Chapter 11
Introduction to Pointers

Pointer retriever

Agenda

- Pointers to objects
- Dynamic Arrays

Review: You can “**store**” **the address** of a variable in a **pointer variable**

```
int num = 42;
int * ptr = &num;
```

“address of” operator

Memory can be allocated using the **new** operator

```
int * ptr1 = new int;
```

Returns address of allocated memory

Newly allocated memory must be returned

```
delete ptr1;
```

Delete operator gives memory back to system

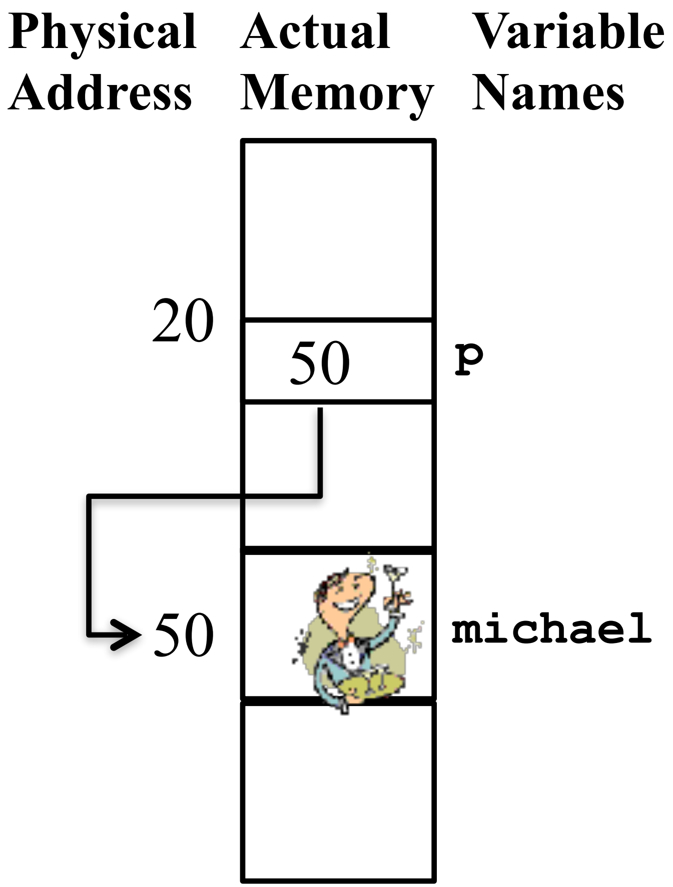
Review: You can **“access” the content** of an address stored in a **pointer variable** with the * **operator**

```
*ptr = 42;
cout << *ptr << endl;

*ptr = 13;
```

“dereference” operator

You can make pointers that “**point to**” any variable (or object) you wish…

```
Person michael;
Person * p = NULL;
p = & michael;
```

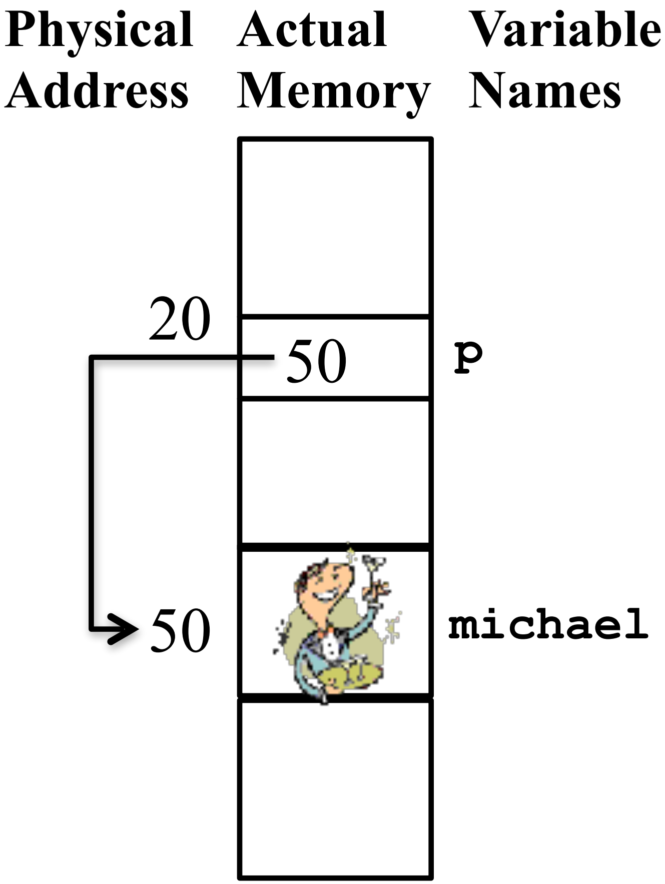


You can use the **dereference operator *** to **access that object's content:**

```
cout <<  (*p).getName();
```

The * operator accesses the **memory location**

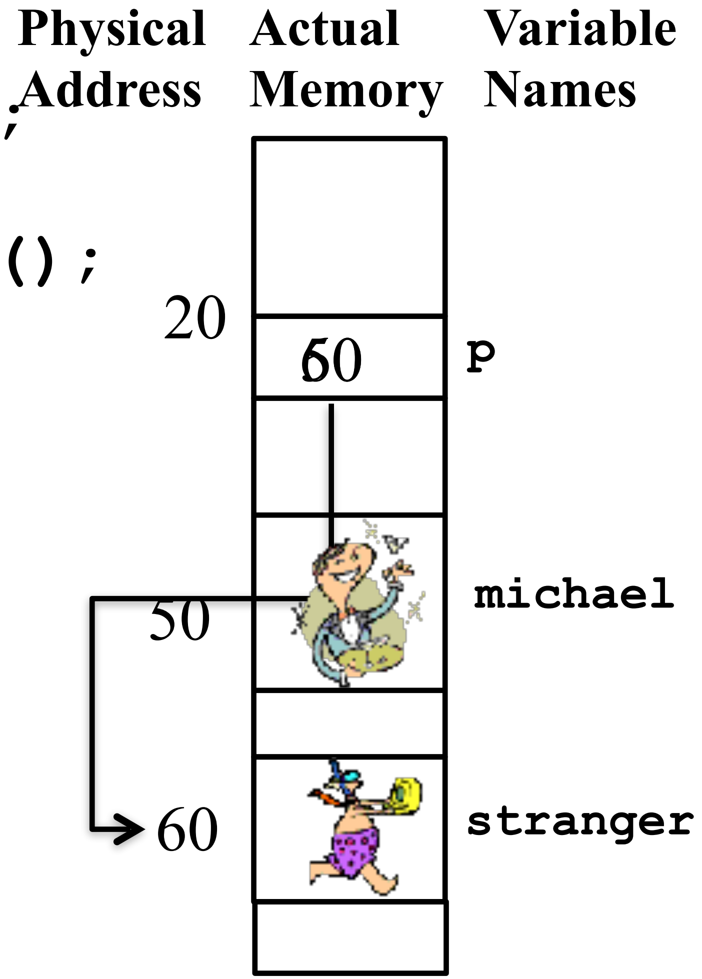
```
michael
```



You can also use the **arrow operator ->** to access any public variables/methods in an object pointee:

```
p = &stranger;
//(*p).setName("stinky");
p->setName("stinky");
cout << stranger.getName();
cout << p->getName();
```

```
stinky
stinky
```



Recap: There are **2 ways** to access properties/methods in a pointer to an object.

- **Dot operator** (.) on the object itself:

```
(*ptr).setName("Jimmy Jones");
```

- **Arrow operator** (->) on the pointer to the object:

```
ptr->setName("John Carter");
```

In-class Exercise

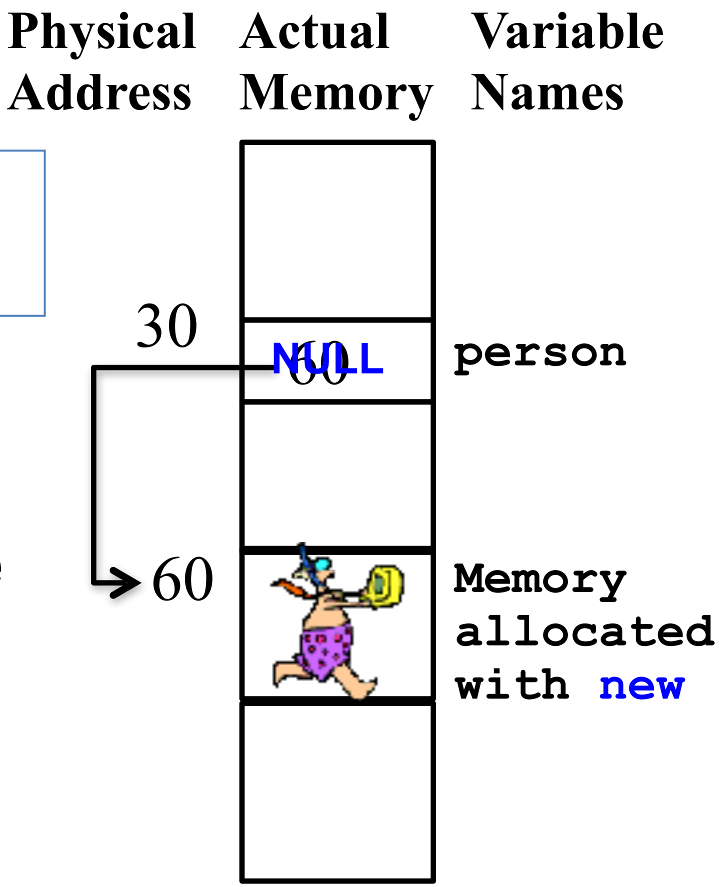
```
class Person {
private:
        string name;
        int age;

public:
        Person() { name = "George Whitworth"; age = 198; }
        Person(string n, int a) { name = n; age = a; }

        string getName() { return name; }
        int getAge() { return age; }
        void birthday() { age++; }
};

int main()
{
        Person p;

        cout << "Name: " << p.getName() << ", age: " << p.getAge() << endl;
        cout << "Happy Birthday " << p.getName() << "!";

        p.birthday();
        cout << " You're now " << p.getAge() << " years old!\n\n";
}
```

Change this to use a pointer

Dynamic Allocation of Objects

```
Person * person = NULL;
person = new Person;
```

We can create an object dynamically by using the new operator



Physical Address

Actual Memory

Variable Names

30

NULL

60

person

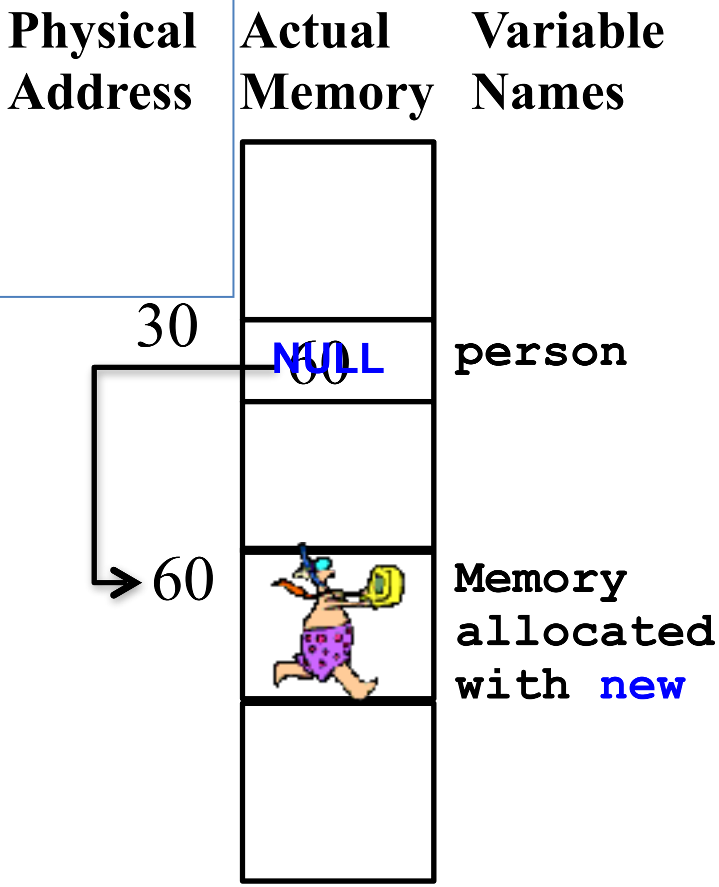
60

Memory allocated with new

Dynamic Allocation of Objects using Constructors

```
Person * p = NULL;
p = new Person("Joe Smith", 32);
cout << p->getName();

…

delete p;
```

We can create an object dynamically by using the new operator



Physical Address

Actual Memory

Variable Names

30

NULL

60

person

60

Memory allocated with new

In-class Exercise

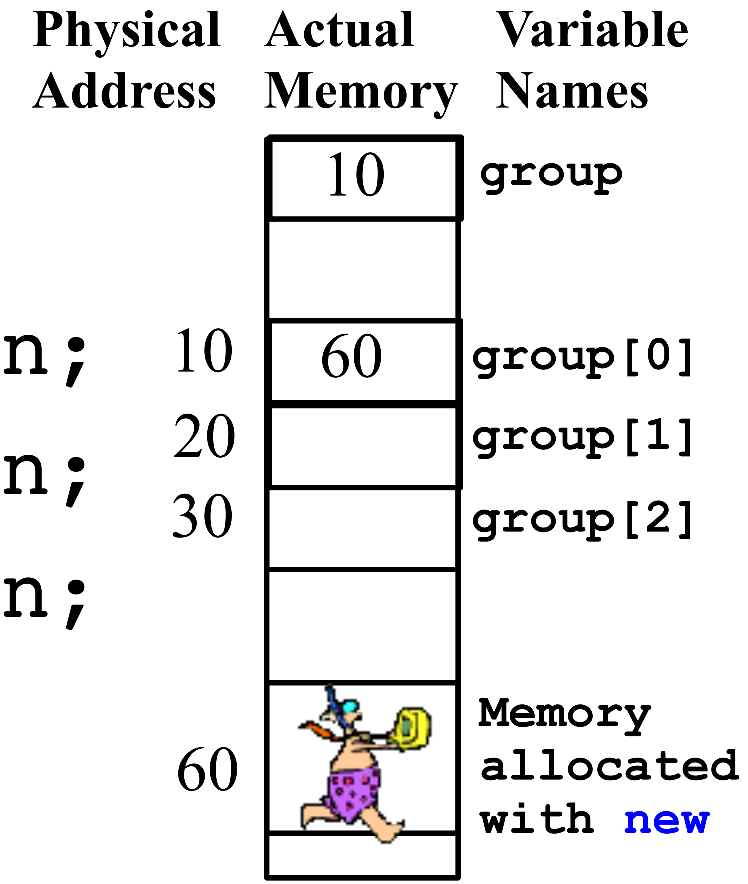
```
class Person {
private:
        string name;
        int age;

public:
        Person() { name = "George Whitworth"; age = 198; }
        Person(string n, int a) { name = n; age = a; }

        string getName() { return name; }
        int getAge() { return age; }
        void birthday() { age++; }
};

int main()
{
        Person p;

        cout << "Name: " << p.getName() << ", age: " << p.getAge() << endl;
        cout << "Happy Birthday " << p.getName() << "!";

        p.birthday();
        cout << " You're now " << p.getAge() << " years old!\n\n";
}
```

Change this to use a Dynamically-allocated pointer

An array of pointers?!

- A **pointer type** is just like any other data type

```
Person*  group[3];
group[0] = new Person;
group[1] = new Person;
group[2] = new Person;
```



Physical Address

Actual Memory

Variable Names

10 group

10 60 group[0]

20 group[1]

30 group[2]

60 Memory allocated with new

Memory **Deallocation**

- When you are done with the memory you need to **deallocate** it with the **delete** operator

```
double * ptr2 = new double(4.2);
…
delete ptr2;
ptr2 = NULL;
Person * person = new Person;
…
delete person;
person = NULL;
```

More examples: Dynamic Memory Allocation/ Deallocation

```
int * ptr2[4]; // Array of pointers
…
ptr2[3] = new int(4);
…
delete ptr2[3];
```

Imagine you need to create an array of some size while your program is running …

- You can allocate and deallocate arrays dynamically …

```
int* ptr3 = new int[4];

…

delete [] ptr3;
```

Note the []