CIS241

System-Level Programming and Utilities

C Pointers

Erik Fredericks, frederer@gvsu.edu

Fall 2025

> **Based on material provided by Erin Carrier, Austin Ferguson, and Katherine Bowers**

Pointers

A new data type

A pointer stores a memory address

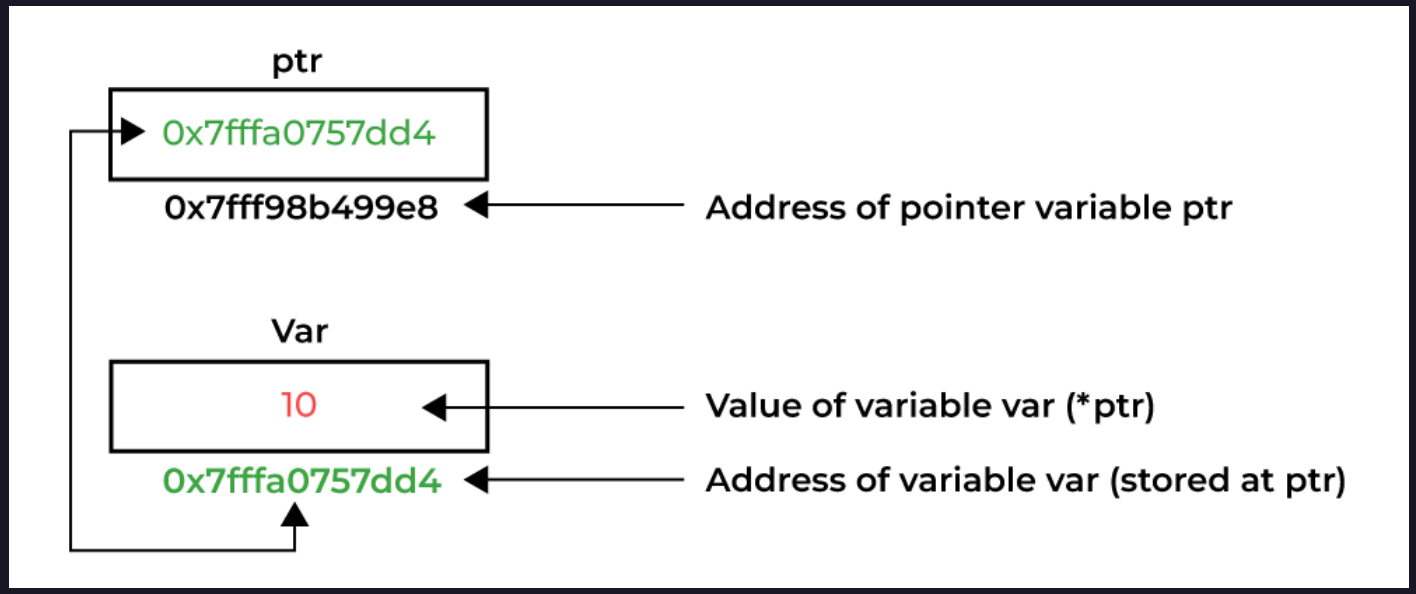
- (i.e., it "points" to a location in memory)

Pointers are associated with a particular type

- (e.g., a float pointer stores the address of a float)

Makes your life `!!FUN!!`

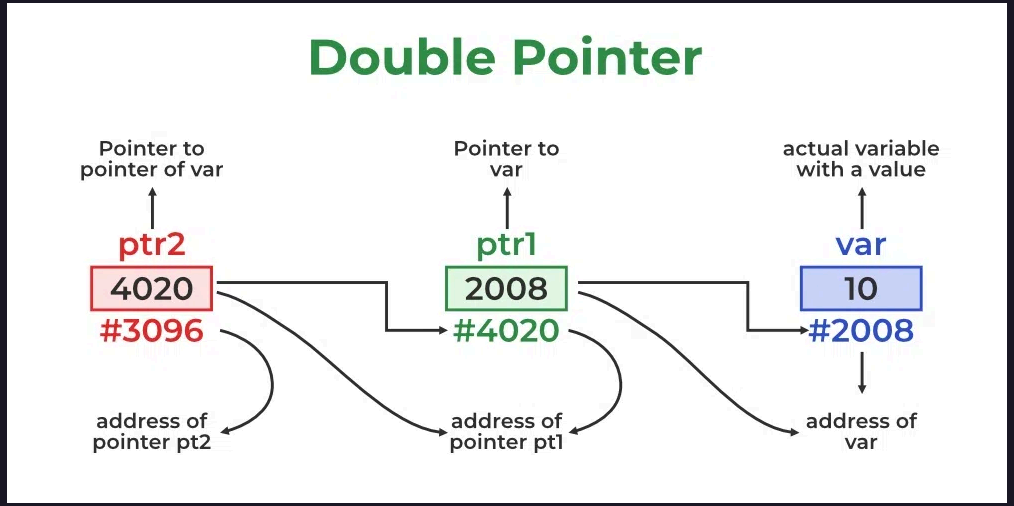
New operators!



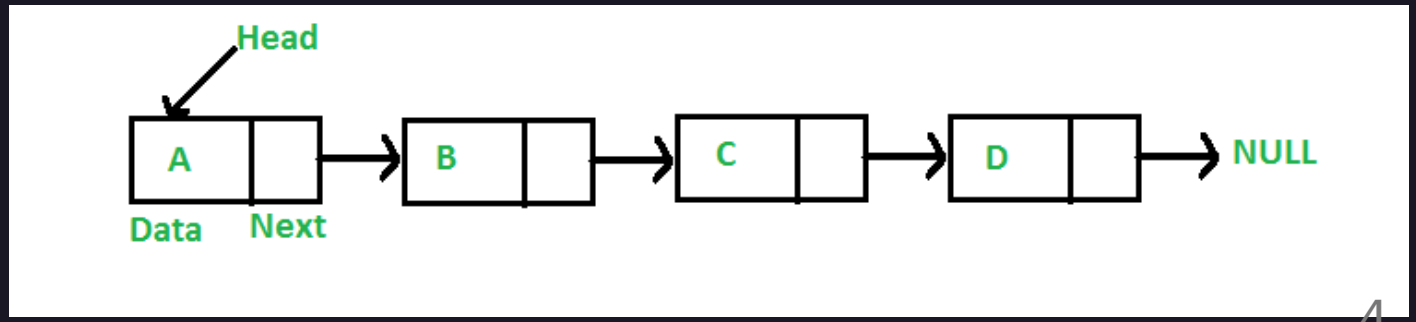
`&var` and `*var`

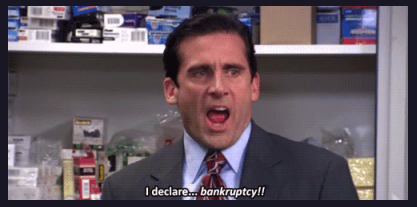
- `&` - The "address of" operator
  - `&var` - returns the address of var
- `*` - The dereference operator
  - When used on a pointer, returns the value at that memory address
  - This is "dereferencing" a pointer

Why pointers?



- **Get an *indirect* reference to a variable**
  - Have multiple variables point at the same location in memory
- **Point to a function that could be potentially swapped at run time**
- **The starting point to a world of data structures**
  - Linked lists, queues, etc.





Declaring pointers

```
int *ptr;

char* my_pointer;
```

The `*` is key!

- Can come before or after the space

Assigning pointers

```
int x = 42;

int* p = ???
```

How to assign `p` to the memory address of `x`?

Assigning pointers

```
int x = 42;

int* p = &x;
```

But, how can we change the value of `x` using `p`?

Assigning pointers

```
int x = 42;

int* p = &x;

*p = 100;
```

Printing pointers

Use `%p` for a pointer

Use the appropriate specifier for the value stored at a pointer

```
int x = 42;

int* p = &x;

printf("Pointer is: %p\n", p);
printf("Value is: %d\n", *p);
```

Pointer arithmetic

```
int x = 42;

int* p = &x;
*p += 1;
```

The above code increments the value stored at p(x)

Pointer arithmetic

```
int x = 42;

int* p = &x;
p += 1;
```

This code moves the pointer (`p`)

It moves based on the size of the type

- E.g., if `ints` are 4 bytes, `p` now points 4 bytes later in memory

Pointer arithmetic

Imagine I have four chars that happen to be stored sequentially in memory

```
char c1 = ‘G’;
char c2 = ‘V’;
char c3 = ‘S’;
char c4 = ‘U’;
```

G	V	S	U

Pointer arithmetic

```
char c1 = ‘G’;
char c2 = ‘V’;
char c3 = ‘S’;
char c4 = ‘U’;
char* p = &c1;
```

What is the value of `*(p+1)`?

- V

Some warnings

C doesn’t care

It will let you do all kinds of crazy stuff.

It is easy to do something other than what you intended

Pay attention to compiler warnings!

RECAP

What does the following code do?

```
int i = 100;   #0
int* p = &i;   #1

i = i + 5;     #2
*p += 6;       #3

p += 1;        #4
i--;           #5
*p = *p * 2;   #6

p--;           #7
*p = 0;        #8
```

```
int i = 100;   #0  i=100;
int* p = &i;   #1

i = i + 5;     #2
*p += 6;       #3

p += 1;        #4
i--;           #5
*p = *p * 2;   #6

p--;           #7
*p = 0;        #8
```

```
int i = 100;   #0  i=100;
int* p = &i;   #1  i=100; p->i; *p=100;

i = i + 5;     #2
*p += 6;       #3

p += 1;        #4
i--;           #5
*p = *p * 2;   #6

p--;           #7
*p = 0;        #8
```

```
int i = 100;   #0  i=100;
int* p = &i;   #1  i=100; p->i; *p=100;

i = i + 5;     #2  i=105; p->i; *p=105;
*p += 6;       #3

p += 1;        #4
i--;           #5
*p = *p * 2;   #6

p--;           #7
*p = 0;        #8
```

```
int i = 100;   #0  i=100;
int* p = &i;   #1  i=100; p->i; *p=100;

i = i + 5;     #2  i=105; p->i; *p=105;
*p += 6;       #3  i=111; p->i; *p=111

p += 1;        #4
i--;           #5
*p = *p * 2;   #6

p--;           #7
*p = 0;        #8
```

```
int i = 100;    #0  i=100;
int* p = &i;    #1  i=100; p->i; *p=100;

i = i + 5;      #2  i=105; p->i; *p=105;
*p += 6;        #3  i=111; p->i; *p=111

p += 1;         #4  i=111; p->&i+1;*p=?
i--;            #5
*p = *p * 2;    #6

p--;            #7
*p = 0;         #8
```

```
int i = 100;    #0  i=100;
int* p = &i;    #1  i=100; p->i; *p=100;

i = i + 5;      #2  i=105; p->i; *p=105;
*p += 6;        #3  i=111; p->i; *p=111

p += 1;         #4  i=111; p->&i+1; *p=?
i--;            #5  i=110; p->&i+1; *p=?
*p = *p * 2;    #6

p--;            #7
*p = 0;         #8
```

```
int i = 100;   #0  i=100;
int* p = &i;   #1  i=100; p->i; *p=100;

i = i + 5;     #2  i=105; p->i; *p=105;
*p += 6;       #3  i=111; p->i; *p=111

p += 1;        #4  i=111; p->&i+1; *p=?
i--;           #5  i=110; p->&i+1; *p=?
*p = *p * 2;   #6  i=110; p->&i+1; *p=?*2

p--;           #7
*p = 0;        #8
```

```
int i = 100;   #0  i=100;
int* p = &i;   #1  i=100; p->i; *p=100;

i = i + 5;     #2  i=105; p->i; *p=105;
*p += 6;       #3  i=111; p->i; *p=111

p += 1;        #4  i=111; p->&i+1; *p=?
i--;           #5  i=110; p->&i+1; *p=?
*p = *p * 2;   #6  i=110; p->&i+1; *p=?*2

p--;           #7  i=110; p->i; *p=110
*p = 0;        #8
```

```
int i = 100;    #0  i=100;
int* p = &i;    #1  i=100; p->i; *p=100;

i = i + 5;      #2  i=105; p->i; *p=105;
*p += 6;        #3  i=111; p->i; *p=111

p += 1;         #4  i=111; p->&i+1; *p=?;
i--;            #5  i=110; p->&i+1; *p=?;
*p = *p * 2;    #6  i=110; p->&i+1; *p=?*2;

p--;            #7  i=110; p->i; *p=110;
*p = 0;         #8  i=0  ; p->i; *p=0;
```

Making connections

Can we do this?

```
int arr[3] = {5,6,7};
*arr = 20;
*(arr + 2) = 100;
```

- YES!

Can we do this?

```
int arr[3] = {5,6,7};
arr++;
*arr = 90;
```

- **NO**! - Can't move `arr`

Making connections

Can we do this?

```
int arr[3] = {5,6,7};
int* p = arr;
p++;
*p = 1000;
```

- YES!

Making connections

Can we do this?

```
int arr[3] = {5,6,7};
int* p = arr;
p[2] = 555;
```

- YES!
  - `p[2]` is equivalent to `*(p+2)`

Applying

What does the following code do?

```
long arr[3] = {0,0,0};
long* p = arr;

p[0] = 10;
*p += 10;
*(p + 1) = 9;
p += 2;
*p = 1;
p--;
*p -= 2;
p[0]--;
```

Applying

What does the following code do?

```
long arr[3] = {0,0,0}; # { 0, 0, 0}
long* p = arr;

p[0] = 10;
*p += 10;
*(p + 1) = 9;
p += 2;
*p = 1;
p--;
*p -= 2;
p[0]--;
```

Applying

What does the following code do?

```
long arr[3] = {0,0,0}; # { 0, 0, 0}
long* p = arr;         # { 0, 0, 0} (p points at first)

p[0] = 10;
*p += 10;
*(p + 1) = 9;
p += 2;
*p = 1;
p--;
*p -= 2;
p[0]--;
```

Applying

What does the following code do?

```
long arr[3] = {0,0,0}; # { 0, 0, 0}
long* p = arr;         # { 0, 0, 0} (p points at first)

p[0] = 10;             # {10, 0, 0}
*p += 10;
*(p + 1) = 9;
p += 2;
*p = 1;
p--;
*p -= 2;
p[0]--;
```

Applying

What does the following code do?

```
long arr[3] = {0,0,0}; # { 0, 0, 0}
long* p = arr;         # { 0, 0, 0} (p points at first)

p[0] = 10;             # {10, 0, 0}
*p += 10;              # {20, 0, 0}
*(p + 1) = 9;
p += 2;
*p = 1;
p--;
*p -= 2;
p[0]--;
```

Applying

What does the following code do?

```
long arr[3] = {0,0,0}; # { 0, 0, 0}
long* p = arr;         # { 0, 0, 0} (p points at first)

p[0] = 10;             # {10, 0, 0}
*p += 10;              # {20, 0, 0}
*(p + 1) = 9;          # {20, 9, 0}
p += 2;
*p = 1;
p--;
*p -= 2;
p[0]--;
```

Applying

What does the following code do?

```
long arr[3] = {0,0,0}; # { 0, 0, 0}
long* p = arr;         # { 0, 0, 0} (p points at first)

p[0] = 10;             # {10, 0, 0}
*p += 10;              # {20, 0, 0}
*(p + 1) = 9;          # {20, 9, 0}
p += 2;                # {20, 9, 0} (p points at last)
*p = 1;
p--;
*p -= 2;
p[0]--;
```

Applying

What does the following code do?

```
long arr[3] = {0,0,0}; # { 0, 0, 0}
long* p = arr;         # { 0, 0, 0} (p points at first)

p[0] = 10;             # {10, 0, 0}
*p += 10;              # {20, 0, 0}
*(p + 1) = 9;          # {20, 9, 0}
p += 2;                # {20, 9, 0} (p points at last)
*p = 1;                # {20, 9, 1}
p--;
*p -= 2;
p[0]--;
```

Applying

What does the following code do?

```
long arr[3] = {0,0,0}; # { 0, 0, 0}
long* p = arr;         # { 0, 0, 0} (p points at first)

p[0] = 10;             # {10, 0, 0}
*p += 10;              # {20, 0, 0}
*(p + 1) = 9;          # {20, 9, 0}
p += 2;                # {20, 9, 0} (p points at last)
*p = 1;                # {20, 9, 1}
p--;                   # {20, 9, 1} (p points at middle)
*p -= 2;
p[0]--;
```

Applying

What does the following code do?

```
long arr[3] = {0,0,0}; # { 0, 0, 0}
long* p = arr;         # { 0, 0, 0} (p points at first)

p[0] = 10;             # {10, 0, 0}
*p += 10;              # {20, 0, 0}
*(p + 1) = 9;          # {20, 9, 0}
p += 2;                # {20, 9, 0} (p points at last)
*p = 1;                # {20, 9, 1}
p--;                   # {20, 9, 1} (p points at middle)
*p -= 2;               # {20, 7, 1}
p[0]--;
```

Applying

What does the following code do?

```
long arr[3] = {0,0,0}; # { 0, 0, 0}
long* p = arr;         # { 0, 0, 0} (p points at first)

p[0] = 10;             # {10, 0, 0}
*p += 10;              # {20, 0, 0}
*(p + 1) = 9;          # {20, 9, 0}
p += 2;                # {20, 9, 0} (p points at last)
*p = 1;                # {20, 9, 1}
p--;                   # {20, 9, 1} (p points at middle)
*p -= 2;               # {20, 7, 1}
p[0]--;                # {20, 6, 1}
```

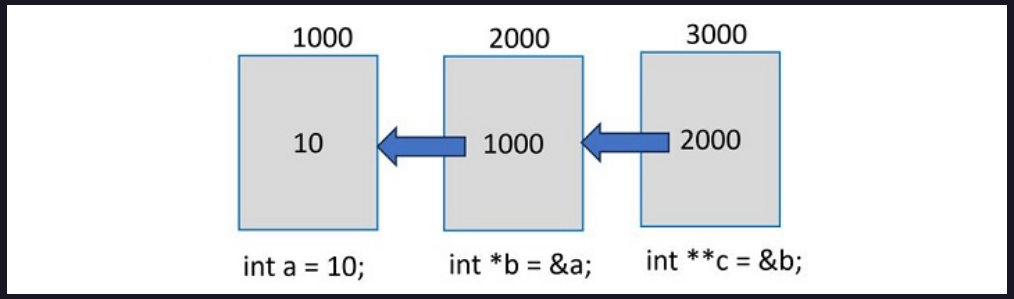
What does this do?

```
int x = 0;
int y = 1;

int* p = &x;
*p = 5;

p = &y;
*p = 6;
```

- At end, `x=5` and `y=6`

How about this?

```
int x = 0;
int* p = &x;
??? z = &p;
```



How about this?

```
int x = 0;
int* p = &x;
int** z = &p;
```

You can have pointer of pointers! Wee!

- More on this later