UCLA

CS 31:
Introduction To Computer Science I

Howard A. Stahl

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

Assuming Student( ) Resolves…

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

Allocates Memory?

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student s( "Sam Smith", 104604956 );
Student * ptrS;
ptrS = &s;
ptrS = new Student( "Sally Jones", 204705987 );
Student array[ 5 ];
```

A Few Odds And Ends…

```
Student * ptrArray;
ptrArray = new Student[ 5 ];
```

A Few Odds And Ends…

```
Student * ptrArray;
ptrArray = new Student[ 5 ];
```

A Few Odds And Ends…

```
Student * ptrArray;
ptrArray = new Student[ 5 ];
```

A Few Odds And Ends…

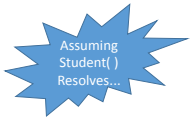
```
Student * ptrArray;
ptrArray = new Student[ 5 ];
```

A Few Odds And Ends…

```
Student * ptrArray;
ptrArray = new Student[ 5 ];
```

A Few Odds And Ends…

```
Student * ptrArray;
ptrArray = new Student[ 5 ];
```



A Few Odds And Ends…

```
Student * ptrArray;
ptrArray = new Student[ 5 ];
```

A Few Odds And Ends…

```
Student * ptrArray;
ptrArray = new Student[ 5 ];
```

A Few Odds And Ends…

```
Student * ptrArray;
ptrArray = new Student[ 5 ];
```

A Few Odds And Ends…

```
Student * ptrArray;
ptrArray = new Student[ 5 ];
```

Equivalencies

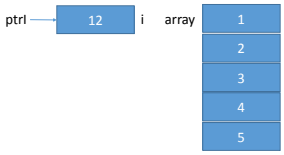
- Arrays Are Pointer Variables

```
int array[5]={1,2,3,4,5};
int i = 12;
int * ptrI = &i;
```

Equivalencies

- Arrays Are Pointer Variables

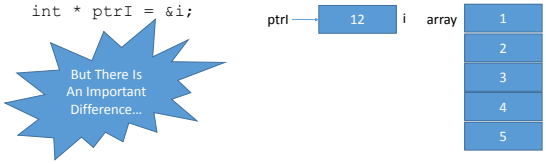
```
int array[5]={1,2,3,4,5};
int i = 12;
int * ptrI = &i;
```



Equivalencies

- Arrays Are Pointer Variables

```
int array[5]={1,2,3,4,5};
int i = 12;
int * ptrI = &i;
```



Equivalencies

- Arrays Are Pointer Variables

```
int array[5]={1,2,3,4,5};
int i = 12;
int * ptrI = &i;
```

- Pointers Can Move But Arrays Can't…

```
ptrI = &array[1];
int array2[5]={7,8,9,10,11};
array = array2;
```

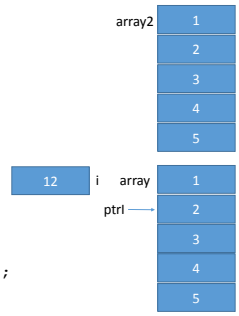
Equivalencies

- Arrays Are Pointer Variables

```
int array[5]={1,2,3,4,5};
int i = 12;
int * ptrI = &i;
```

- Pointers Can Move But Arrays Can't…

```
ptrI = &array[1];
int array2[5]={7,8,9,10,11};
array = array2;
```



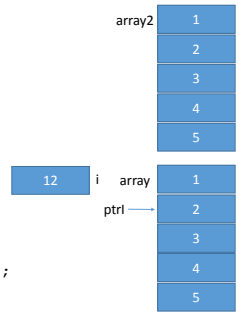
Equivalencies

- Arrays Are Pointer Variables

```
int array[5]={1,2,3,4,5};
int i = 12;
int * ptrI = &i;
```

- Pointers Can Move But Arrays Can't…

```
ptrI = &array[1];
int array2[5]={7,8,9,10,11};
array = array2;
```



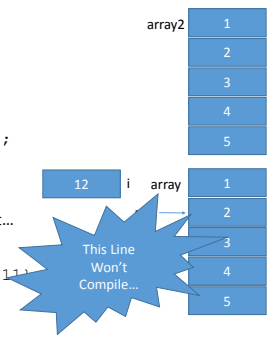
Equivalencies

- Arrays Are Pointer Variables

```
int array[5]={1,2,3,4,5};
int i = 12;
int * ptrI = &i;
```

- Pointers Can Move But Arrays Can't…

```
ptrI = &array[1];
int array2[5]={7,8,9,10,11};
array = array2;
```



Equivalencies

- Arrays Are Pointer Variables
- Pointers Can Move But Arrays Can't Move Because They Are Fixed Pointers
- The Arrow Cannot Move (Be Changed)
- The Box It Points To Change Be Changed However…
- `int array[ 5 ]` Is Implemented As A: `int * const array`

The Issue Of `const`

- `const` Always Means "Fixed"
- The Question Is "What Are We Fixing In…"

The Issue Of `const`

- `const` Always Means "Fixed"
- The Question Is "What Are We Fixing In…"

Example #1:

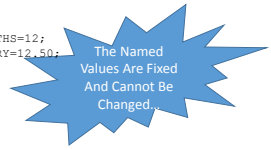
```
const int NUM_MONTHS=12;
const double SALARY=12.50;
```

The Issue Of `const`

- `const` Always Means "Fixed"
- The Question Is "What Are We Fixing In…"

Example #1:

```
const int NUM_MONTHS=12;
const double SALARY=12.50;
```



The Issue Of `const`

- `const` Always Means "Fixed"
- The Question Is "What Are We Fixing In…"

Example #1:

```
const int NUM_MONTHS=12;
const double SALARY=12.50;
```

The Named Values Are Fixed And Cannot Be Changed…

`const` Applies To The Datatype `int` Or `double`…

The Issue Of `const`

- `const` Always Means "Fixed"
- The Question Is "What Are We Fixing In…"

Example #1:

```
const int NUM_MONTHS=12;
const double SALARY=12.50;
```

Example #2:

```
void foo( const int & i);
```

The Issue Of `const`

- `const` Always Means "Fixed"
- The Question Is "What Are We Fixing In…"

Example #1:

```
const int NUM_MONTHS=12;
const double SALARY=12.50;
```

Example #2:

```
void foo( const int & i);
```

The Parameter i's Value Is Fixed And Cannot Be Changed…

The Issue Of `const`

- `const` Always Means "Fixed"
- The Question Is "What Are We Fixing In…"

Example #1:

```
const int NUM_MONTHS=12;
const double SALARY=12.50;
```

The Parameter i's Value Is Fixed And Cannot Be Changed…

Example #2:

```
void foo( const int & i);
```

`const` Applies To The Datatype `int &`…

The Issue Of `const`

- `const` Always Means "Fixed"
- The Question Is "What Are We Fixing In…"

Example #1:

```
const int NUM_MONTHS=12;
const double SALARY=12.50;
```

Example #2:

```
void foo( const int & i);
```

Example #3:

```
class Foo {
public:
  int getFoo( ) const;
private:
  int mFoo;
};
```

The Issue Of `const`

- `const` Always Means "Fixed"
- The Question Is "What Are We Fixing In…"

Example #1:

```
const int NUM_MO
const do
```

`getFoo` Is Read-Only And Cannot Change Any Member Variable Values…

Example #2:

```
void foo( const int & i);
```

Example #3:

```
class Foo {
public:
  int getFoo( ) const;
private:
  int mFoo;
};
```

The Issue Of `const`

- `const` Always Means "Fixed"
- The Question Is "What Are We Fixing In…"

Example #1:

```
const int NUM_MO
const do
```

Example #2:

```
void foo(
```

Example #3:

```
class Foo {
public:
  int getFoo( ) const;
private:
  int mFoo;
};
```

`getFoo` Is Read-Only And Cannot Change Any Mem Variable Values…

`const` Applies To The Method `getFoo( )`…

The Issue Of `const`

- When Working With Pointers, What Can We Mark `const`?

```
int i = 12;
int * ptrI = &i;
```

The Issue Of `const`

- When Working With Pointers, What Can We Mark `const`?

```
int i = 12;
int * ptrI = &i;
```

- We Could Mark The Arrow `const`

The Issue Of `const`

- When Working With Pointers, What Can We Mark `const`?

```
int i = 12;
int * ptrI = &i;
```

- We Could Mark The Arrow `const`
  - In Which Case `ptrI` Cannot Be Pointed Anywhere Else…
  - In Which Case `int j = 13; ptrI = &j;` Would Be Illegal

The Issue Of `const`

- When Working With Pointers, What Can We Mark `const`?

```
int i = 12;
int * const ptrI = &i;
```

- We Could Mark The Arrow `const`
  - In Which Case `ptrI` Cannot Be Pointed Anywhere Else…
  - In Which Case `int j = 13; ptrI = &j;` Would Be Illegal

The Issue Of `const`

- When Working With Pointers, What Can We Mark `const`?

```
int i = 12;
int * const ptrI = &i;
```

- We Could Mark The Arrow `const`
  - In Which Case `ptrI` Cannot Be Pointed Anywhere
  - In Which Case `int j = 13; ptrI = &`

`const` Applies To The Datatype `int *` …

The Issue Of `const`

- When Working With Pointers, What Can We Mark `const`?

```
int i = 12;
int * ptrI = &i;
```

- We Could Mark The Box `const`

The Issue Of `const`

- When Working With Pointers, What Can We Mark `const`?

```
int i = 12;
int * ptrI = &i;
```

- We Could Mark The Box `const`
  - In Which Case `i` Cannot Be Changed Via `ptrI`…
  - In Which Case `*ptrI = 13;` Would Be Illegal

The Issue Of `const`

- When Working With Pointers, What Can We Mark `const`?

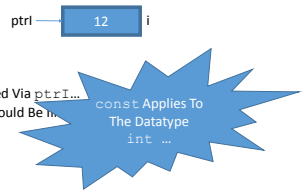
```
int i = 12;
const int * ptrI = &i;
```

- We Could Mark The Box `const`
  - In Which Case `i` Cannot Be Changed Via `ptrI`…
  - In Which Case `*ptrI = 13;` Would Be Illegal

The Issue Of `const`

- When Working With Pointers, What Can We Mark `const`?

```
int i = 12;
const int * ptrI = &i;
```



- We Could Mark The Box `const`
  - In Which Case `i` Cannot Be Changed Via `ptrI`…
  - In Which Case `*ptrI = 13;` Would Be In…

Question 5 Hint

```
char msg[100] = "UC Los Angeles";
deleteCapitals(msg);
cout << msg << endl;
```

- Should Print: `" os ngeles"`

Question 5 Hint

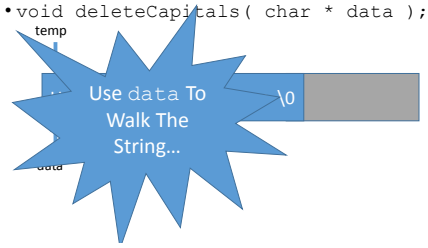
- `void deleteCapitals( char * data );`

Question 5 Hint

- `void deleteCapitals( char * data );`

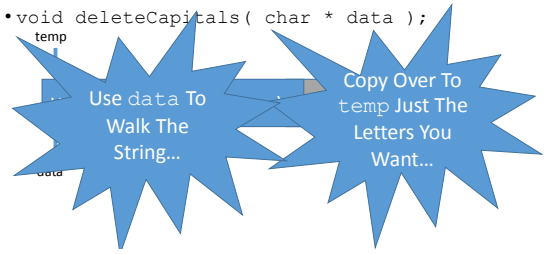
Question 5 Hint

- `void deleteCapitals( char * data );`



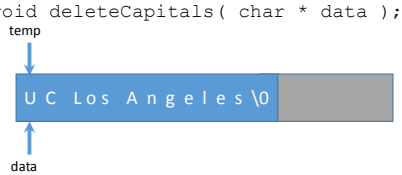
Question 5 Hint

- `void deleteCapitals( char * data );`



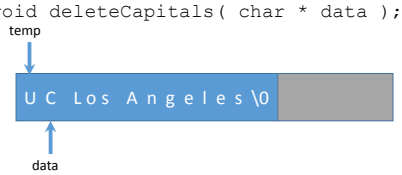
Question 5 Hint

- `void deleteCapitals( char * data );`



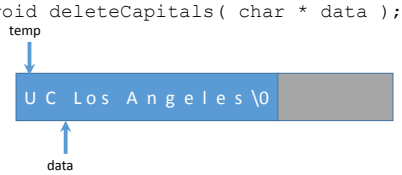
Question 5 Hint

- `void deleteCapitals( char * data );`



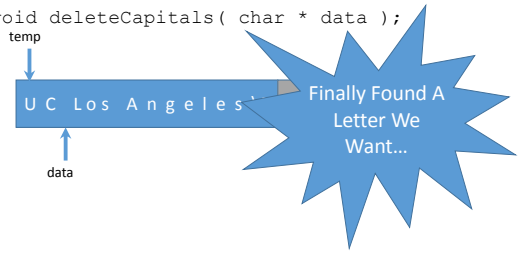
Question 5 Hint

- `void deleteCapitals( char * data );`



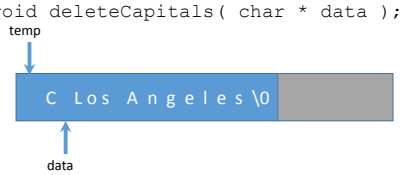
Question 5 Hint

- `void deleteCapitals( char * data );`



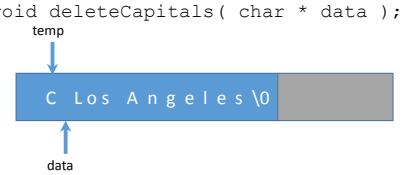
Question 5 Hint

- `void deleteCapitals( char * data );`



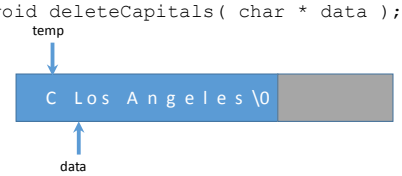
Question 5 Hint

- `void deleteCapitals( char * data );`



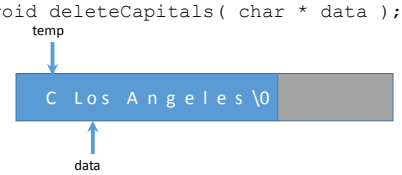
Question 5 Hint

- `void deleteCapitals( char * data );`



Question 5 Hint

- `void deleteCapitals( char * data );`

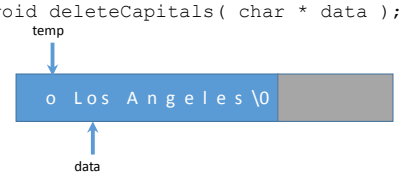


Question 5 Hint

- `void deleteCapitals( char * data );`

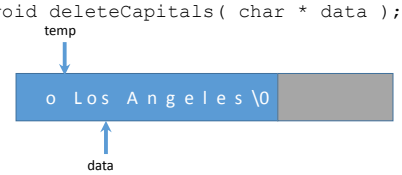
Question 5 Hint

- `void deleteCapitals( char * data );`



Question 5 Hint

- `void deleteCapitals( char * data );`



Question 5 Hint

- One Very Important Final Point!
- Be Sure To Copy Over The Ending NULL Character To Be Sure The Data You Produce Is A Valid C-String