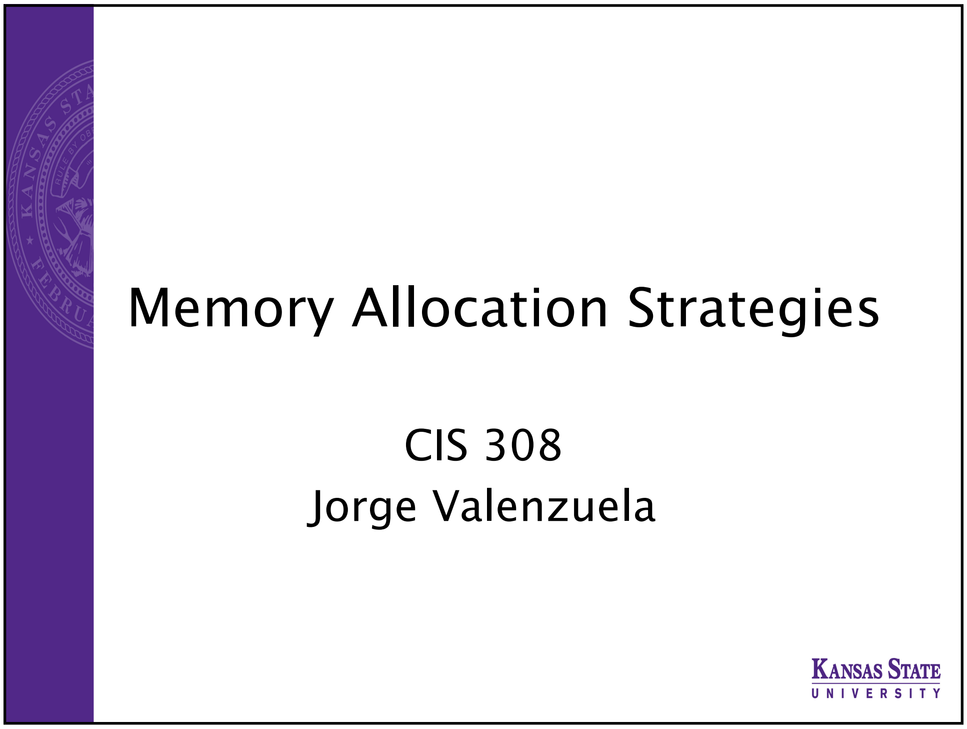
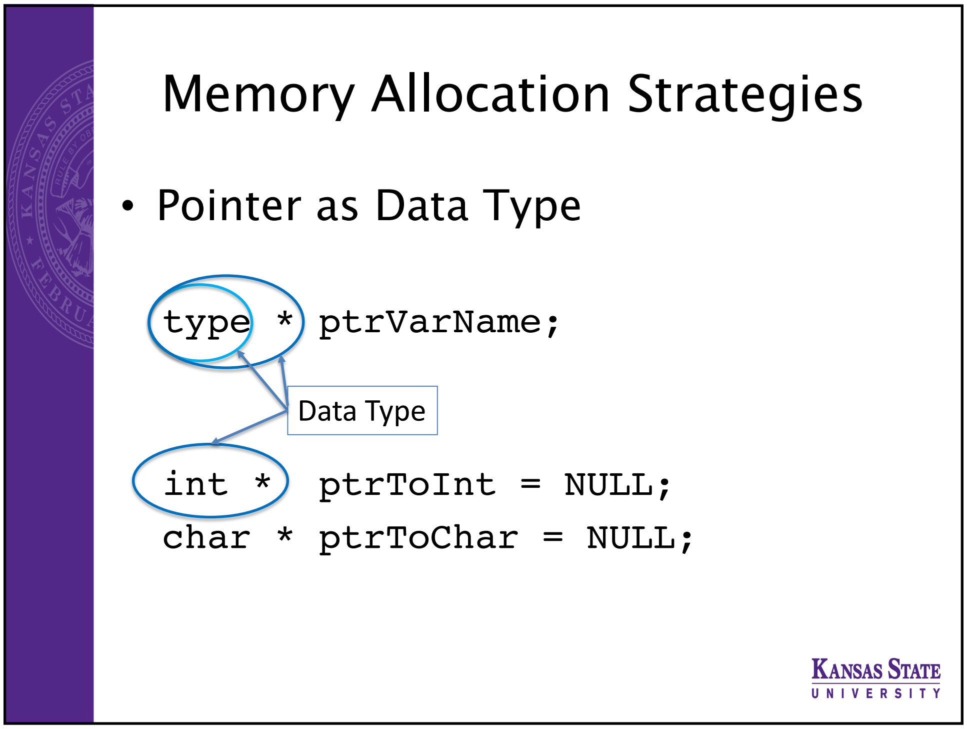


Memory Allocation Strategies

CIS 308
Jorge Valenzuela

KANSAS STATE UNIVERSITY

1



Memory Allocation Strategies

- Pointer as Data Type

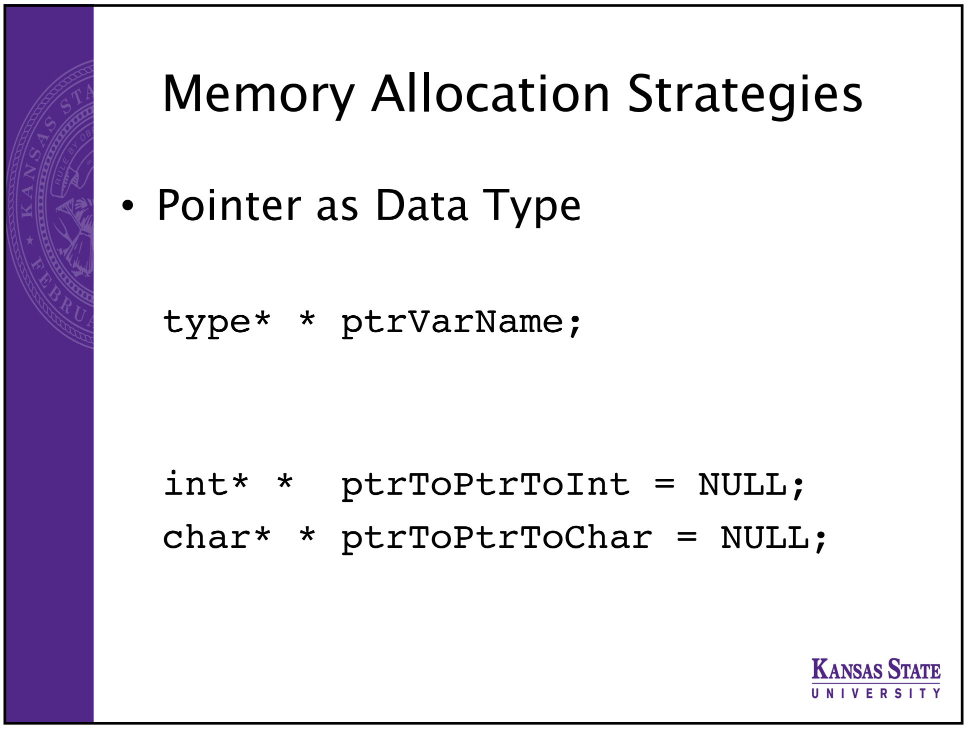
```
type * ptrVarName;
```

Data Type

```
int *  ptrToInt = NULL;
char * ptrToChar = NULL;
```

KANSAS STATE UNIVERSITY

2



Memory Allocation Strategies

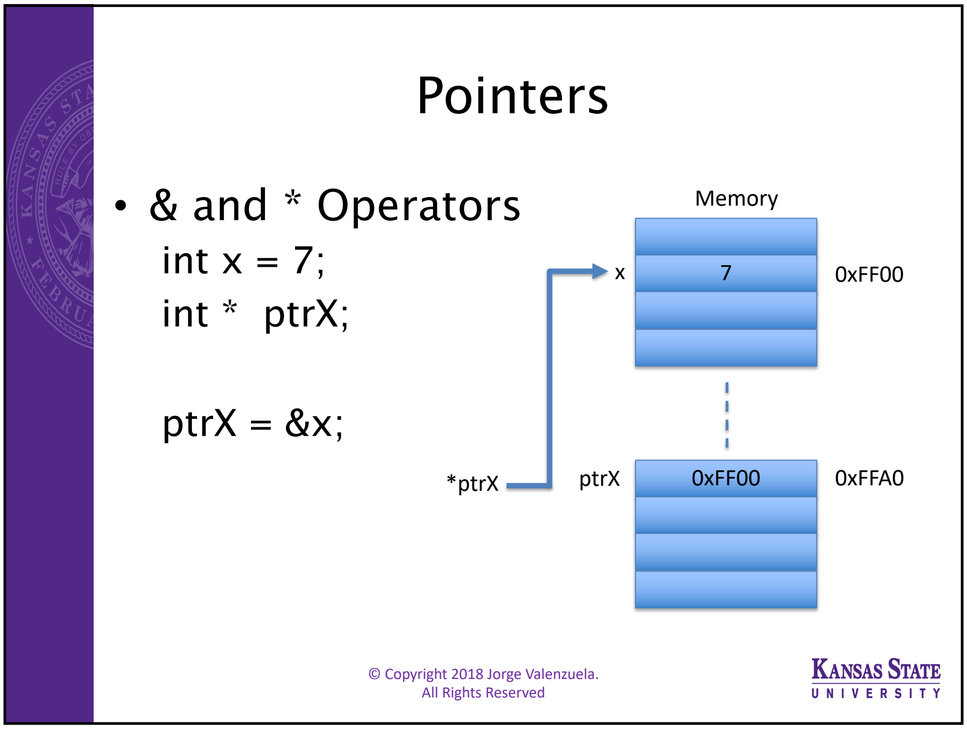
- Pointer as Data Type

```
type* * ptrVarName;


int*  *  ptrToPtrToInt = NULL;
char* * ptrToPtrToChar = NULL;
```

KANSAS STATE UNIVERSITY

3



Pointers

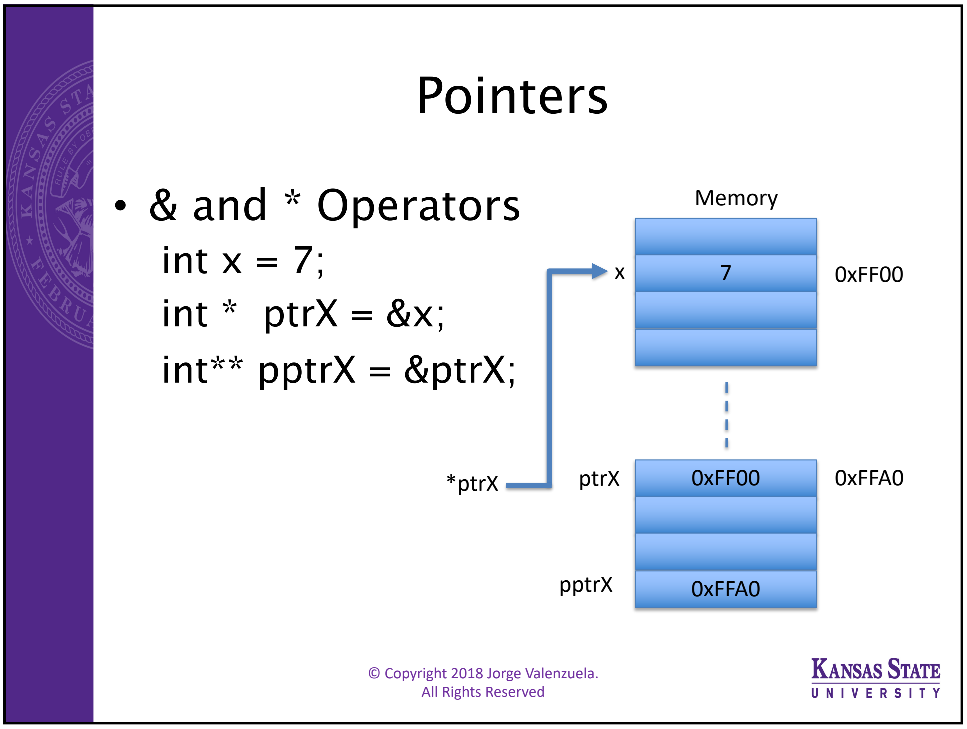
- & and * Operators

  int x = 7;
  int * ptrX;

  ptrX = &x;

KANSAS STATE UNIVERSITY

4



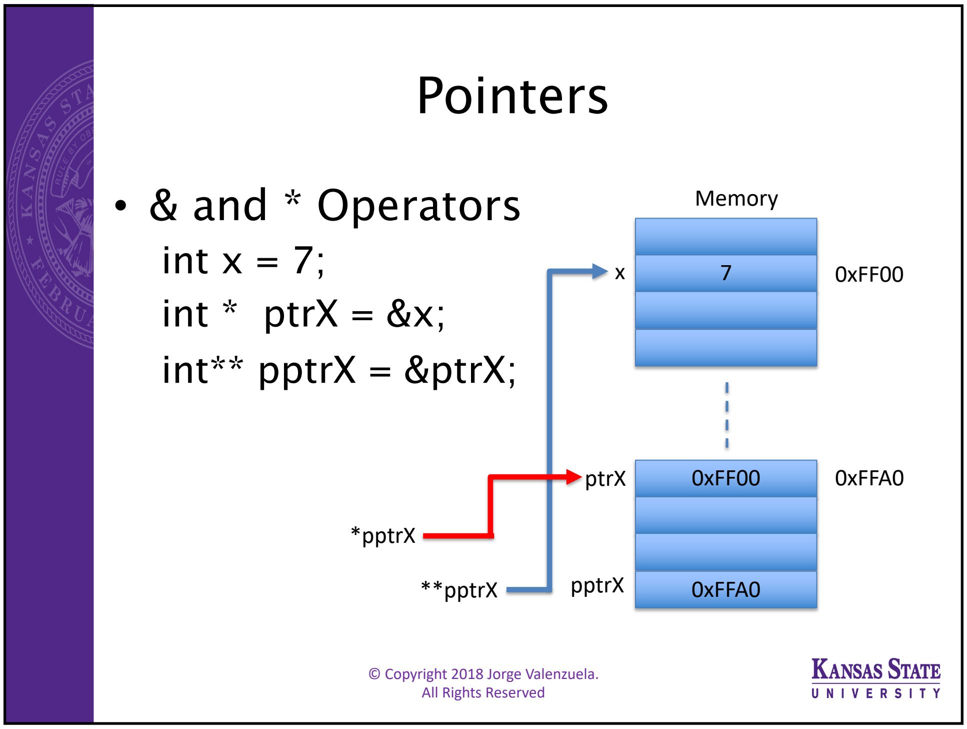
Pointers

- & and * Operators

```
int x = 7;
int *  ptrX = &x;
int** pptrX = &ptrX;
```

KANSAS STATE UNIVERSITY

5



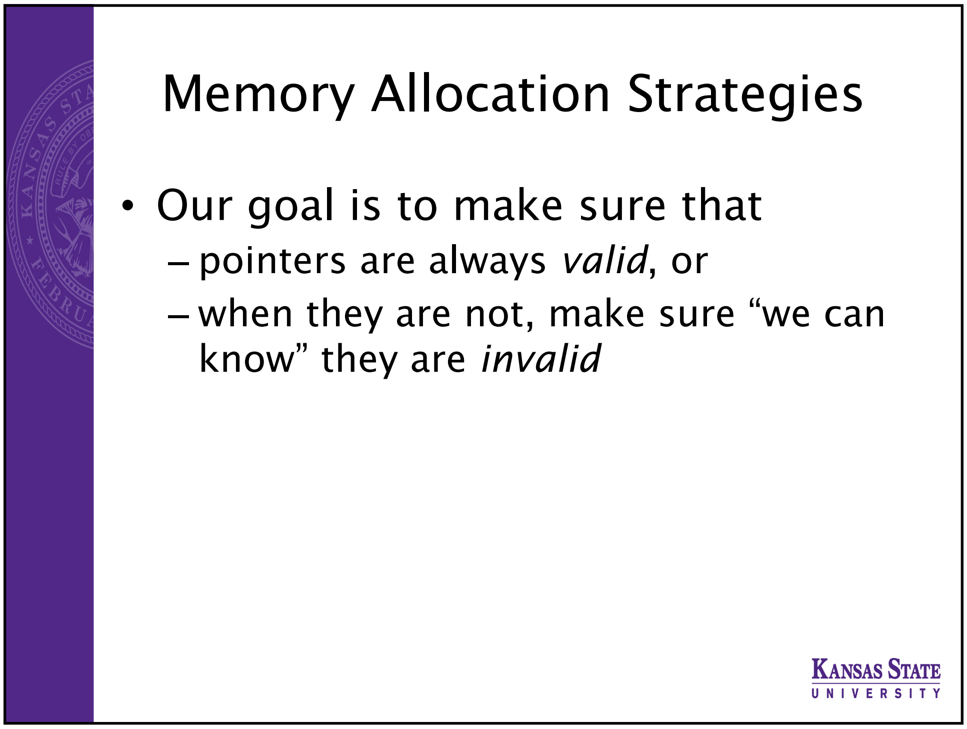
Pointers

- & and * Operators

```
int x = 7;
int *  ptrX = &x;
int** pptrX = &ptrX;
```

KANSAS STATE UNIVERSITY

6

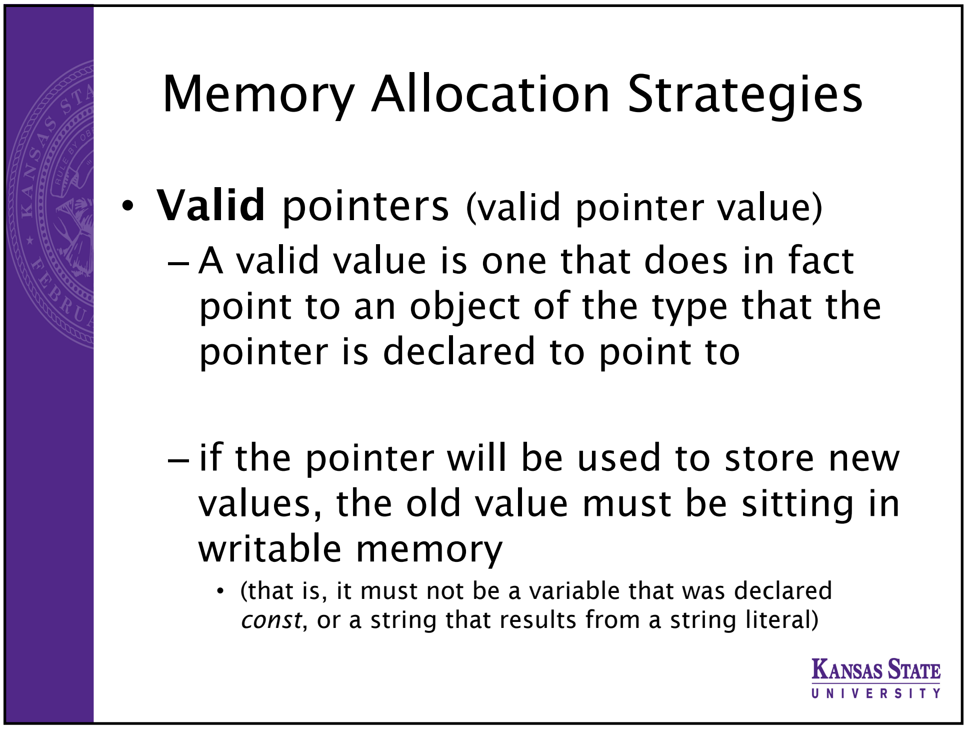


Memory Allocation Strategies

- Our goal is to make sure that
  - pointers are always *valid*, or
  - when they are not, make sure “we can know” they are *invalid*

KANSAS STATE UNIVERSITY

7

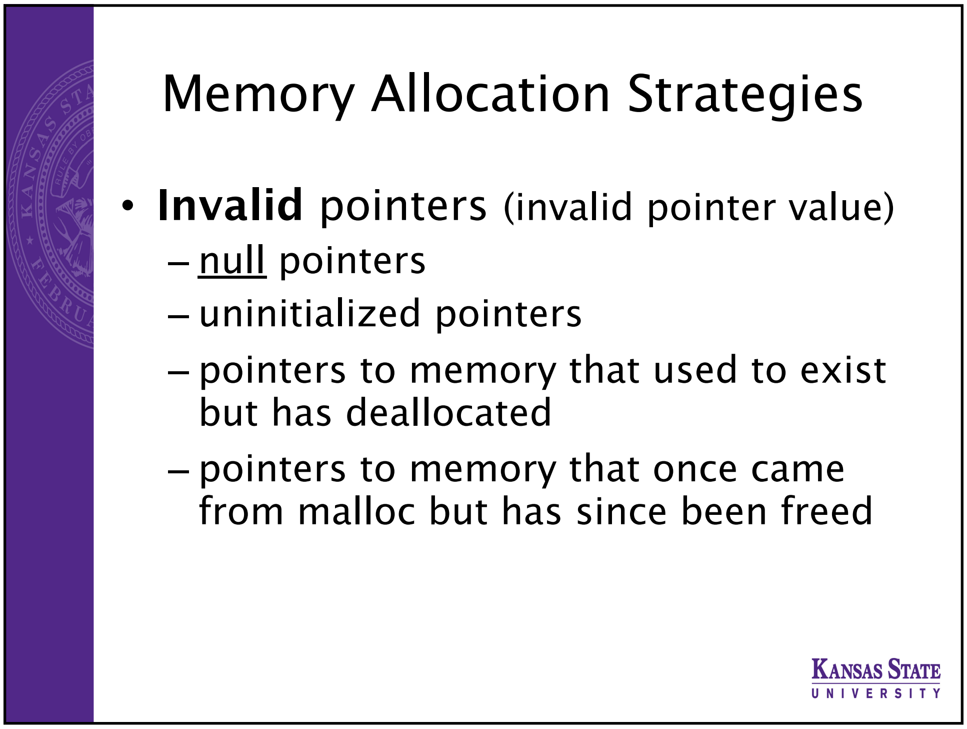


Memory Allocation Strategies

- **Valid** pointers (valid pointer value)
  - A valid value is one that does in fact point to an object of the type that the pointer is declared to point to
  - if the pointer will be used to store new values, the old value must be sitting in writable memory
    - (that is, it must not be a variable that was declared *const*, or a string that results from a string literal)

KANSAS STATE UNIVERSITY

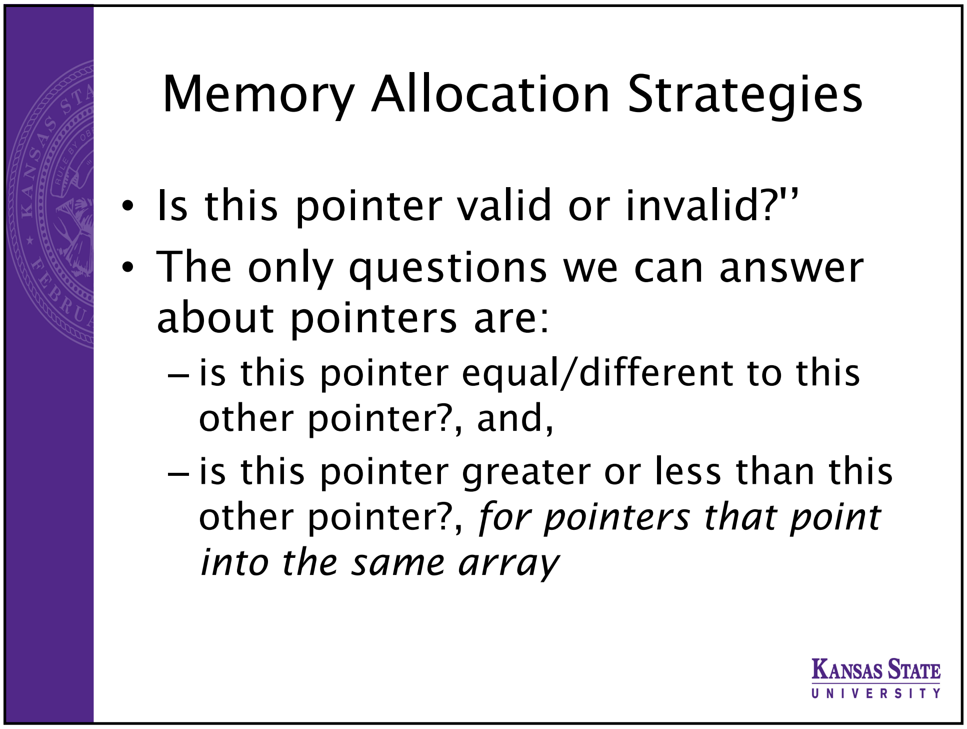
8



Memory Allocation Strategies

- **Invalid** pointers (invalid pointer value)
  - null pointers
  - uninitialized pointers
  - pointers to memory that used to exist but has deallocated
  - pointers to memory that once came from malloc but has since been freed

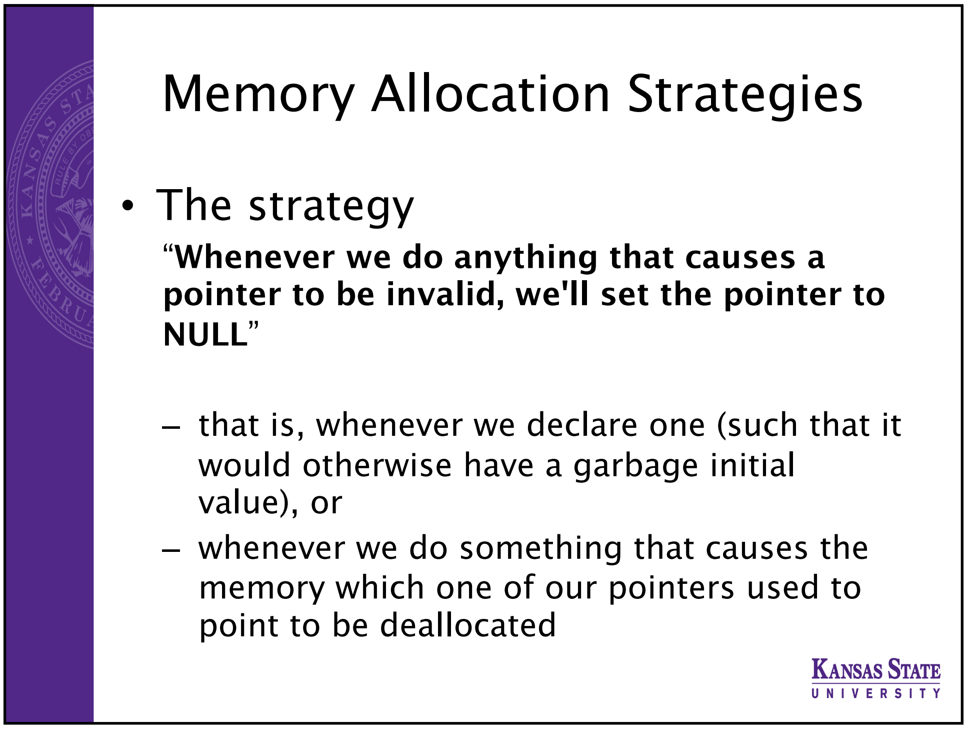
9



Memory Allocation Strategies

- Is this pointer valid or invalid?''
- The only questions we can answer about pointers are:
  - is this pointer equal/different to this other pointer?, and,
  - is this pointer greater or less than this other pointer?, *for pointers that point into the same array*

10



Memory Allocation Strategies

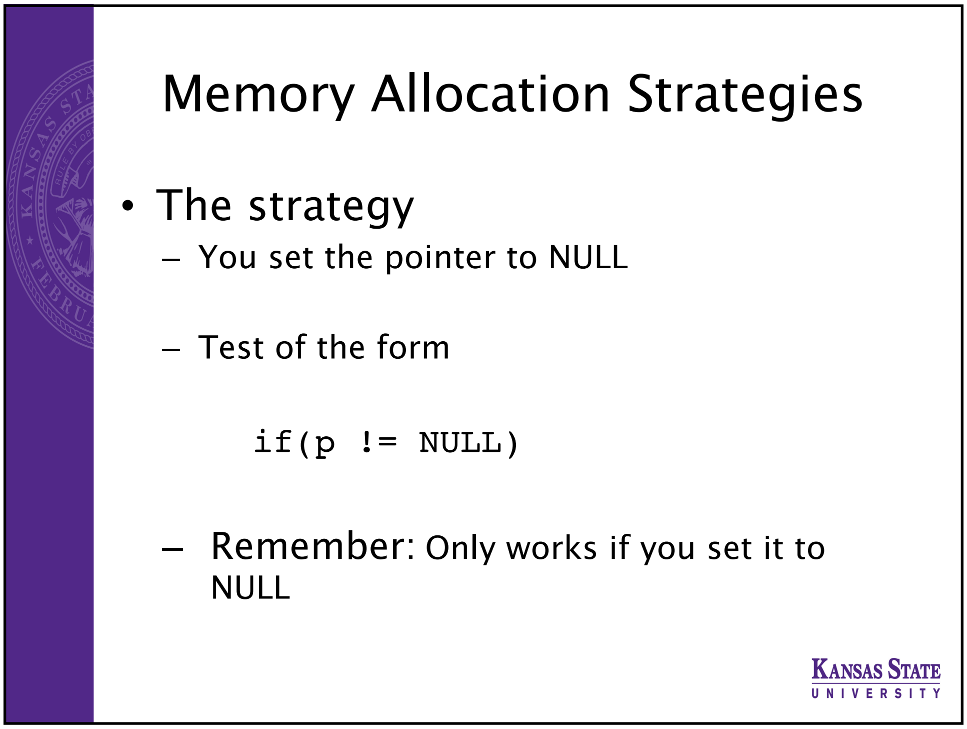
- The strategy

  “**Whenever we do anything that causes a pointer to be invalid, we'll set the pointer to NULL**”

  – that is, whenever we declare one (such that it would otherwise have a garbage initial value), or

  – whenever we do something that causes the memory which one of our pointers used to point to be deallocated

11



Memory Allocation Strategies

- The strategy

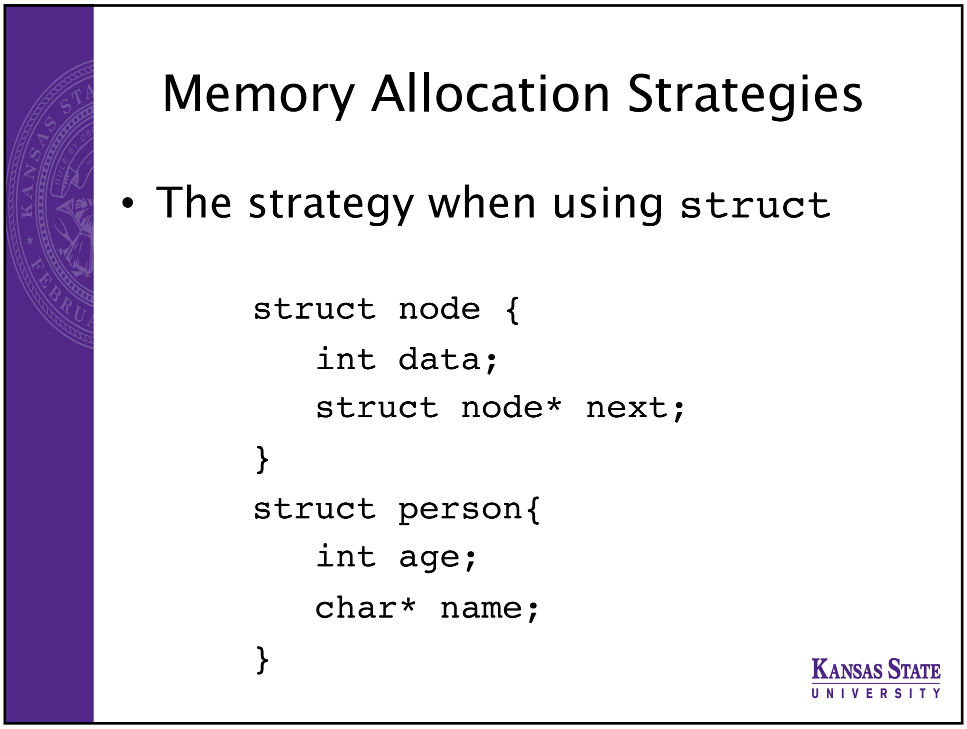
  – You set the pointer to NULL

  – Test of the form

```
if(p != NULL)
```

  – Remember: Only works if you set it to NULL

12

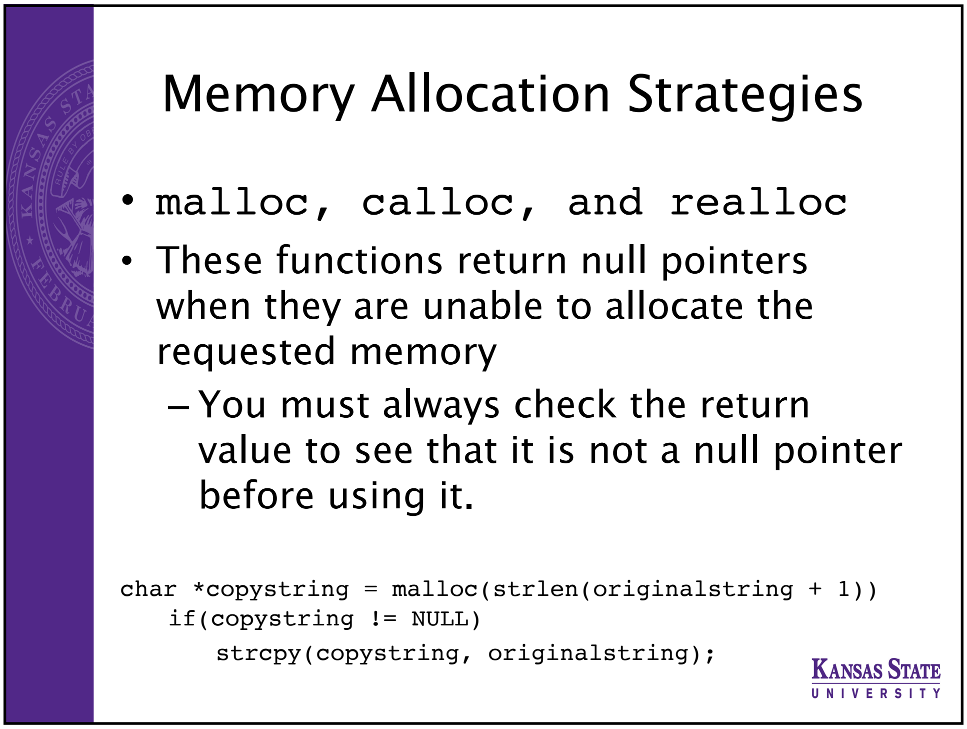


Memory Allocation Strategies

- The strategy when using `struct`

```
struct node {
   int data;
   struct node* next;
}
struct person{
   int age;
   char* name;
}
```

13

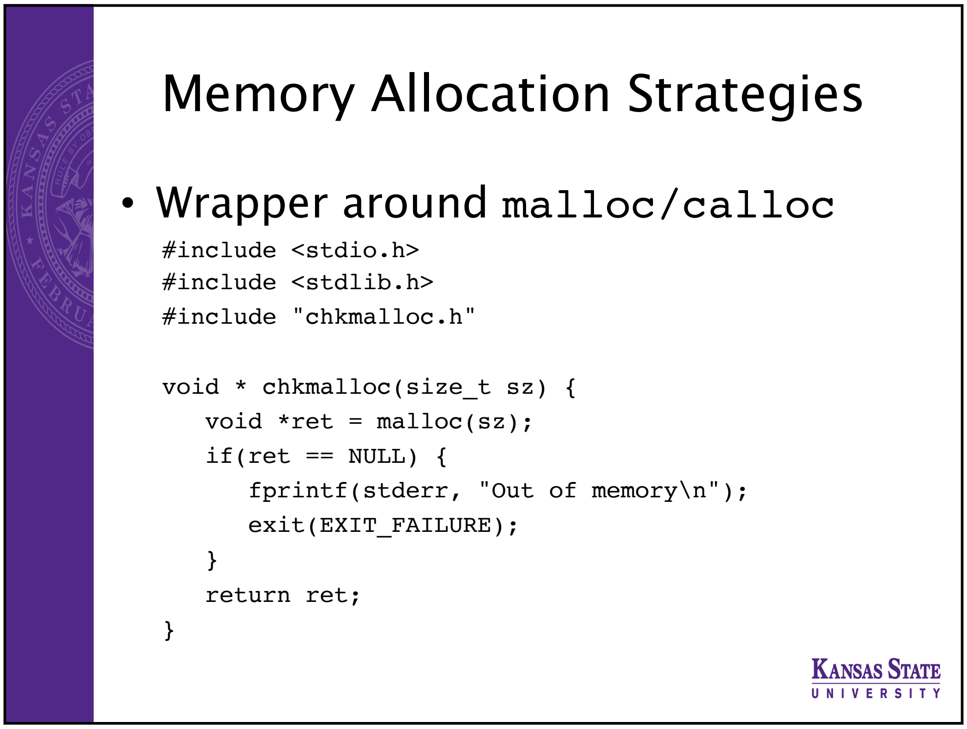


Memory Allocation Strategies

- `malloc, calloc, and realloc`
- These functions return null pointers when they are unable to allocate the requested memory
  - You must always check the return value to see that it is not a null pointer before using it.

```
char *copystring = malloc(strlen(originalstring + 1))
   if(copystring != NULL)
      strcpy(copystring, originalstring);
```

14



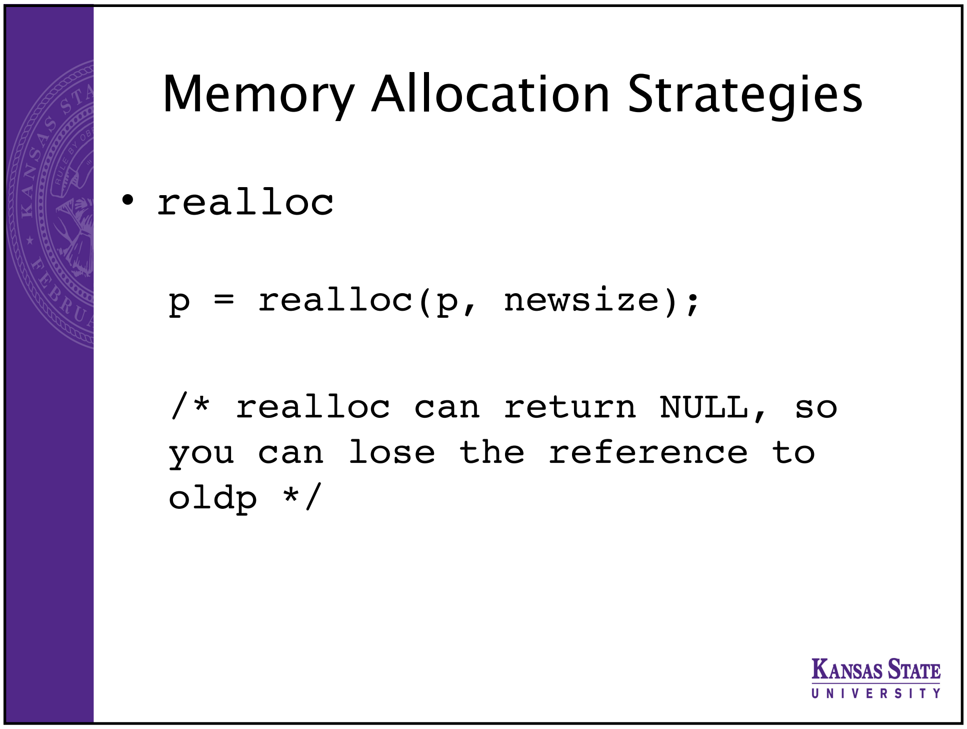
Memory Allocation Strategies

- Wrapper around `malloc/calloc`

```
#include <stdio.h>
#include <stdlib.h>
#include "chkmalloc.h"

void * chkmalloc(size_t sz) {
   void *ret = malloc(sz);
   if(ret == NULL) {
      fprintf(stderr, "Out of memory\n");
      exit(EXIT_FAILURE);
   }
   return ret;
}
```

15



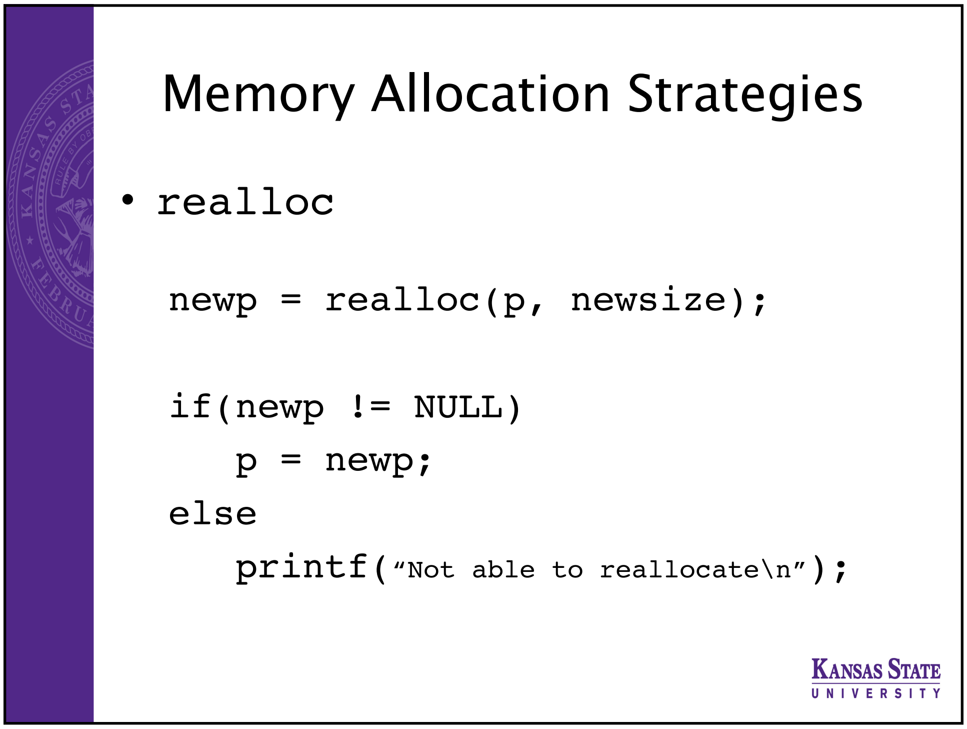
Memory Allocation Strategies

- `realloc`

```
p = realloc(p, newsize);

/* realloc can return NULL, so
you can lose the reference to
oldp */
```

16



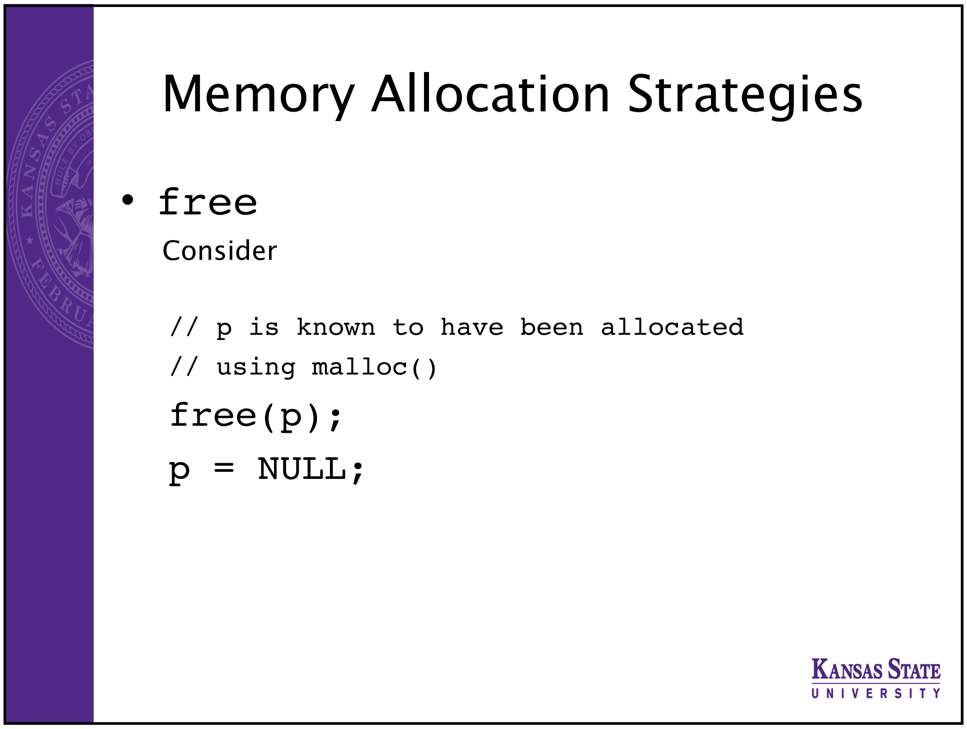
Memory Allocation Strategies

- `realloc`

```
newp = realloc(p, newsize);

if(newp != NULL)
   p = newp;
else
   printf(“Not able to reallocate\n”);
```

17



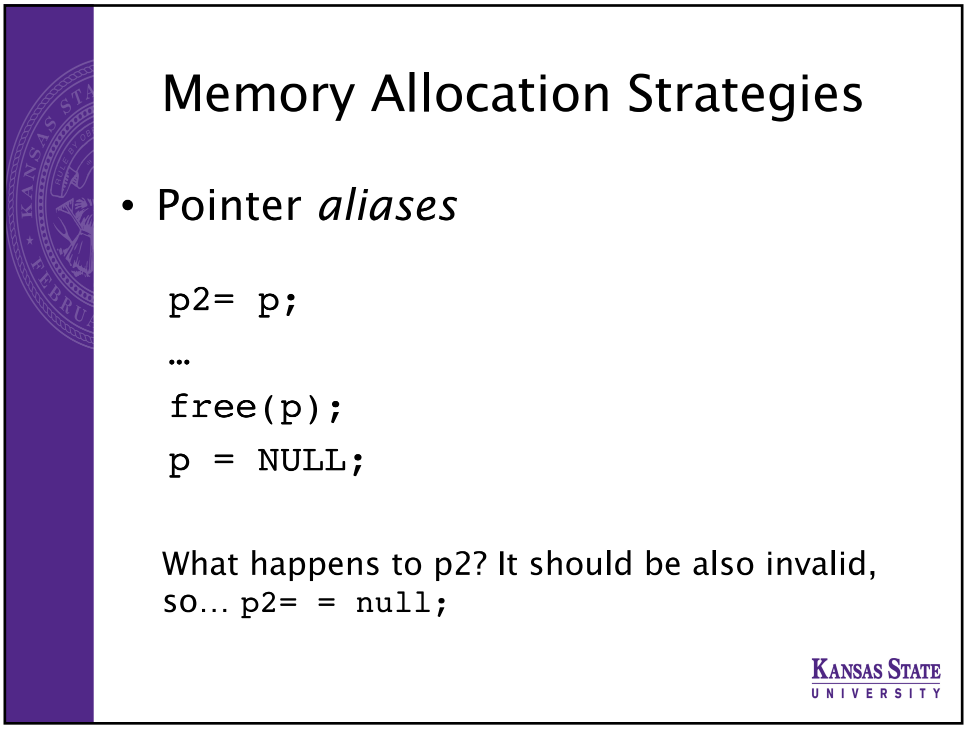
Memory Allocation Strategies

- `free`

  Consider

```
// p is known to have been allocated
// using malloc()
free(p);
p = NULL;
```

18



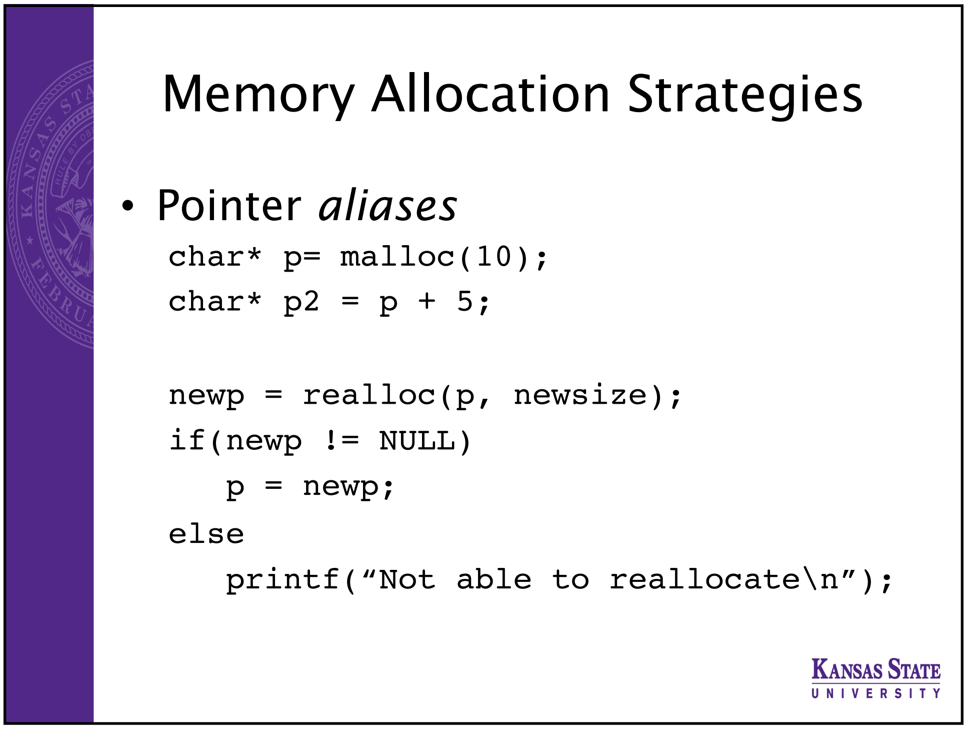
Memory Allocation Strategies

- Pointer *aliases*

```
p2= p;
…
free(p);
p = NULL;
```

What happens to p2? It should be also invalid, so… `p2= = null;`

19



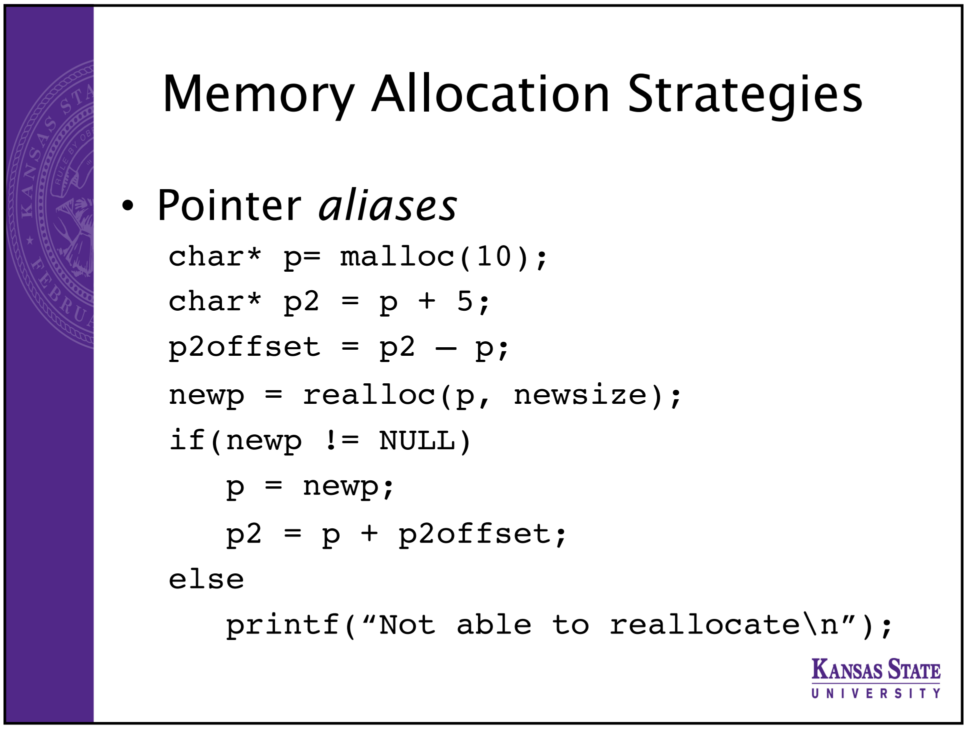
Memory Allocation Strategies

- Pointer *aliases*

```
char* p= malloc(10);
char* p2 = p + 5;

newp = realloc(p, newsize);
if(newp != NULL)
   p = newp;
else
   printf(“Not able to reallocate\n”);
```

20



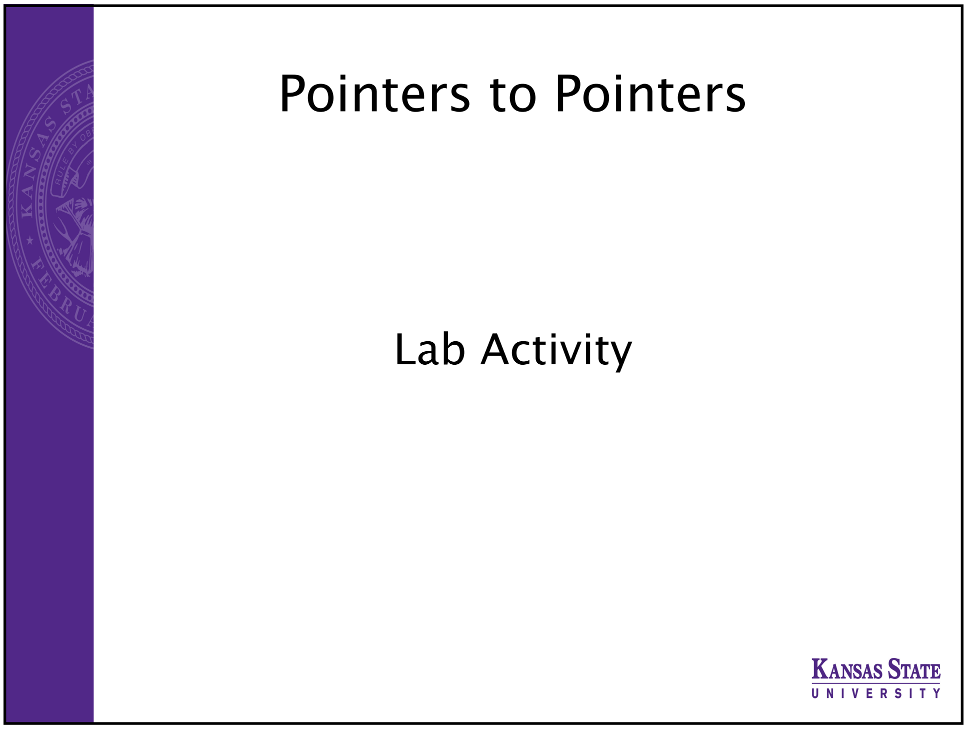
Memory Allocation Strategies

- Pointer *aliases*

```
char* p= malloc(10);
char* p2 = p + 5;
p2offset = p2 – p;
newp = realloc(p, newsize);
if(newp != NULL)
   p = newp;
   p2 = p + p2offset;
else
   printf(“Not able to reallocate\n”);
```

KANSAS STATE UNIVERSITY

21

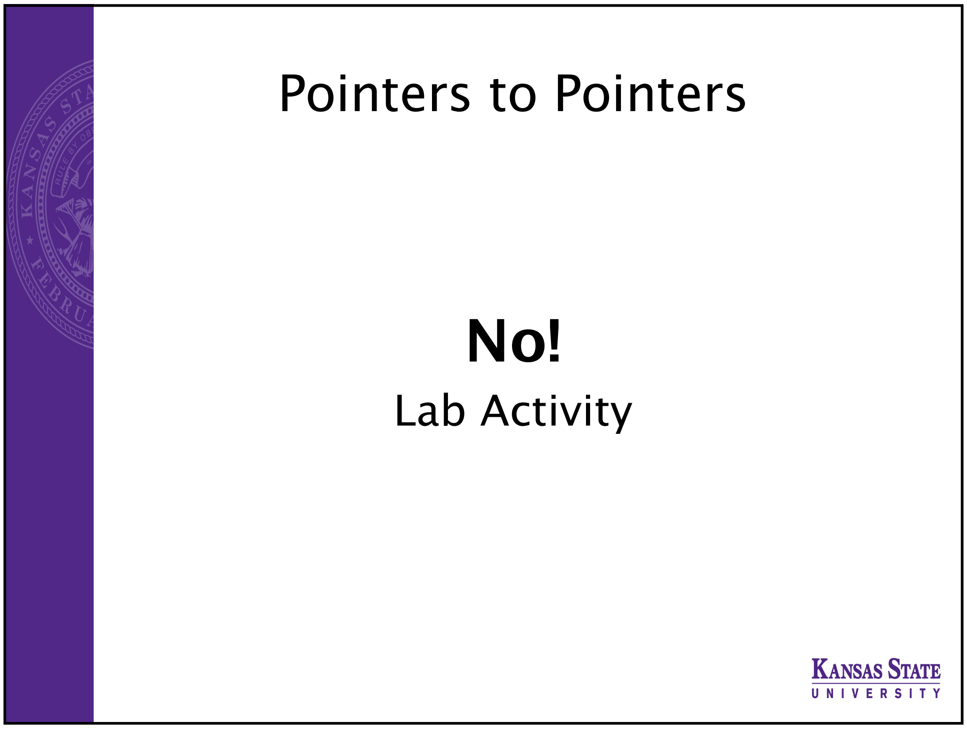


Pointers to Pointers

Lab Activity

KANSAS STATE UNIVERSITY

22



Pointers to Pointers

No!

Lab Activity

23