C++ Short Course Part 1

- Classes
- Pointers
- Arrays
- Parameter passing
- Return values

C++ Short Course Part 1

- Classes
- Pointers
- Arrays
- Parameter passing
- Return values

Classes in C++:

Every variable has name, type, value, location in memory

Primitive types:

```
bool tired;

int myFavoInt;

char rating = 'E';

double u = 37.;
```

User defined types:

```
sphere myFavoSphere;
```

class is a group of member variables (fields) and functions (methods)

members

Structure of a class defn:

how do we implement `sphere myFavoSphere;` ?

sphere.h
class definition

```
class sphere{
//member declarations
..public:
private:
};
```

Sphere.cpp

`sphere` member function definitions.

Structure of a class defn (cont):

```
class sphere{

public:
   sphere();
   double getDiameter();
   void setRadius(double nr);



private:
   double theRadius;
};
```

sphere representation:

double theRadius;

sphere functionality:

1. create a new one
2. get info from an existing
3. change an existing

```
int main(){
    sphere ms;
    ms.setRadius(4.0);
    cout << ms.getDiameter()
         << endl;
}
```

8

Structure of a class defn (cont):

```
class sphere{
public:
sphere();
sphere(double r);
void setRadius(double newRad);
double getDiameter() const;
…



private:
double theRadius;
};
```

```
//constructor(s)  (next page)

void sphere::setRadius(double newRad){
    theRadius= newRad;
}

double sphere::getDiameter() const {
    return 2*theRadius;
}
…
```

Asides:

const : promise that getDiam will not change variables assoc w object.

: : Scope resolution operator

sphere g(8.0); invokes new sphere ctor

Constructors (intro): When you *declare* a sphere, a sphere class constructor is invoked.

Points to remember abt ctors:

1. if you don't write a ctor, system provides 1 — a no-arg ctor;
2. if you write any, system provides none!
3. ctor never called by your code. We provide it for system to use.

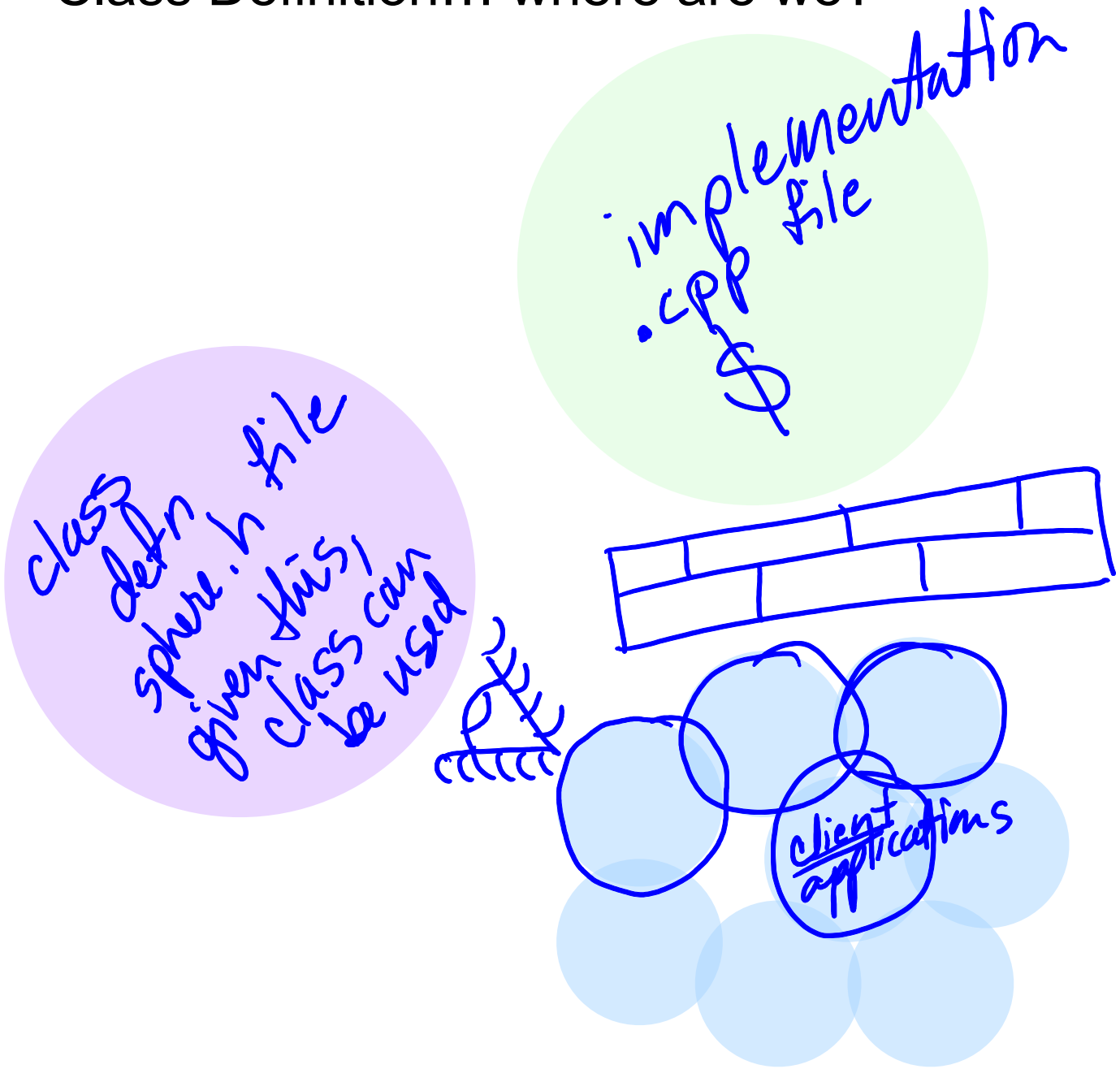
```
int main(){
  sphere s;



}
```

```
…
//default constructor
sphere::sphere(){
    theRadius = 0.0;
}

//default constructor, alternative
sphere::sphere(): theRadius(0.0), color(red)
{                  initialization line.
}

//constructor with given radius
sphere::sphere(double r){
  error handling
  theRadius = r;
}

…
```

theRadius(r)

sphere p(-52.8);

Class Definition… where are we?



Stepping back…

Ideas/concepts:

- Class definitions
- Class function implementation
- Constructors
- Clients

OOP: we now understand how C++ supports

Inheritance

Encapsulation (separation of interface from implementation)

1) use of private to control access to the data associated w an object
2) decoupling of implementation from interface by scope + by file naming conventions.

Polymorphism

Our first class…

sphere.h

```
class sphere{

};
```

What surprises you about this code?

main.cpp

```
#include "sphere.h"

int main(){
   sphere a;
}
```

1. Upon command > `clang++ main.cpp` does this code compile?
2. Upon command > `./a.out` does it run?

Access control and encapsulation:

sphere.h

```
class sphere{
  private:
    double theRadius;

};
```

main.cpp

```
#include "sphere.h"
#include <iostream>
using namespace std;


int main(){
   sphere a;
   cout << a.theRadius << endl;
}
```

What surprises you about this code?

1. Upon command `> clang++ main.cpp` does this code compile?
2. ~~Upon command `> ./a.out`~~ does it run?
3. In c++ class members are, by default, "private". Why would we want to hide our representation of an object from a client?
4. How many collaborators are you allowed to have for PAs in this cpsc221?

C++ Short Course Part 1

- Classes
- Pointers
- Arrays
- Parameter passing
- Return values

Switching gears…

Configure your iMac 27-inch

Use the options below to build the system of your dreams

Memory

More memory (RAM) increases performance and enables your computer to perform faster and better. Choose additional 1066MHz DDR3 memory for your iMac.

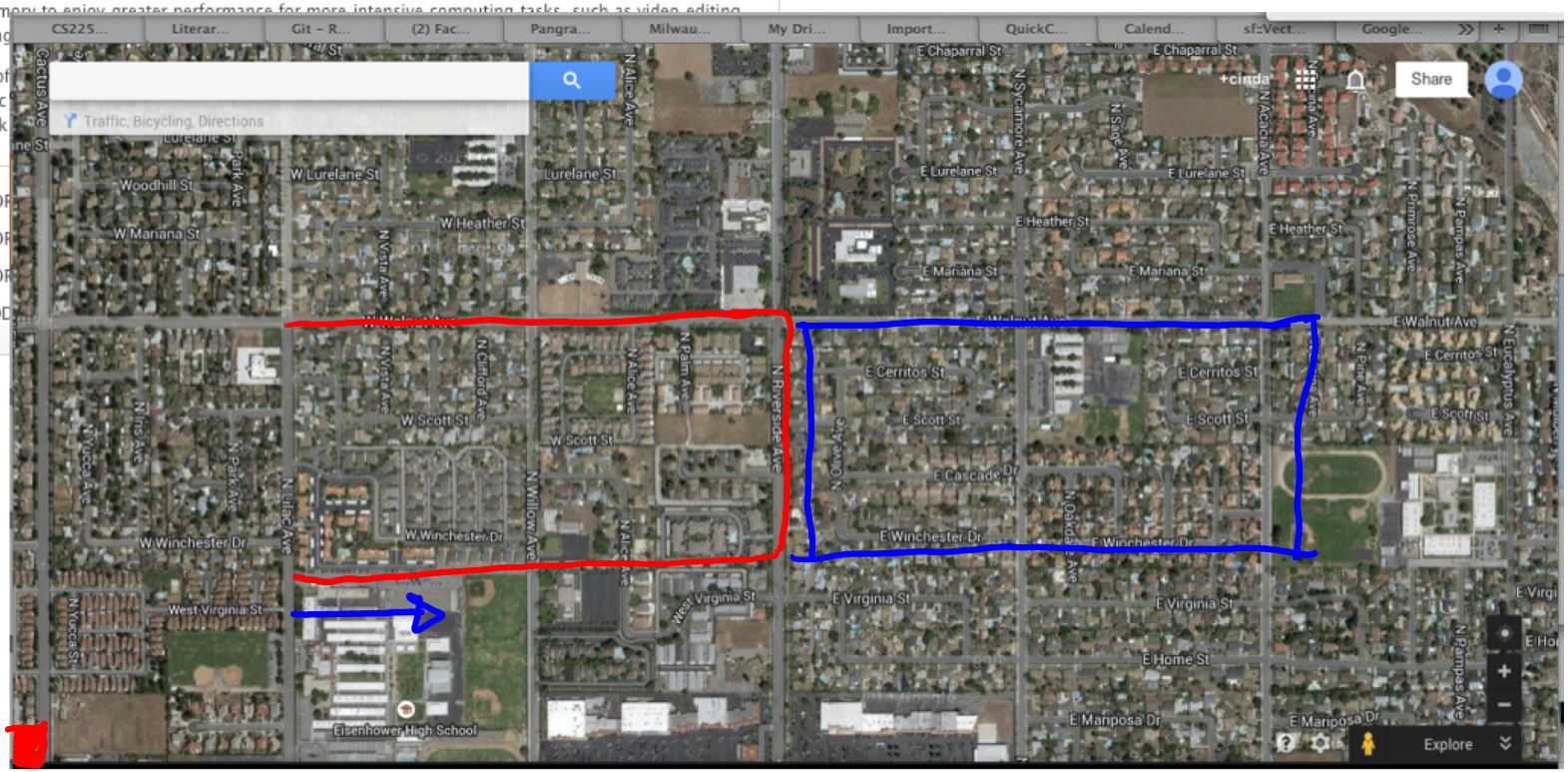
Learn more

The more memory your computer has, the more programs you can run simultaneously, and the better performance you get from your computer.

- Select the standard memory configuration to support day-to-day tasks such as email, word processing, and web browsing as well as more complex tasks such as editing photos, creating illustrations, and building presentations.
- Upgrade your memory to enjoy greater performance for more intensive computing tasks, such as video editing and DVD authoring

Your iMac uses one of synchronous dynamic without wasting clock

- 4GB 1066MHz DDR
- 8GB 1066MHz DDR
- 8GB 1066MHz DDR
- 16GB 1066MHz DD



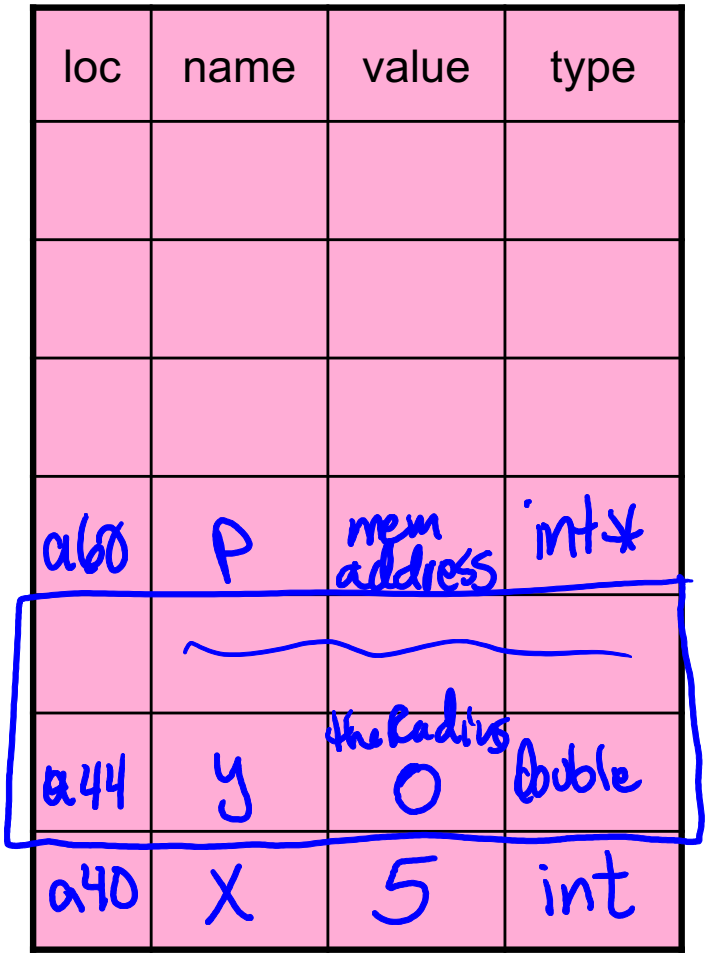
0x0

Variables and memory in C++

```
int x;
x = 5;

sphere y;

int * p;
```

System administered

Stack memory



Pointers - Intro

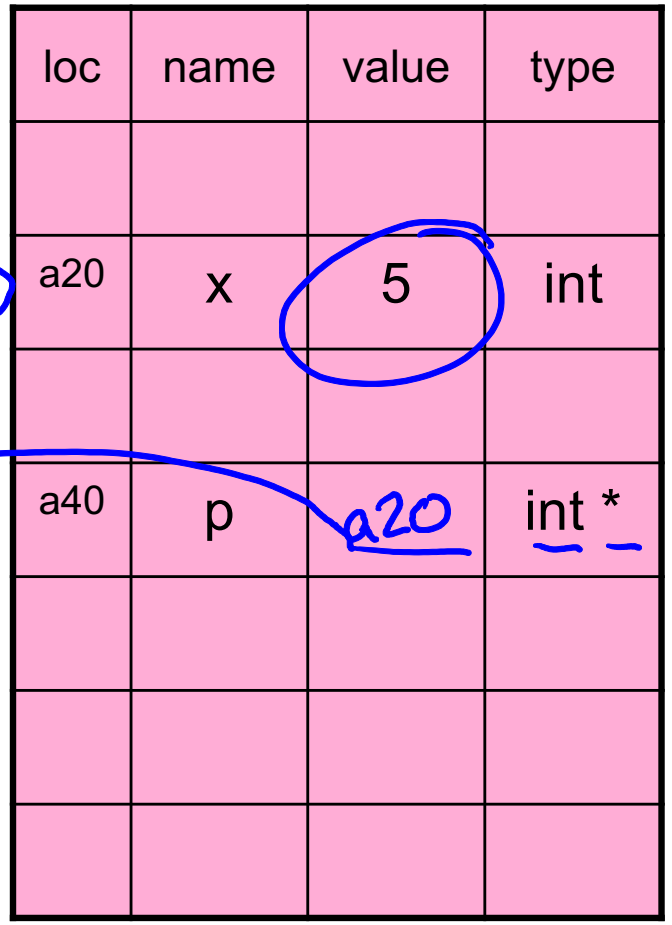
```
int x;
int * p;
```

How do we assign to p?

```
p = x;
```

```
p = a20;
```

Stack memory



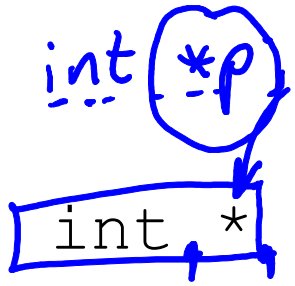
mem address operator: &

returns mem address of x in &x.

dereference operator: *

p=&x; cout << *p << endl; → 5

Pointer variables and dynamic memory allocation:



p = new int;

*p = 7;

Stack memory

loc	name	type	value
a40	p	int *	b12
	q		b12

You administer via code

Heap memory

loc	name	type	value
b12		int	7

Youtube: pointer binky c++

Fun and games with pointers: (warm-up)

```
int * p,*q;
```

What type is q? int

```
int *p;
int x;
p = &x;
*p = 6;
cout << x;
cout << p;
```

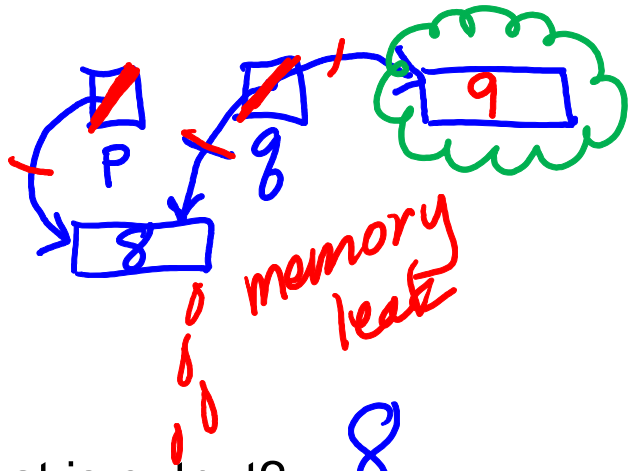
What is output? 6

What is output? _____

Write a statement whose output is the value of `x`, using variable `p`: _____

cout << *p;

```
int *p, *q;
p = new int;
q = p;
*q = 8;
cout << *p;
q = new int;
*q = 9;
p = NULL;
delete q;
q = NULL;
```



What is output? 8

Do you like this? _____

cout << *q; → 9

Do you like this? _____

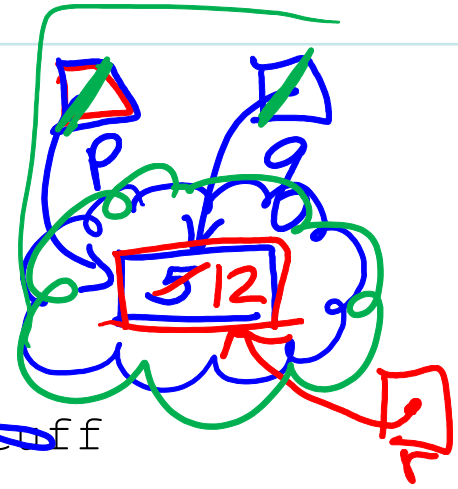
Memory leak: occurs when all references to a piece of memory are removed.

Deleting a null pointer: no op — nothing happens

Dereferencing a null pointer: runtime error - seg fault bus error

Fun and games with pointers:

```
int * p, * q;

p = new int(5);

q = p;

delete p;

... // some random stuff

cout << *q;
```



*q = *p

p
q

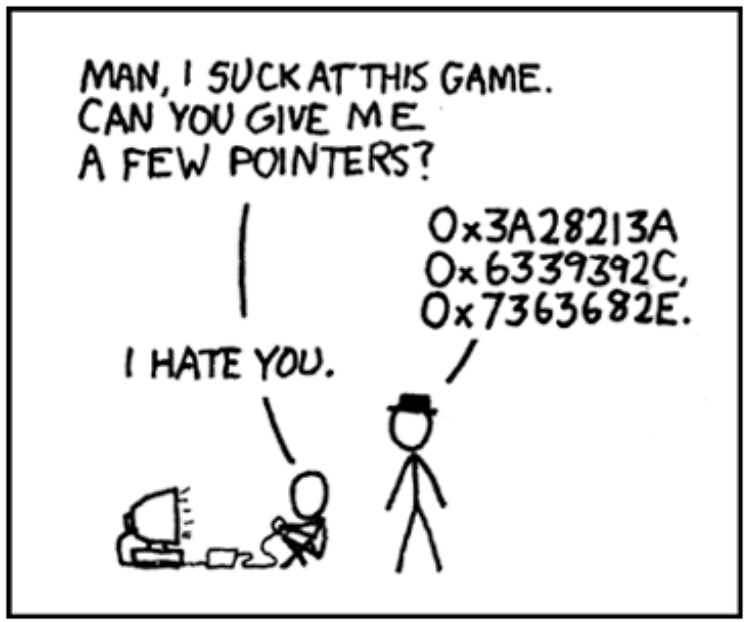
Do you like this?_____

↳ 5

int * r = new int(12);

*q = 5;

cout << *r;



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;
}
```

necessary!

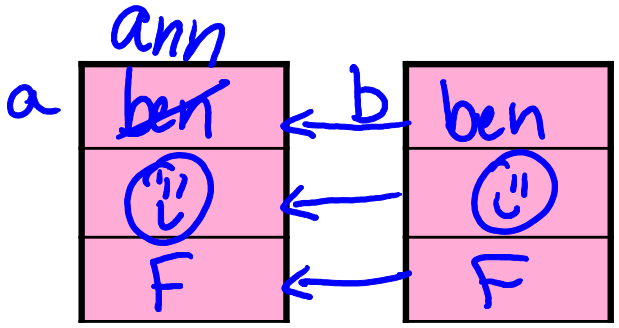
Allocated memory must be deleted programmatically.

Data must be accessed by a pointer.

Pointers and objects:

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

↳ ben

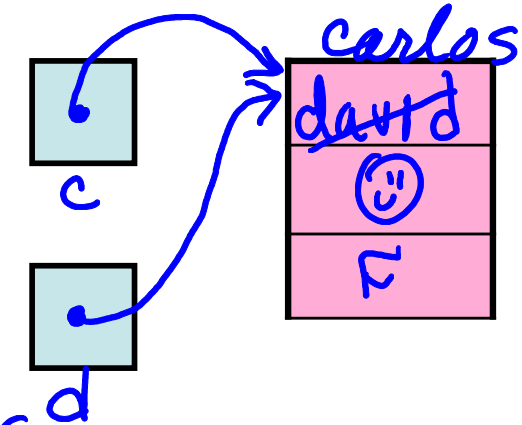



```
class face {
public:
     void setName(string n);
     string getName();
     …
private:
     string name;
     PNG pic;
     boolean done;
};
```

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

carlos

→ (*c).setName("carlos");



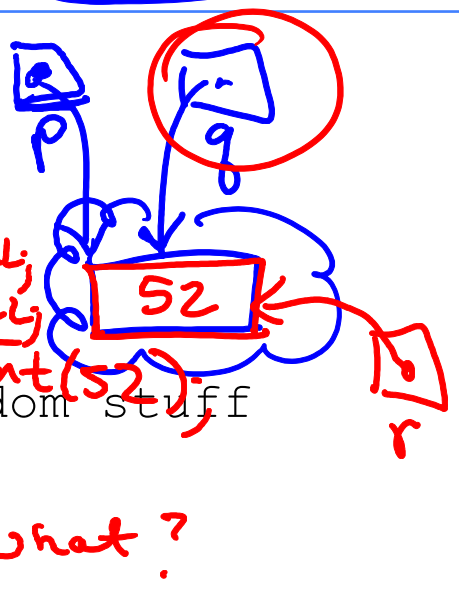
Practice--

```
int * p; int x;

p = x;
```

Do you like this? NO

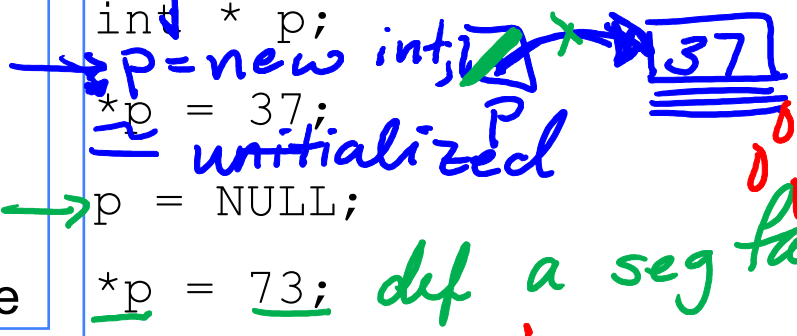
What kind of error? Compiler Runtime

```
int * p, * q;

p = new int;

q = p;

delete p;

… // some random stuff

cout << *q;
```



Do you like this? well we do now.

```
int * p;
p = new int;
*p = 37;
p = NULL;
*p = 73;
```



Do you like this? No

What kind of error? Compiler Runtime

```
int * p;  int x;
```

Variable `p` can be given a target (pointee) in two ways. Write an example of each.

H p = new int;

S p = &x;

int ** q;

q = &p;

Use the letters S and H in a meaningful way to tell where the pointee exists in memory.

C++ Short Course Part 1

- Classes
- Pointers
- Arrays
- Parameter passing
- Return values

Arrays: static (stackic)

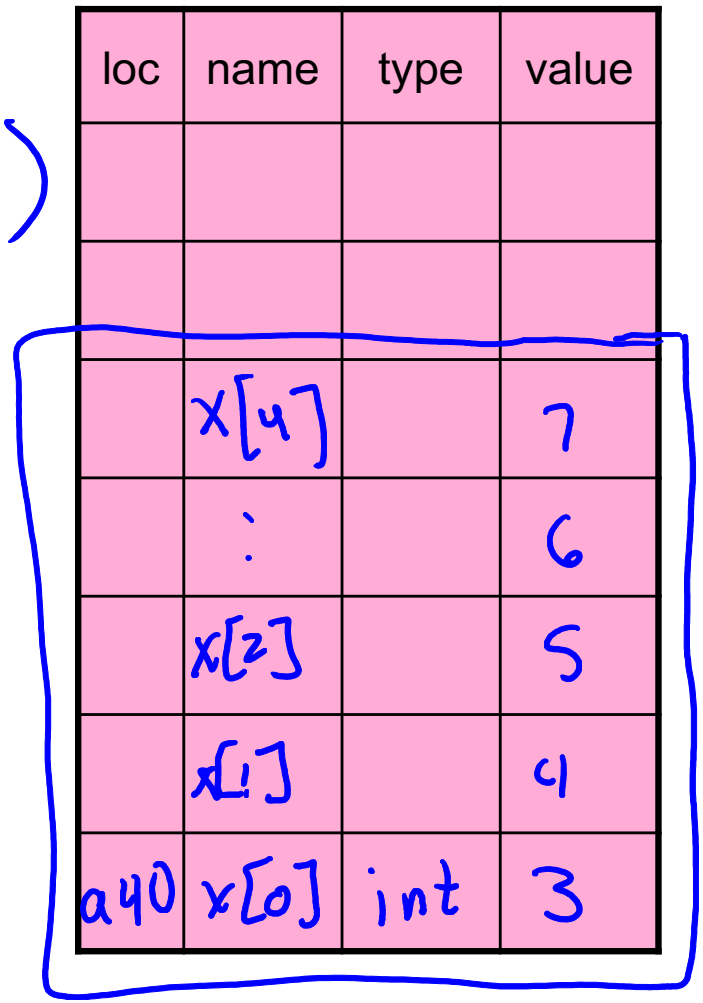
```
int x[5];
```

```
for(int i=0; i<5; i++)
    x[i] = i+3;
```

#

&x

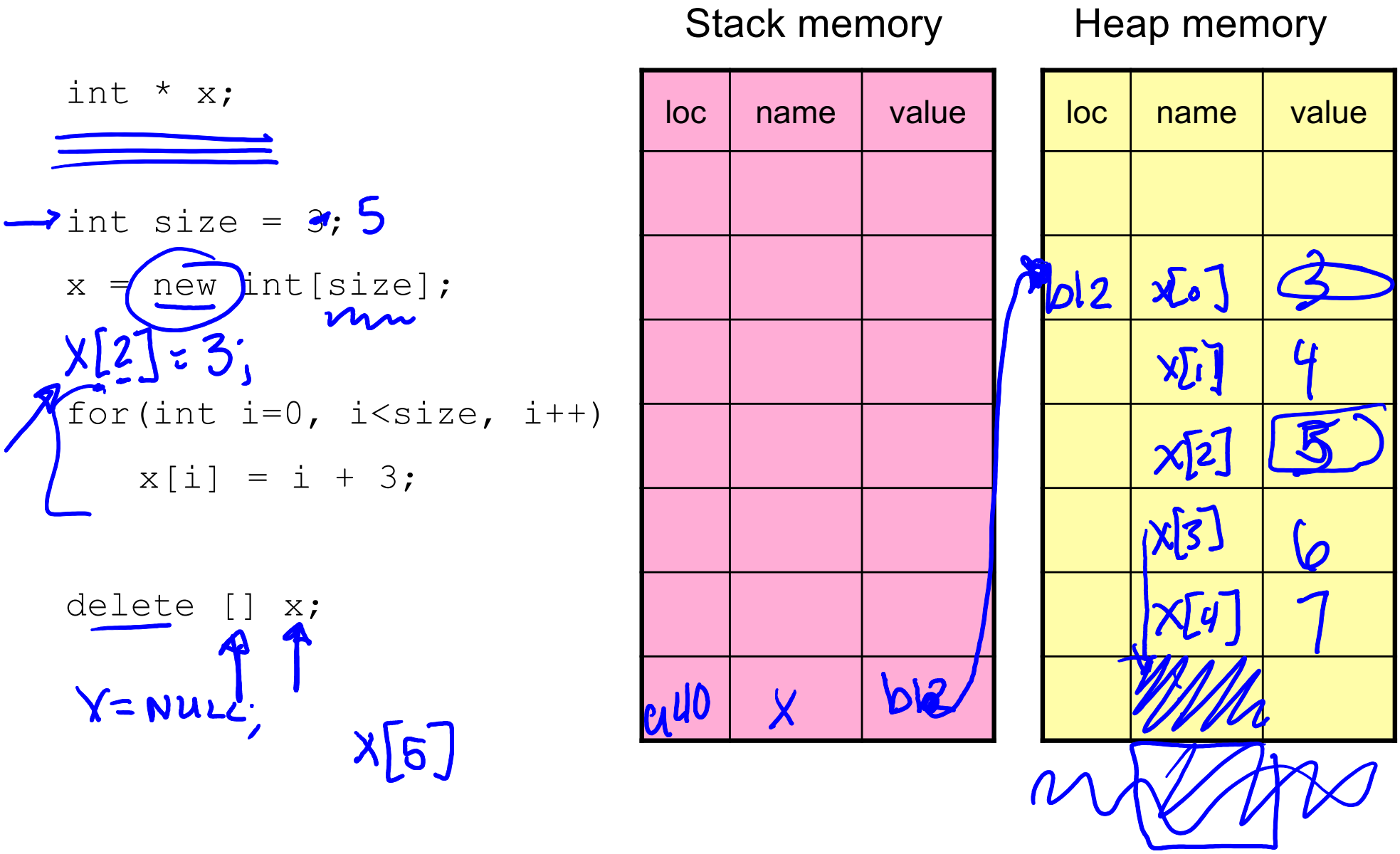
Stack memory



loc	name	type	value
	x[4]		7
	⋮		6
	x[2]		5
	x[1]		4
a40	x[0]	int	3

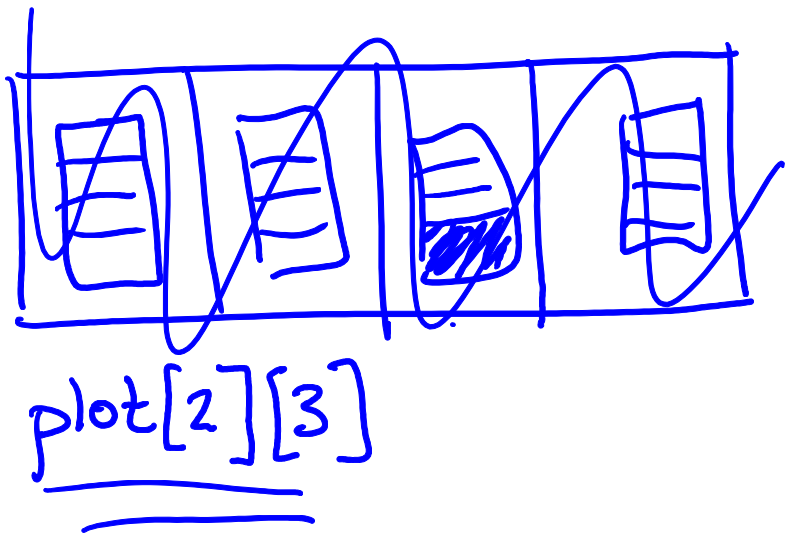
Arrays: dynamic (heap)

```
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:

plot can be a 2D array of flowers.

Option 2:

Option 3:

array of flower ptrs

Option 4:

ptr to an array of flowers

C++ Short Course Part 1

- Classes
- Pointers
- Arrays
- Parameter passing
- Return values

Parameter passing:

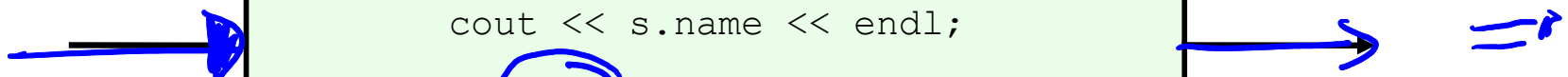
```
struct student {
    string name;
    PNG mug;
    bool printed; // print flag
};
```

What happens when we run code like this:

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

?

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



Parameter passing:

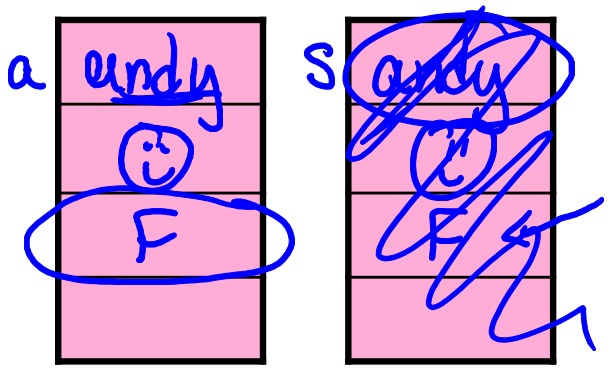
C++ passes by value by default.

```
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;
}
```

→ andy



Example of use

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

- can't change the value of calling param
- copies may be slow!!

Parameter passing:

passing a pointer by value.

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

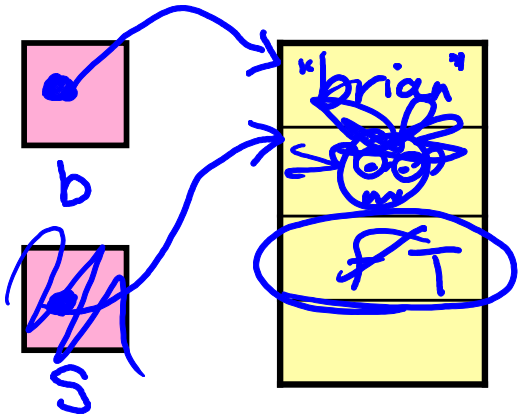
Function defn

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

if (s != NULL) {

// s->name

s->printed = TRUE;



Example of use

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

+ fast
+ can change value of fn argument.
- requires ptr type
  - must do null check
  - dereferences make code unreadable & induce lots of bugs.

Parameter passing:

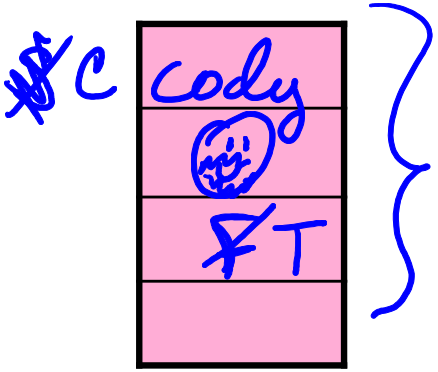
pass by reference

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

Function defn

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

s.printed = true; → cody



Example of use

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

True

C++ Short Course Part 1

- Classes
- Pointers
- Arrays
- Parameter passing
- Return values

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

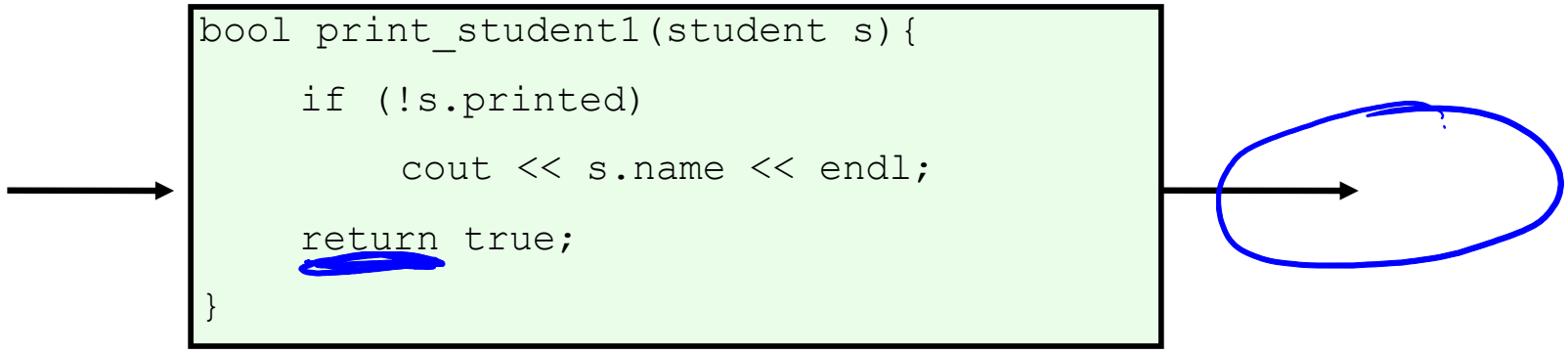
Return values:

What happens when we run code like this:

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

?

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



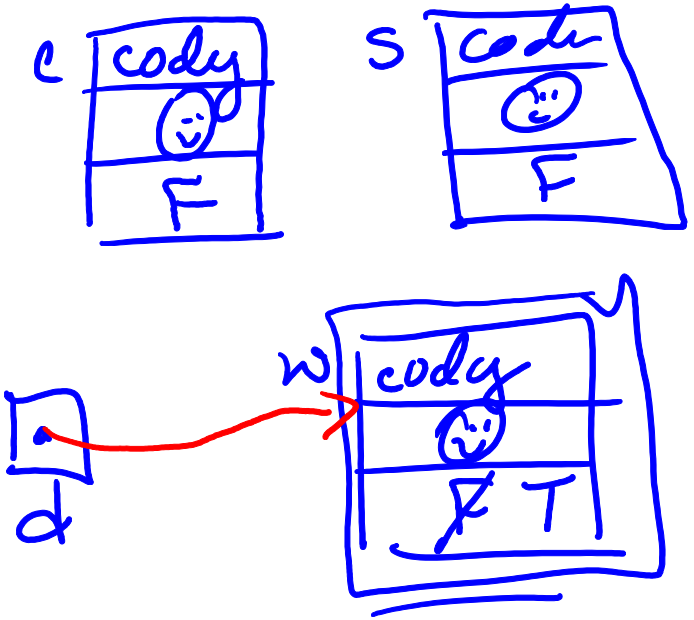
Return by value or *by value or by reference .

Returns:

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

Function defn

```
student * print_student5(student s){
     student w = s;
     if (!w.printed){
          cout << w.name << endl;
          w.printed = true;
     }
     return &w;
}
```



Example of use

```
student c;
student * d;
… // initialize c
d = print_student5(c);
```

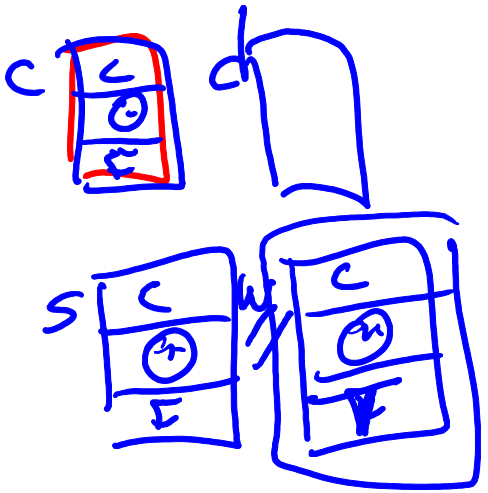
DONOT RETURN MEM ADDR OF A LOCAL VAR OR PARAM.

Returns:

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

Function defn

```
student & print_student5(student s){
    student w = s;
    if (!w.printed){
        cout << w.name << endl;
        w.printed = true;
    }
    return w;
}
```



Example of use

```
student c,d;
… // initialize c
d = print_student5(c);
```

Lesson: don't return 1) a pointer to a local variable, nor 2) a local variable by reference.