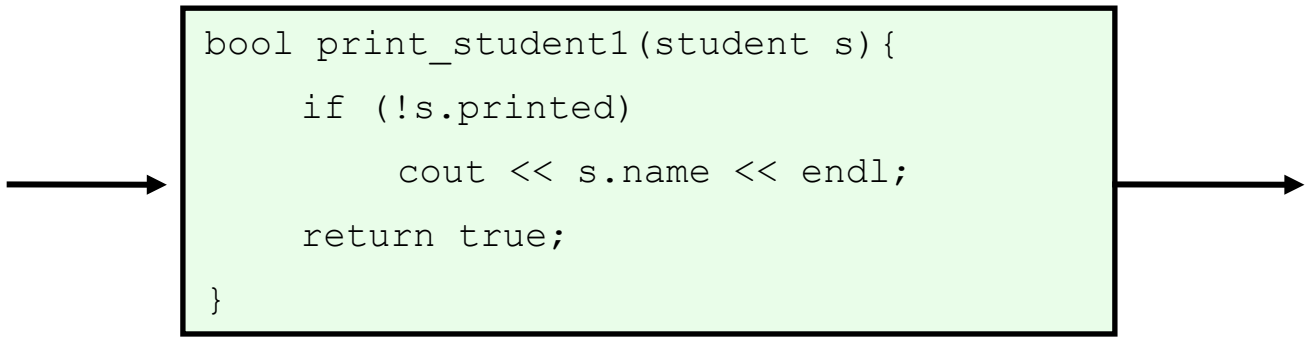
Parameter passing:

```
struct student {
     string name;
     PNG mug;
     bool printed; // print flag
};
```

What happens when we run code like this:

```
int main() {
     student a;
     print_student1(a);
}
```

?



Parameter passing:

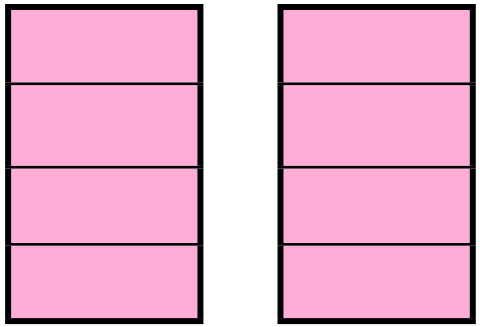
```
struct student {
     string name;
     PNG mug;
     bool printed; // print flag
};
```

Function defn

```
bool print_student1(student s){
     if (!s.printed)
          cout << s.name << endl;
     return true;
}
```

Example of use

```
student a;
… // initialize a
a.printed = print_student1(a);
cout << a.printed << endl;
```



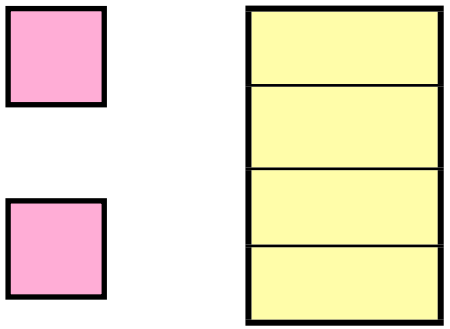
Parameter passing:

```
struct student {
     string name;
     PNG mug;
     bool printed; // print flag
};
```

Function defn

```
void print_student2(student     s){
     if (!  s.printed)
          cout <<    s.name << endl;

}
```



Example of use

```
student * b;
… // initialize b
print_student2(b);
cout << b.printed << endl;
```

Parameter passing:

```
struct student {
     string name;
     PNG mug;
     bool printed; // print flag
};
```

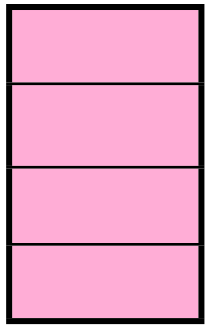
Function defn

```
void print_student3(student     s){
     if (!  s.printed)
          cout <<    s.name << endl;

}
```

Example of use

```
student c;
… // initialize c
print_student3(c);
cout << c.printed << endl;
```



Parameter passing summary:

```
struct stu {
     string n;
     PNG mug;
     bool pt; // print flag
};
```

Function defn

```
bool ps1(stu s){
     if (!s.pt)
          cout << s.n;
     return true;
}
```

```
void ps2(stu * s){
     if (!s->pt)
          cout << s->n;
     s->pt = true;
}
```

```
void ps3(stu & s){
     if (!s.pt)
          cout << s.n;
     s.pt = true;
}
```

Example of use

```
stu a;
… // init a
a.pt = ps1(a);
cout << a.pt;
```

```
stu * b;
… // init *b
ps2(b);
cout << b->pt;
```

```
stu c;
… // init c
ps3(c);
cout << c.pt;
```

Today's new learning: arrays and...

1. pass/return by value
2. pass/return pointer by value
3. pass/return by reference

Assignments for independent learning:

1. flower ** plot;
2. reference variables:

   http://www.cprogramming.com/tutorial/references.html