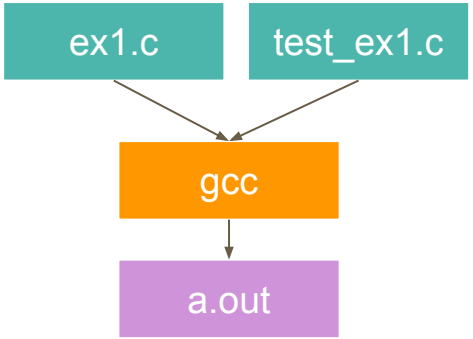
Lab 1

61C Spring 2023

Compiling a C Program

- `gcc` is used to compile C programs
- `gcc ex1.c test_ex1.c`



- Can specify the name of the executable file with the `-o` flag
  - `gcc -o ex1 ex1.c test_ex1.c`

Running a C Program

- To run an executable located in the current directory, use
  `./<executable_name>`
  - `./ex1`
- The dot refers to the current directory
- If you want to run a file in a different directory, specify the path after the dot
  - `./path/to/file/ex1`

Variable Types and Sizes

Guarantee: sizeof(long long) >= sizeof(long) >= sizeof(int) >= sizeof(short)

To know for sure what size your variable is, **use `uintN_t` or `intN_t` types.**

- `char` = 1 byte (8 bits)
- `short` = at least 2 bytes (16 bits), can be longer
- `int` = at least 2 bytes (16 bits), can be longer
  - `unsigned int`
- `float` = 4 bytes (32 bits)
- `double` = 8 bytes (64 bits)
- `long` = at least 4 bytes (32 bits), can be longer
- `long long` = at least 8 bytes (64 bits), can be longer

Defining a Function

Specify return type, function name, and function parameters.

```
int add(int x, int y) {      return x + y;     }

      void nothing() {      return;     }
```

Conditionals

If-else

```
if (condition) {
    do this;
} else if {
    do this;
} else {
    do this;
}
```

Switch statements

```
switch (expression) {
    case constant1:
        do these;
    case constant2:
        do these;
    default:
        do these;
}
break;
```

Loops

While loop

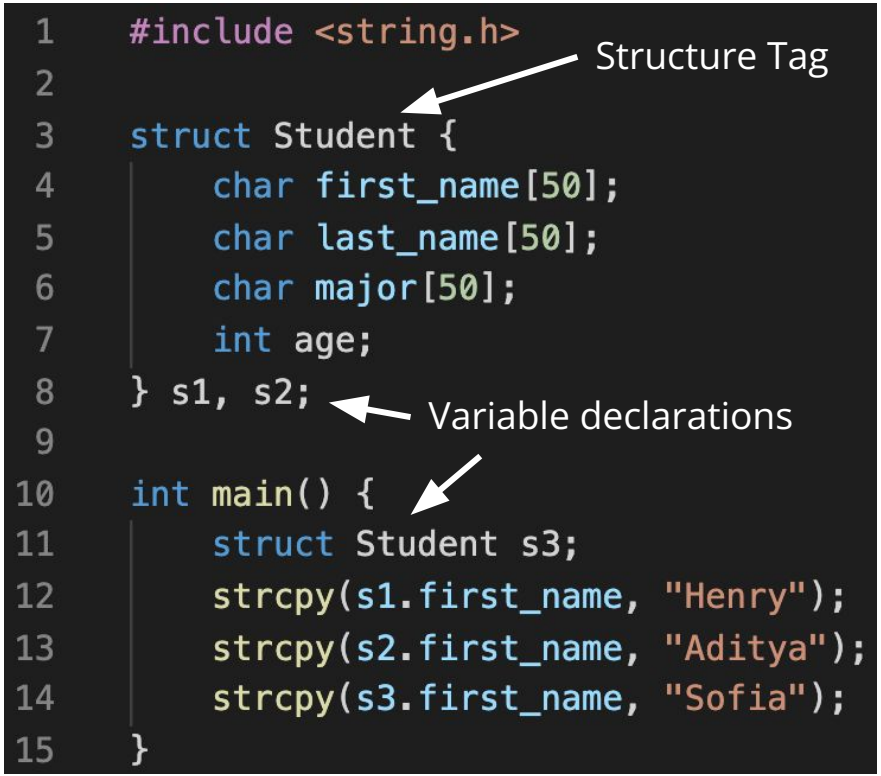
```
while (condition) {
    do this;
}
```

For loop

```
for (int i; i < 10; i++) {
    do this;
}
```

Structs

- What do they allow us to do?
- What is a structure tag?
- What are the two ways to declare struct variables?
- How do we access the members of a struct?



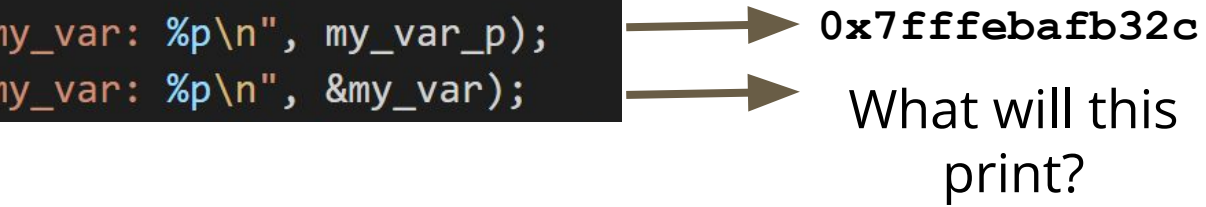
Structs

- typedef
  - Lets you avoid rewriting `struct` every time you want to declare a new struct variable
  - Can no longer declare variables in the struct definition

```c
#include <string.h>

typedef struct {
    char first_name[50];
    char last_name[50];
    char major[50];
    int age;
} Student;

int main() {
    Student s1, s2, s3;
    strcpy(s1.first_name, "Henry");
    strcpy(s2.first_name, "Aditya");
    strcpy(s3.first_name, "Sofia");
}
```

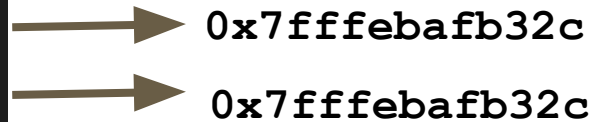
Pointers

```c
#include <stdio.h>

int main () {
    int my_var = 20;
    int* my_var_p;
    my_var_p = &my_var;
    printf("Address of my_var: %p\n", my_var_p);
    printf("Address of my_var: %p\n", &my_var);
```



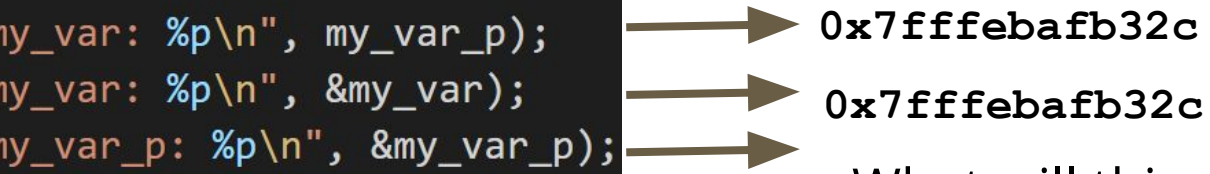
Pointers

```c
#include <stdio.h>

int main () {
    int my_var = 20;
    int* my_var_p;
    my_var_p = &my_var;
    printf("Address of my_var: %p\n", my_var_p);
    printf("Address of my_var: %p\n", &my_var);
```



Pointers

```c
#include <stdio.h>

int main () {
    int my_var = 20;
    int* my_var_p;
    my_var_p = &my_var;
    printf("Address of my_var: %p\n", my_var_p);
    printf("Address of my_var: %p\n", &my_var);
    printf("Address of my_var_p: %p\n", &my_var_p);
```



What will this print?

Pointers

```c
#include <stdio.h>

int main () {
    int my_var = 20;
    int* my_var_p;
    my_var_p = &my_var;
    printf("Address of my_var: %p\n", my_var_p);
    printf("Address of my_var: %p\n", &my_var);
    printf("Address of my_var_p: %p\n", &my_var_p);
```

→ **0x7fffebafb32c**

→ **0x7fffebafb32c**

→ Another address, ex:

0x7fffebafb320

Pointers

```c
#include <stdio.h>

int main () {
    int my_var = 20;
    int* my_var_p;
    my_var_p = &my_var;
    printf("Address of my_var: %p\n", my_var_p);
    printf("Address of my_var: %p\n", &my_var);
    printf("Address of my_var_p: %p\n", &my_var_p);
    *my_var_p += 2;
    printf("my_var: %d\n", my_var);
```

0x7fffebafb32c

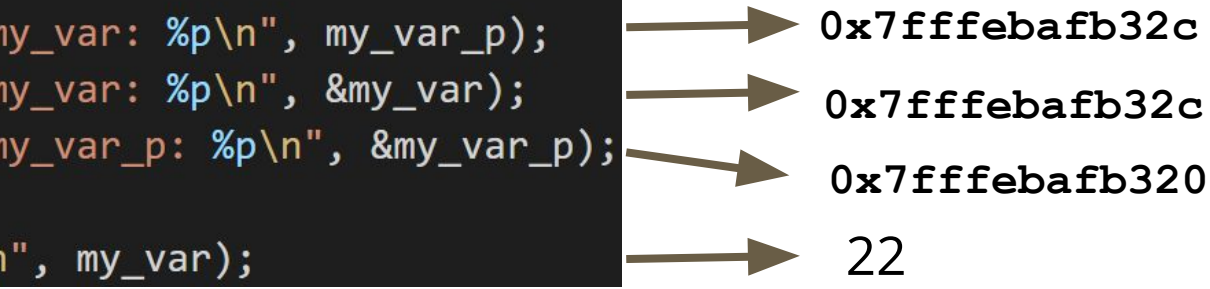
0x7fffebafb32c

0x7fffebafb320

What will this print?

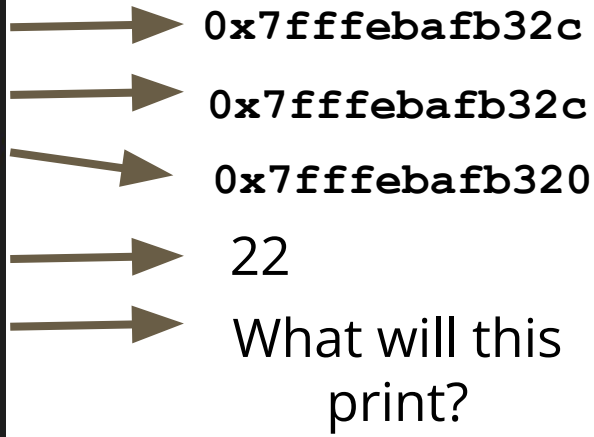
Pointers

```c
#include <stdio.h>

int main () {
    int my_var = 20;
    int* my_var_p;
    my_var_p = &my_var;
    printf("Address of my_var: %p\n", my_var_p);
    printf("Address of my_var: %p\n", &my_var);
    printf("Address of my_var_p: %p\n", &my_var_p);
    *my_var_p += 2;
    printf("my_var: %d\n", my_var);
```



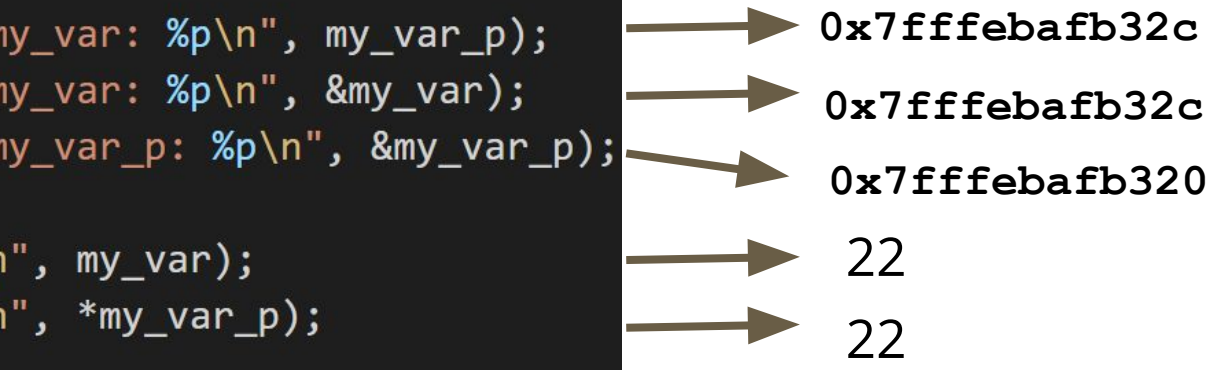
Pointers

```c
#include <stdio.h>

int main () {
    int my_var = 20;
    int* my_var_p;
    my_var_p = &my_var;
    printf("Address of my_var: %p\n", my_var_p);
    printf("Address of my_var: %p\n", &my_var);
    printf("Address of my_var_p: %p\n", &my_var_p);
    *my_var_p += 2;
    printf("my_var: %d\n", my_var);
    printf("my_var: %d\n", *my_var_p);
    return 0;
}
```



Pointers

```c
#include <stdio.h>

int main () {
    int my_var = 20;
    int* my_var_p;
    my_var_p = &my_var;
    printf("Address of my_var: %p\n", my_var_p);
    printf("Address of my_var: %p\n", &my_var);
    printf("Address of my_var_p: %p\n", &my_var_p);
    *my_var_p += 2;
    printf("my_var: %d\n", my_var);
    printf("my_var: %d\n", *my_var_p);
    return 0;
}
```



Pointers to Structs

- Pass a pointer of a struct to a function so that you can edit the contents
- Access the struct contents by:
  - (*student).major
  - student->major

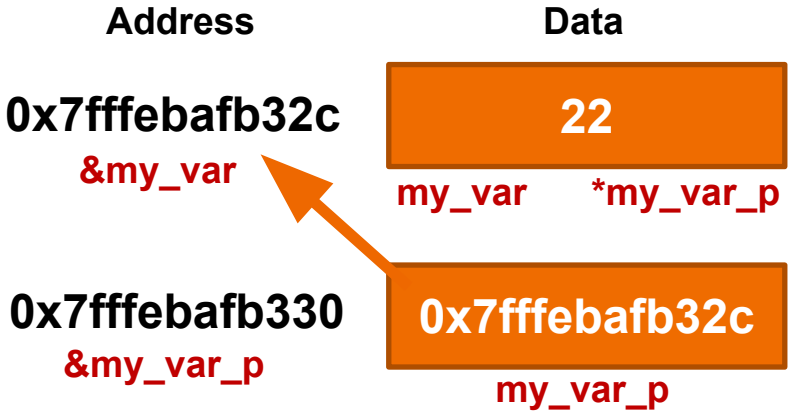
```
major: chemistry
major: biology
```

```c
#include <stdio.h>
#include <string.h>

typedef struct {
    char first_name[50];
    char last_name[50];
    char major[50];
    int age;
} Student;

void update_major (Student *student, char *new_major) {
    //strcpy((*student).major, new_major);
    strcpy(student->major, new_major);
}

int main() {
    Student s1;
    strcpy(s1.major, "chemistry");
    printf("major: %s\n", s1.major);
    update_major(&s1, "biology");
    printf("major: %s\n", s1.major);
}
```

Arrow operator

Pointers

&x = address of x
*x = contents at x




```c
#include <stdio.h>

int main () {
    int my_var = 20;
    int* my_var_p;
    my_var_p = &my_var;
    printf("Address of my_var: %p\n", my_var_p);
    printf("Address of my_var: %p\n", &my_var);
    printf("Address of my_var_p: %p\n", &my_var_p);
    *my_var_p += 2;
    printf("my_var: %d\n", my_var);
    printf("my_var: %d\n", *my_var_p);
    return 0;
}
```

```
Address of my_var: 0x7fffebafb32c
Address of my_var: 0x7fffebafb32c
Address of my_var_p: 0x7fffebafb330
my_var: 22
my_var: 22
```

Arrays

- A block of memory: size is **static**
  - `int arr[2];`
  - `int arr[] = {1, 2};`
- Accessing elements: array indexing
  - `arr[1]`
- An array variable is a “pointer” to the first element.
  - You can use pointers to access arrays!
  - `arr[0]` is the same as `*arr`
  - Can use pointer arithmetic to move the pointer
    - Each operation automatically moves the size of one whole “type” that ptr points to
    - `arr[1]` is the same as `*(arr + 1)`

Strings

- An array of **chars** representing individual characters
  - Ends in a null terminator '\0'
- strlen(char* string)
- strcpy(char* src, char* dest)

```
char str1[] = "hello";
char str2[6]; //dont forget to add space for the null terminator!
strcpy(str2, str1);
printf ("stri: %s\nstr2: %s\n", str1, str2); //str1: hello
                                             //str2: hello
return strlen(str2); //returns 5, not 6
```

Dynamic Memory

- Pointers can be allocated using malloc that persist in the heap
- Heap is a memory space separate from other variables declared in the stack
- Always remove dynamically allocated pointers by calling free(ptr); after use

```
char str1[] = "hello";
char * str2 = malloc(sizeof(char) * 6);
//malloc needs to know how many bytes to allocate
strcpy(str2, str1);

printf ("str1: %s\nstr2: %s\n", str1, str2); //str1: hello
                                             //str2: hello
free(str2);
return strlen(str2); //Crashes. Do not use a pointer after freeing it!
```