CMPE 180-92

Data Structures and Algorithms in C++

September 14 Class Meeting

Department of Computer Engineering
San Jose State University

Spring 2017
Instructor: Ron Mak
www.cs.sjsu.edu/~mak

SILICON VALLEY'S
FIRST CHOICE
FOR NEW
ENGINEERING HIRES
— Silicon Valley Business Journal, 2013

San José State
UNIVERSITY

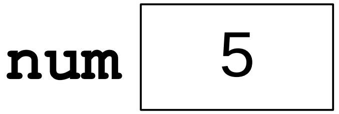
Assignment #3 Sample Solutions

Pointers

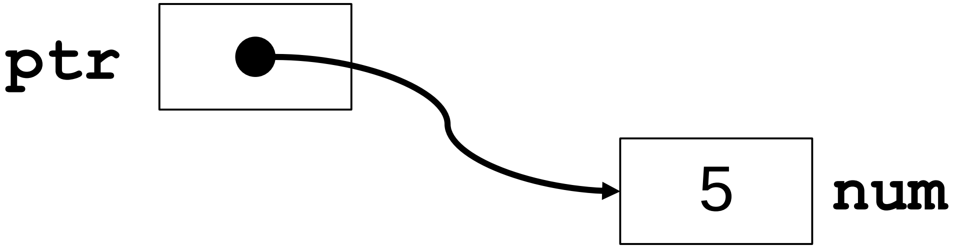
- Pointers are an extremely powerful feature of C and C++ programs.
  - You would not be a competent C or C++ programmer if you did not know how to use pointers effectively.
- Pointers can also be extremely dangerous.
  - Many runtime errors and program crashes are due to misbehaving pointers.
  - Pointers are a prime cause of memory errors.

An **int** vs. Pointer to an **int**

- A graphical representation of an **int** variable named **num** and its value:



- A graphical representation of a pointer variable named **ptr** that points to an **int** value of a variable named **num**:

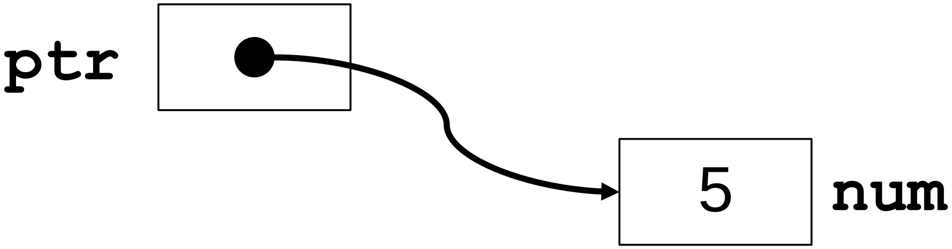


Declaring and Assigning Pointers

- After the following statements are executed:

```
int  num = 5;
int *ptr = &num;
```

- We have this situation:



Pointers are Addresses

- To declare that a variable is a pointer, use a **`*`** before the variable name:

```
int *ptr;
double *ptr2;
```

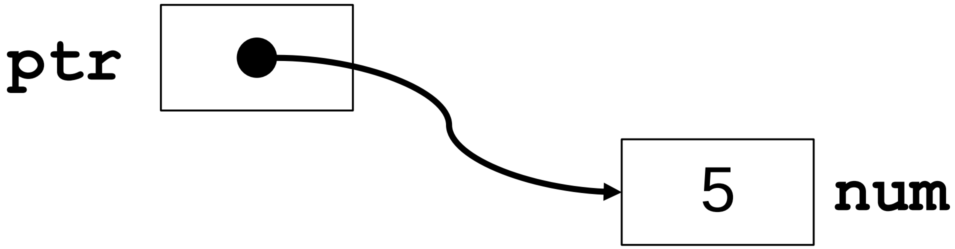
  - **`ptr`** can point to an **`int`** value
  - **`ptr2`** can point to a **`double`** value

`&` is the address-of operator

- The statement `ptr = &num;` assigns the address of variable **`num`** to pointer variable **`ptr`**
  - Make **`ptr`** point to the address of variable **`num`**.

The Dereferencing Operator

```
int  num = 5;
int *ptr = &num;
```



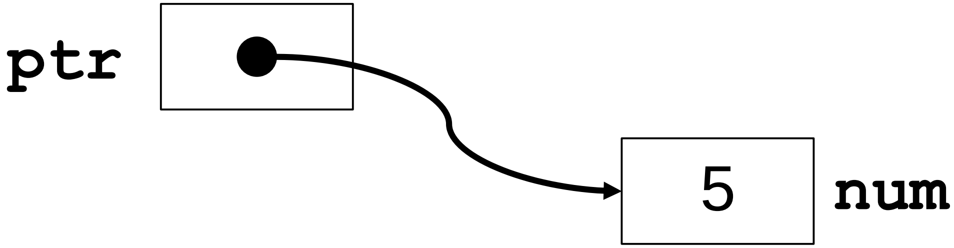
- To get the value that pointer ptr is pointing to:

```
*ptr
```

- Now the * is the dereferencing operator.
  - "Follow the pointer to get what it's pointing to."
- We can use *ptr in an expression.
  - Example: *ptr + 2 gives the value 7.

The Dereferencing Operator, *cont'd*

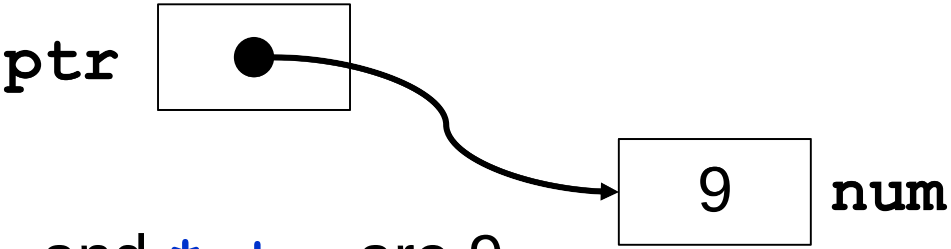
```
int  num = 5;
int *ptr = &num;
```



- In the above example, both **`*ptr`** and **`num`** refer to the same value 5.

- What happens if we execute the statement?

```
*ptr = 9;
```



  - Now both **`num`** and **`*ptr`** are 9.

A Pointer Declaration Warning

- You can declare several pointer variables in one line:

```
double *ptr1, *ptr2, *ptr3;
```

- How many pointer variables do we have?

```
double* ptr1, ptr2, ptr3;
```

  - Only `ptr1` is a pointer to a double value. `ptr2` and `ptr3` are simple double variables.

Break

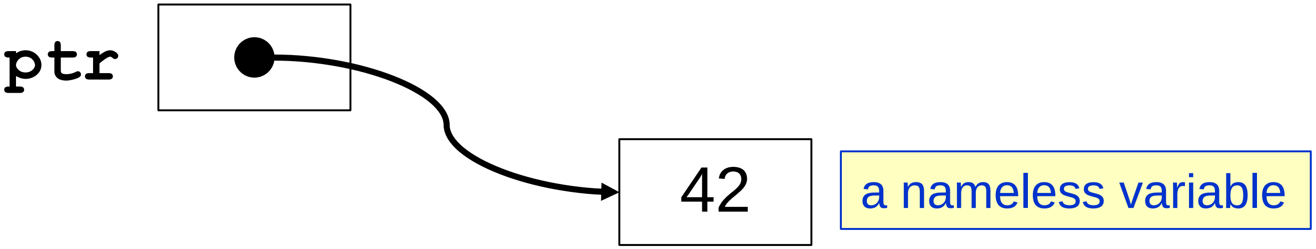
The **new** Operator

- So far, all our variables have names and are created automatically when we declare them:

```
int num;
```

- We can also create nameless variables.
  - The **new** operator returns a pointer to the variable it just created.
  - This is ideal for pointer variables.

```
int *ptr = new int(42);
```



ptr

42

a nameless variable

The **delete** Operator

- If your program creates nameless variables, then it must remove them from memory when the program no longer needs them.
  - Delete from memory the nameless variable that **ptr** points to.

```
delete ptr;
```

- If your program doesn’t get rid of all the nameless variables it created, those variables clutter up memory, and therefore you are said to have a memory leak.

Pointer Parameters

- We can pass a pointer by value to a function:

```
void foo(int *ptr1, double *ptr2);
```

  - We can change the value of the variable that `ptr1` points to.

- We can also pass a pointer by reference:

```
void bar(int* &ptr1, double* &ptr2);
```

  - We can change what variable `ptr1` points to.
  - Ugly syntax!

typedef

- Use **typedef**s to simplify pointer notation:

```
typedef int    *IntPtr;
typedef double *DoublePtr;
```

- Now you can use **IntPtr** in place of **int *** and **DoublePtr** in place of **double ***

```
void foo(IntPtr ptr1, DoublePtr ptr2);
```

```
void bar(IntPtr& ptr1, DoublePtr& ptr2);
```

Using Pointers to Pass-by-Reference

- C programmers used pointers to pass parameters by reference.
  - Example:
    ```
    function baz(int *parm);
    ```
- A call to the function needed the address of the corresponding argument:

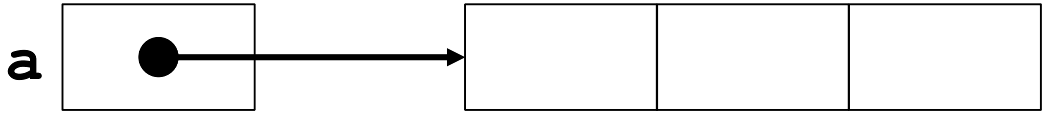
  ```
  int arg;
  baz(&arg);
  ```

  - Because `parm` points back to the actual argument, the function can use `*parm` to change the value of the actual argument.

Pointers and Arrays

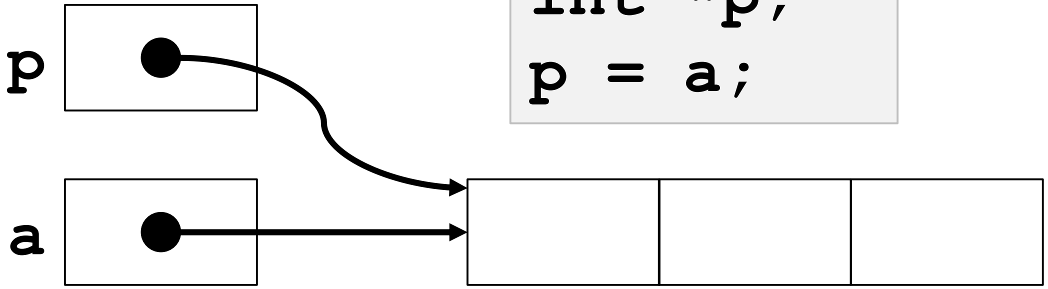
- An array variable is actually a pointer variable.

```
int a[3];
```



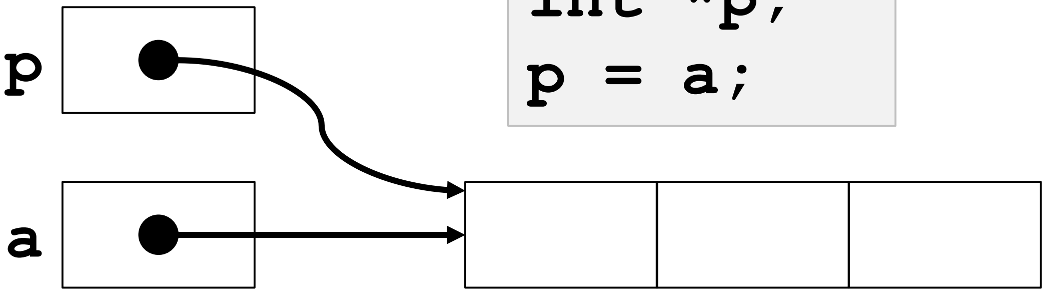
- The array/pointer variable points to the first element of the array.

```
int a[3];
int *p;
p = a;
```



Pointer Arithmetic

```
int a[3];
int *p;
p = a;
```

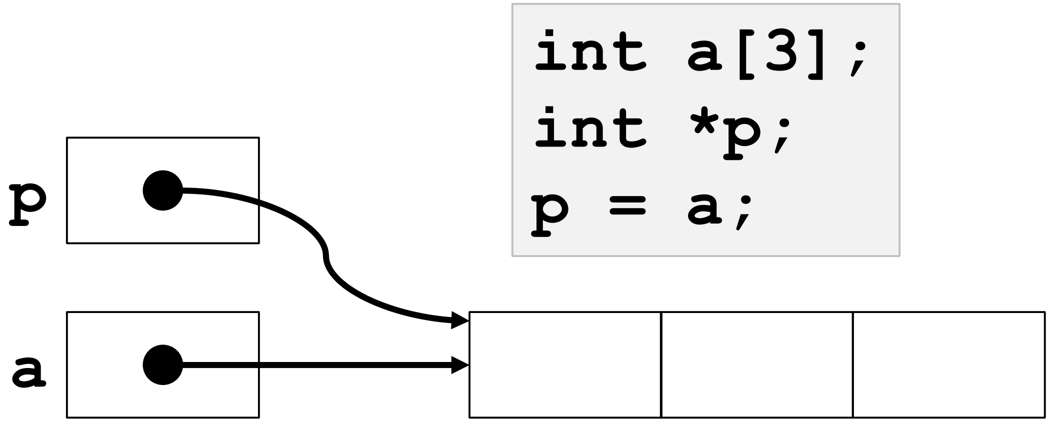


- The following expressions all access the third array element:

```
a[2]
p[2]
*(p+2)
*(a+2)
```

What is `*p+2` ?

Pointer Arithmetic, *cont'd*



```
int a[3];
int *p;
p = a;
```

- Use a pointer to iterate through an array.
  - In the above example, `p` initially points to the first element of the array.
  - Then `p++` points to the second element.
  - And next, `p++` points to the third element.

Dynamic Arrays

- Up until now, whenever we declared an array, we explicitly gave its size.
  - Example:

```
int a[10];
```

- But suppose we don't know until run time how many elements we need.
  - Example: At run time, your program reads in a count of names, and then the names. You want to create an array that can hold exactly that many names.
- You can use a dynamic array (instead of a vector).

Dynamic Arrays, *cont'd*

- If the size of the array you want is in variable `n` whose value you don't know until run time, use the `new` operator to create an array of size `n`.

- Use a pointer variable to point to the first element of the dynamic array.

```
string *names = new string[n];
```

- When you're done with the array, use the special form of the `delete` operator to remove the array from memory:

```
delete [] names;
```

`char*` and `char**`

- Recall that C programs didn't have C++ style strings, but instead had arrays of characters.
- The declaration

```
char *cstr;
```

is for a dynamic character array, a C-string.

- If you have a dynamic array of C-strings, you need a pointer to a pointer of characters:

```
char **cstr_array;
```

Assignment #4. Big Pi

- You will compute and print the first 1,000 decimal digits of pi.
  - Algorithm: Nonic convergence at https://en.wikipedia.org/wiki/Borwein's_algorithm

- You will use the Multiple-Precision Integers and Rationals (MPIR) library.
  - http://mpir.org/
  - The library is distributed as C source files.

- Use the library to create and work with numbers with arbitrarily long precision.

Assignment #4. Big Pi, *cont'd*

- You will learn how to download the source files, compile them, and configure, build, and install the MPIR library.

- Useful skills to have, because you will most likely need to use other libraries in the future.
  - graphics libraries
  - circuit simulation libraries
  - numerical computing libraries
  - etc.

Assignment #4. Big Pi, *cont'd*

- Building and installing the MPIR library is straightforward on Linux and Mac OS.

- Therefore, if you are on Windows, use VirtualBox to run Linux as a virtual machine.
  - VirtualBox: https://www.virtualbox.org/wiki/VirtualBox
  - Debian Linux: https://www.debian.org/
  - Ubuntu Linux: https://www.ubuntu.com/

- Download and install a Linux disk image (**.iso** file) into VirtualBox.

Assignment #4. Big Pi, *cont'd*

- Please work together to help each other to build and install MPIR.
- Programs must be individual work, as usual.
- Extra credit: Compute one million digits of pi.