Announcements


```c
int *p, *q;
p = new int;
q = p;
*q = 8;
q = new int;
*q = 9;
p = NULL;

int *p;
int x = 5;
p = &x;
delete x;
p = NULL;
```

```c
int *p, *q;
p = new int;
q = p;
*q = 8;
delte q;
*p = 12;
p = NULL;
```

```c
int *p;
int x = 5;
*p = x;
```
Stack vs. Heap memory:

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.

Allocated memory must be deleted programmatically.

Data must be accessed by a pointer.

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

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

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

  face a, b;
  ...
  // init b
  a = b;
  a.setName("ann");
  b.getName();

  face * c, * d;
  ...
  // init *d
  c = d;
  c->setName("carlos");
  (*d).getName();
```
Arrays: static (stackic)

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

<table>
<thead>
<tr>
<th>loc</th>
<th>name</th>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

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

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

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

What happens when we run code like this?
Parameter passing:

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

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

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