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 ];

Allocates Memory?

Stack Or Heap?

YES
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? YES
Stack Or Heap? Stack
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?

NO
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? NO
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? YES

Stack Or Heap? Heap
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...

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 ];

Allocates Memory? YES

Stack Or Heap? Stack
A Few Odds And Ends...

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

Allocates Memory?
A Few Odds And Ends...

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

Allocates Memory? NO

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 ];

Assuming Student() Resolves...
A Few Odds And Ends...

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

Allocates Memory?

YES

Stack Or Heap?
**A Few Odds And Ends...**

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

---

**Allocates Memory?**

YES

---

**Stack Or Heap?**

Heap

---

**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;
  ```

---

```
ptr 12 array
1
2
3
4
5
```
Equivalencies

• Arrays Are Pointer Variables

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

But there is an important difference:

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

• Arrays Are Pointer Variables
  ```c
  int array[5]={1,2,3,4,5};
  int i = 12;
  int * ptrI = &i;
  ```

• Pointers Can Move But Arrays Can’t...
  ```c
  ptrI = &array[1];
  int array2[5]={7,8,9,10,11};
  array = array2;
  ```

This Line Won’t Compile...

• Arrays Are Pointer Variables
  ```c
  int array[5]={1,2,3,4,5};
  int i = 12;
  int * ptrI = &i;
  ```

• Pointers Can Move But Arrays Can’t...
  ```c
  ptrI = array[1];
  int array2[5]={7,8,9,10,11};
  array = array2;
  ```

• Arrays Are Pointer Variables
  ```c
  int array[5]={1,2,3,4,5};
  int i = 12;
  int * ptrI = &i;
  ```

• Pointers Can Move But Arrays Can’t...
  ```c
  ptrI = array[1];
  int array2[5]={7,8,9,10,11};
  array = array2;
  ```

• 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...
  ```c
  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..."

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:**
```c
const int NUM_MONTHS=12;
const double SALARY=12.50;
```

**Example #2:**
```c
void foo(const int &i);
```

The Named Values Are Fixed And Cannot Be Changed...

**Example #1:**
```c
const int NUM_MONTHS=12;
const double SALARY=12.50;
```

**Example #2:**
```c
void foo(const int &i);
```

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

**Example #1:**
```c
const int NUM_MONTHS=12;
const double SALARY=12.50;
```

**Example #2:**
```c
void foo(const int &i);
```
The Issue Of const

- const Always Means “Fixed”
- The Question Is “What Are We Fixing In...”

Example #1:
```c
const int NUM_MONTHS=12;
```

Example #2:
```c
void foo( const int &i);
```

Example #3:
```c
class Foo {
    public:
        int getFoo( ) const;
        private:
            int mFoo;
};
```

Example #1:
```c
const int NUM_MONTHS=12;
const double SALARY=12.50;
```

Example #2:
```c
void foo( const int &i);
```

Example #3:
```c
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:**
```c
const int NUM_MONTHS = 12;
const double SALARY = 12.50;
```

**Example #2:**
```c
void foo(const int &i);
```

**Example #3:**
```c
class Foo {
public:
    int getFoo() const;
private:
    int mFoo;
};
```

getFoo is read-only and cannot change any member variable values...

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

The Issue Of `const`

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

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

- We Could Mark The Arrow `const`

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

```c
ptrI = 12  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`

  - In Which Case `ptrI` Cannot Be Pointed Anywhere Else...
  - In Which Case `int j = 13; ptrI = &j;` Would Be Illegal

```
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

```
const int * ptrI = &i;
```
The Issue Of `const`

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

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

  - We Could Mark The Box `const`

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

    - In Which Case *ptrI = 13; Would Be Illegal
The Issue Of `const`

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

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

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

---

**Question 5 Hint**

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

- Should Print: "os ngeles"

---

**Question 5 Hint**

```c
void deleteCapitals( char * data );
```

```
UC Los Angeles\0
```

- `data`
• void deleteCapitals( char * data );

Use data To Walk The String…

Copy Over To temp Just The Letters You Want…
Question 5 Hint

• void deleteCapitals( char * data );

```c
void deleteCapitals( char * data ) {
    temp
    str
    temp
}
```

UC Los Angeles

7/31/16
Question 5 Hint

• void deleteCapitals( char * data );

Finally Found A Letter We Want...

Question 5 Hint

• void deleteCapitals( char * data );

Question 5 Hint

• void deleteCapitals( char * data );

Question 5 Hint

• void deleteCapitals( char * data );
Question 5 Hint

```c
void deleteCapitals( char * data );
```

```
C Los Angeles
```

```c
Finally Found A Letter We Want...
```

```
0
data
```
Question 5 Hint

• **void deleteCapitals( char * data );**

    data

    o Los Angeles\0

    temp

**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