CS 142
C++ Pointers

Rhodes College

Announcements

- Reminder
  - Program 6 due Monday, April 6th by 11:55pm

Demo

- simple_ptr_ex.cpp in Public directory under C++ -> pointers

Pointers and the Address Operator

- Each variable in a program is stored at a unique address in memory
- Use the address operator **&** to get the address of a variable:

```
int num = -23;
cout << &num; // prints address
              // in hexadecimal
```

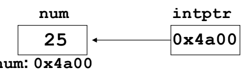
- The address of a memory location is a pointer

Pointer Variables

- Assignment:

```
int num = 25;
int *intptr;
intptr = &num;
```

- Memory layout:



address of `num`: `0x4a00`

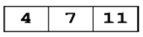
- Can access **`num`** using **`intptr`** and indirection operator **`*`**:

```
cout << intptr;  // prints 0x4a00
cout << *intptr; // prints 25
```

Relationship Between Arrays and Pointers

- Array name is starting address of array

```
int vals[] = {4, 7, 11};
```



starting address of `vals`: `0x4a00`

```
cout << vals;    // displays 0x4a00
cout << vals[0]; // displays 4
```

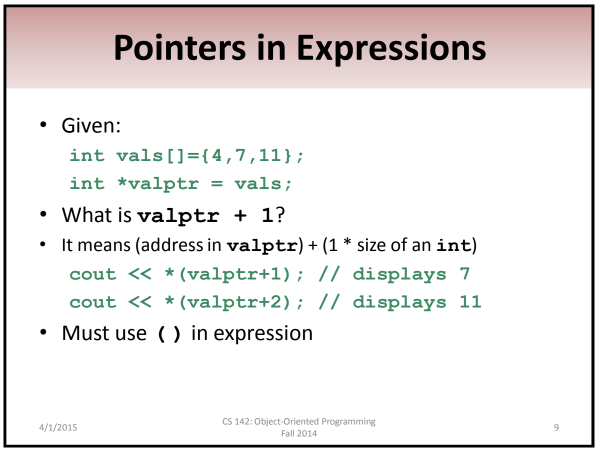
Arrays & Pointers

- Array name can be used as a pointer constant

```
int vals[] = {4, 7, 11};
cout << *vals;    // displays 4
```

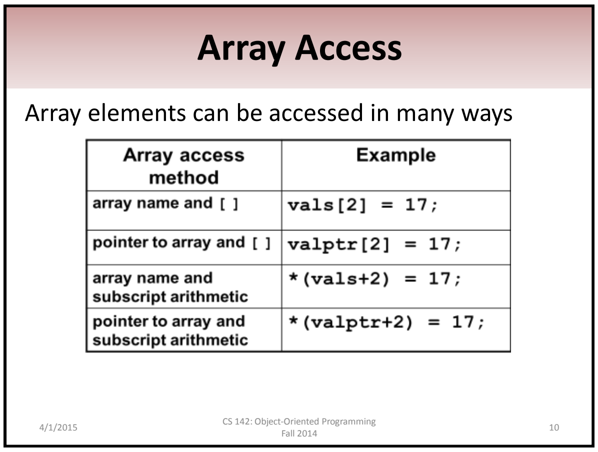
- Pointer can be used as an array name

```
int *valptr = vals;
cout << valptr[1]; // displays 7
```



Pointers in Expressions

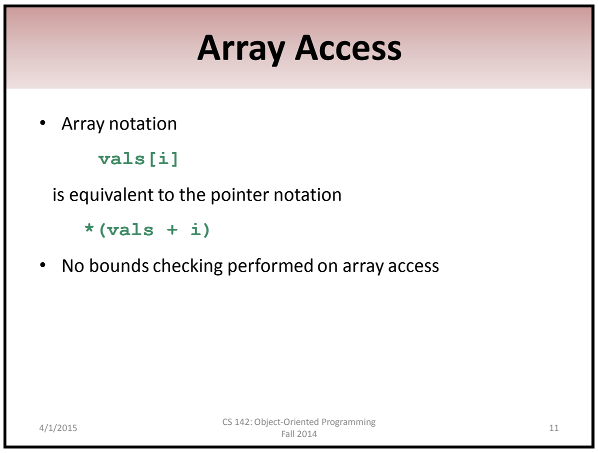
- Given:
```
int vals[]={4,7,11};
int *valptr = vals;
```
- What is **`valptr + 1`**?
- It means (address in **`valptr`**) + (1 * size of an **`int`**)
```
cout << *(valptr+1); // displays 7
cout << *(valptr+2); // displays 11
```
- Must use **`( )`** in expression



Array Access

Array elements can be accessed in many ways

Array access method	Example
array name and [ ]	`vals[2] = 17;`
pointer to array and [ ]	`valptr[2] = 17;`
array name and subscript arithmetic	`*(vals+2) = 17;`
pointer to array and subscript arithmetic	`*(valptr+2) = 17;`



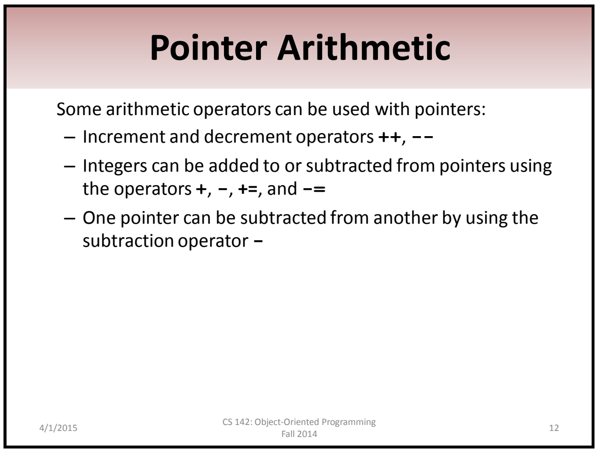
Array Access

- Array notation

  `vals[i]`

  is equivalent to the pointer notation

  `*(vals + i)`
- No bounds checking performed on array access



Pointer Arithmetic

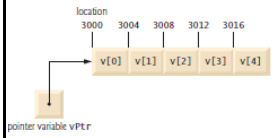
Some arithmetic operators can be used with pointers:
- Increment and decrement operators **++**, **--**
- Integers can be added to or subtracted from pointers using the operators **+**, **-**, **+=**, and **-=**
- One pointer can be subtracted from another by using the subtraction operator **-**

Pointer Arithmetic

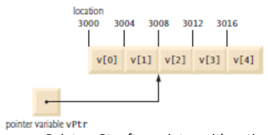
Assume that array int v[5] has been declared and that its first element is at memory location 3000.

```
int *vPtr = v;
int *vPtr = &v[ 0 ];
```

```
vPtr += 2;
```



Array v and a pointer variable int *vPtr that points to v.



Pointer vPtr after pointer arithmetic.

Pointer Arithmetic

Assume the variable definitions

```
int vals[]={4,7,11};
int *valptr = vals;
```

Examples of use of **++** and **--**

```
valptr++; // points at 7
valptr--; // now points at 4
```

More on Pointer Arithmetic

Assume the variable definitions:

```
int vals[]={4,7,11};
int *valptr = vals;
```

Example of the use of **+** to add an int to a pointer:

```
cout << *(valptr + 2); // prints 11
```

Example of use of +=:

```
valptr = vals; // points at 4
valptr += 2;   // points at 11
```

Example of pointer subtraction

```
valptr += 2;
cout << valptr - val; // prints 2; the number
                      // of ints between valptr
                      // and val.
```

Pointers as Function Parameters

- A pointer can be a parameter
- Works like a reference parameter to allow change to argument from within function
- A pointer parameter must be explicitly dereferenced to access the contents at that address

Pointers as Function Parameters

Requires:

1) asterisk `*` on parameter in prototype and heading
```
void getNum(int *ptr);
```
2) asterisk `*` in body to dereference the pointer
```
cin >> *ptr;
```
3) address as argument to the function
```
getNum(&num);
```

Pointers as Function Parameters

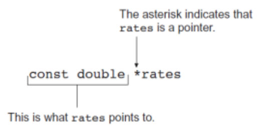
```
void swap(int *x, int *y)
{
   int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
int num1 = 2, num2 = -3;
swap(&num1, &num2);
```

Returning Pointers from Functions

- Pointer can be return type of function
  ```
  int* newNum();
  ```
- Function must not return a pointer to a local variable in the function
- Function should only return a pointer
  - to data that was passed to the function as an argument
  - to dynamically allocated memory (we'll talk about dynamically allocated memory next week)

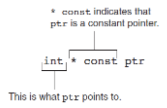
Pointers to Constants

- **Pointer to a constant**: cannot change the value that is pointed at
  - Must use **`const`** keyword in pointer definition:
    ```
    const double taxRates[] = {0.65, 0.8, 0.75};
    const double *ratePtr;
    ```
  - Use **`const`** keyword for pointers in function headers to protect data from modification within function


The asterisk indicates that rates is a pointer.
const double *rates
This is what rates points to.

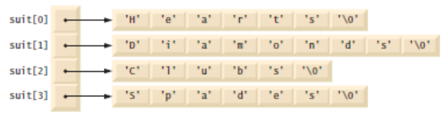
Constant Pointer

- **Constant pointer:** address in pointer cannot change once pointer is initialized
  - Defined with **`const`** keyword adjacent to variable name:
    ```
    int classSize = 24;
    int * const classPtr = &classSize;
    ```
  - Must be initialized when defined
  - Can be used without initialization as a function parameter
    - Initialized by argument when function is called
    - Function can receive different arguments on different calls
  - While the <u>address</u> in the pointer cannot change, the <u>data</u> at that address may be changed


* const indicates that ptr is a constant pointer.
int * const ptr
This is what ptr points to.

Arrays of Pointers

```
const char * const suit[ 4 ] =
   { "Hearts", "Diamonds", "Clubs", "Spades" };
```


suit[0] 'H' 'e' 'a' 'r' 't' 's' '\0'
suit[1] 'D' 'i' 'a' 'm' 'o' 'n' 'd' 's' '\0'
suit[2] 'C' 'l' 'u' 'b' 's' '\0'
suit[3] 'S' 'p' 'a' 'd' 'e' 's' '\0'

The const char * portion of the declaration indicates that each element of array suit is of type “pointer to char constant data.”

Demo

- Public directory under C++ -> pointers
  - pointers2.cpp //examples of more pointers
  - arraysWithPointers.cpp
  - swap.cpp //different function calls with pointers