Pointers

Adapted from materials by Dr. Carrier

charlesdeluvio - Unsplash

Pointers

Pointers

A new data *type*

Pointers

A new data *type*

A pointer stores a memory address

Pointers

A new data *type*

A pointer stores a memory address
(i.e., it “points” to a location in memory)

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)

New operators

`&var` and `*var`

New operators

`&var` and `*var`

& - The “address of” operator

New operators

`&var` and `*var`

& - The “address of” operator

- &var returns the address of var

New operators

`&var` and `*var`

& - The “address of” operator

- &var returns the address of var

* - The dereference operator

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

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

Declaring pointers

```
int *ptr;
```

Declaring pointers

```
int *ptr;

char* my_pointer;
```

Declaring pointers

```
int *ptr;

char* my_pointer;
```

- The * is key!
- Can come before or after the space (I prefer before)

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;
```

Assigning values

```
int x = 42;

int* p = &x;
```

How can we change the value of x using p?

Assigning values

```
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;
```

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

```
int x = 42;

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

This code *moves the pointer (p)*

Pointer arithmetic

```
int x = 42;

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

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

```
int x = 42;

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

This code *moves the pointer (p)*

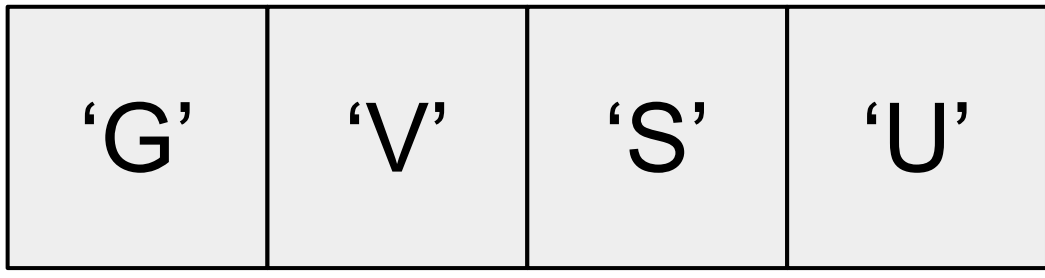
It moves based on the size of the type

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

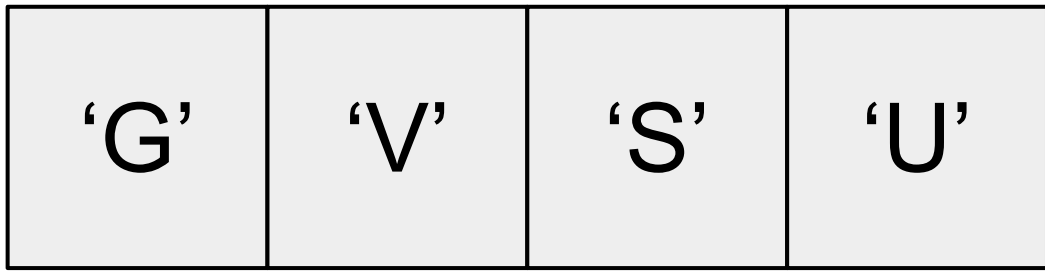
```
char c1 = ‘G’;
char c2 = ‘V’;
char c3 = ‘S’;
char c4 = ‘U’;
```



Pointer arithmetic

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

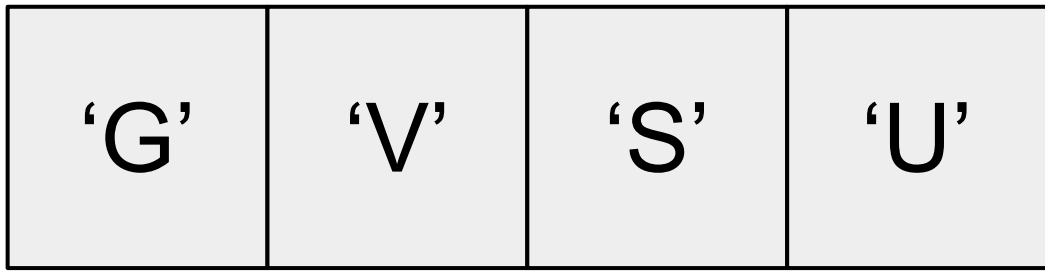
```
char c1 = ‘G’;
char c2 = ‘V’;
char c3 = ‘S’;
char c4 = ‘U’;
char* p = &c1;
```



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’;
char* p = &c1;
```

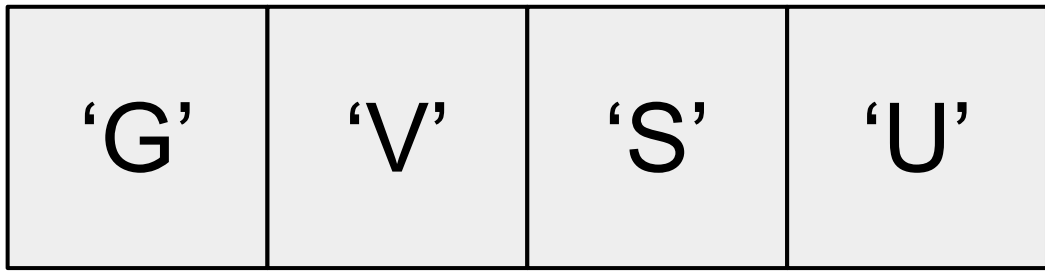


What is the value of *(p+1)?

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’;
char* p = &c1;
```



What is the value of *(p+1)?
‘V’!

A word of warning…

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! (Day two)

- What is a pointer?
- What are these new operators called, and what do they do: `*var` and `&var`
- How do you declare a pointer?
- How do you assign a pointer to a variable's address?
- How do you print a pointer?

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
```

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
```

i=100;

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
```

```
i=100;
i=100; p->i; *p=100
```

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
```

```
i=100;
i=100; p->i; *p=100

i=105; p->i; *p=105
```

Recap!

What does the following code do?

```
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
```

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
```

```
i=100;
i=100; p->i; *p=100

i=105; p->i; *p=105
i=111; p->i; *p=111

i=105; p->&i+1;*p=?
```

Recap!

What does the following code do?

```
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=105; p->&i+1;*p=?
i--;             #5  i=104; p->&i+1;*p=?
*p = *p * 2;     #6

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

Recap!

What does the following code do?

```
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=105; p->&i+1;*p=?
i--;             #5  i=104; p->&i+1;*p=?
*p = *p * 2;     #6  i=104; p->&i+1;*p=?*2

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

Recap!

What does the following code do?

```
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=105; p->&i+1;*p=?
i--;             #5  i=104; p->&i+1;*p=?
*p = *p * 2;     #6  i=104; p->&i+1;*p=?*2

p--;             #7  i=105; p->i; *p=105
*p = 0;          #8
```

Recap!

What does the following code do?

```
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=105; p->&i+1;*p=?
i--;             #5  i=104; p->&i+1;*p=?
*p = *p * 2;     #6  i=104; p->&i+1;*p=?*2

p--;             #7  i=105; p->i; *p=105
*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;
```

Making connections

Can we do this?

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

Making connections

Can we do this?

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

This?

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

Making connections

Can we do this?

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

This?

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

Cannot move arr

Making connections

Can we do this?

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

This?

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

Cannot move arr

This?

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

Making connections

Can we do this?

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

This?

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

Cannot move arr

This?

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

Making connections

Can we do this?

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

Making connections

Can we do this?

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

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

Applying what we know…

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 we know…

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]--;
```

```
{ 0, 0, 0}
{ 0, 0, 0}
```

Applying what we know…

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]--;
```

```
{ 0, 0, 0}
{ 0, 0, 0}

{10, 0, 0}
```

Applying what we know…

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]--;
```

```
{ 0, 0, 0}
{ 0, 0, 0}

{10, 0, 0}
{20, 0, 0}
```

Applying what we know…

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]--;
```

```
{ 0, 0, 0}
{ 0, 0, 0}

{10, 0, 0}
{20, 0, 0}
{20, 9, 0}
```

Applying what we know…

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]--;
```

```
{ 0, 0, 0}
{ 0, 0, 0}

{10, 0, 0}
{20, 0, 0}
{20, 9, 0}
{20, 9, 0}
```

Applying what we know…

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]--;
```

```
{ 0, 0, 0}
{ 0, 0, 0}

{10, 0, 0}
{20, 0, 0}
{20, 9, 0}
{20, 9, 0}
{20, 9, 1}
```

Applying what we know…

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]--;
```

```
{ 0, 0, 0}
{ 0, 0, 0}

{10, 0, 0}
{20, 0, 0}
{20, 9, 0}
{20, 9, 0}
{20, 9, 1}
{20, 9, 1}
```

Applying what we know…

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]--;
```

```
{ 0, 0, 0}
{ 0, 0, 0}

{10, 0, 0}
{20, 0, 0}
{20, 9, 0}
{20, 9, 0}
{20, 9, 1}
{20, 9, 1}
{20, 7, 1}
```

Applying what we know…

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]--;
```

```
{ 0, 0, 0}
{ 0, 0, 0}

{10, 0, 0}
{20, 0, 0}
{20, 9, 0}
{20, 9, 0}
{20, 9, 1}
{20, 9, 1}
{20, 7, 1}
{20, 6, 1}
```

Applying what we know…

What does the following code do?

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

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

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

Applying what we know…

What does the following code do?

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

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

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

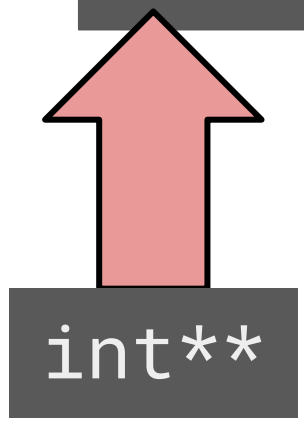
At the end, x=5 and y=6

What does this do?

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

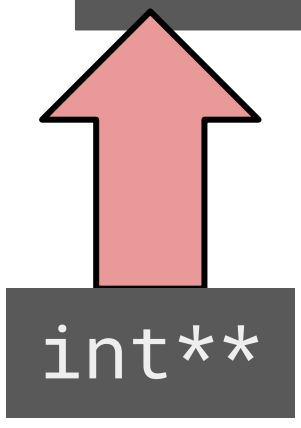
What does this do?

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



What does this do?

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



You can have pointers of pointers!

We'll talk about this more later! :D