Create a Program in C (Last Class)

- Input:
  - three floating point numbers
- Output:
  - the average of those three numbers

- Use:
  - `scanf` to get the input
  - `printf` to show the result
  - a function to calculate the average

printf, scanf Syntax

```
printf(char *format, ...)

scanf(char *format, ...)
```

Format String

- %d – decimal integer
- %s – string
- %c – character
- %f – floating-point number

Example

```
int x;

scanf("Enter an integer: ", &i);
printf("The integer you entered is: ", i);
```

C/Java Syntax – Arrays and Strings

Lecture 03 Summary

- Arrays
- In-class Exercises
- Strings

By the end of this lecture, you will be able to write C code that uses and manipulates arrays and/or strings.

You will also be able to describe what happens in the computer's memory when this code is executed.

In Python, how would you write a function that takes the average of a set of numbers?

Example

```
def average(list):
    sum = 0.0
    size = 0
    for num in list:
        sum = sum + num
        size = size + 1

    return sum / size
```

- What would you pass into this function?

Example

```
avg = average([3,5,10,4,1,6])
```

In C, there are no lists, only arrays

Arrays vs. Lists

Arrays in C

- Have a fixed size that never changes
  - once full, will not grow
- All elements are of the same type (int, float, etc.)
- Has no insert or append operations
  - must write these yourself

Lists in Python

- Can add/remove elements at will
- Elements can be of different types
- Has special operations to insert, append, get the size, etc.

Array Declaration Syntax

```
<type> array_name[<# elements>];
```

Examples

```
int test_scores[10];
char student_name[50];
short avg_rainfall[31];
float observations[10000];
double temperatures[100];
unsigned int no_negatives[25];
long long big_numbers[3000];
```

Array Access Syntax

```
array_name[<element-index>]
```

Example

```
main()
{
    int test_scores[100];
    int i;

    /* initialize all array elements to 0 */
    for (i = 0; i< 100; i++)
        test_scores[i] = 0;

    /* Print out array elements */
    for (i = 0; i< 100; i++)
        printf("test_scores[%d] = %d\n", i, test_scores[i]);

    /* modify some array elements */
    test_scores[30] = 89;
    test_scores[25] = 37;
    test_scores[98] = 56;
    test_scores[33] = 21;

    /* print out array elements */
    for (i = 0; i< 100; i++)
        printf("test_scores[%d] = %d\n", i, test_scores[i]);
}
```

Array Initialization Syntax

```
<array-declaration> = { <element1>,
    <element2>, <element3>, ... };
```

Example

```
main ()
{
     int my_array[] = {50, 25, 31, 22, 16};
     int i;

     for (i = 0; i < 5; i++)
     {
          printf("my_array[%d] = %d\n", i, my_array[i]);
     }
}
```

Arrays as Function Parameters

```
void print_int_array(int array[])
```

- Write this function

Write a C function that computes the average of an array of numbers.

Exercises: Arrays in Memory

Exercise 1: Draw a Diagram

```
main()
{
    char array1[5];
    short array2[5];
    int array3[5];
    long array4[5];
    long long array5[5];
    float array6[5];
    double array7[5];

    unsigned char array8[5];
    unsigned short array9[5];
    unsigned int array10[5];
    unsigned long array11[5];
    unsigned long long array12[5];
}
```

Exercise 2

- Compute the size (in bytes) of each array.
- How did you compute the size?
- What information do you need to know to compute the amount of memory taken by an array?

Exercise 3

- Assume all of the arrays start at location 1000 (decimal).
- Compute the address (in memory) for each array element for each array.

Exercise 4

- What is the relationship between the index of an array element and its actual address? (express your answer in the form of an equation)

Exercise 5: Explain the output

```
main ()
{
  int x = 1000;
  int my_array[100];
  int y = 1000;
  int i;

  printf("x = %d\n", x);
  printf("y = %d\n\n", y);
  my_array[-1] = 5000;
  printf("x = %d\n", x);
  printf("y = %d\n", y);
}
```

Output:

```
x = 1000
y = 1000

x = 1000
y = 5000
```

Consider the following

```
void function(float[] array)
{
    ...
}
```

- What is the maximum size of the array?
- How many elements are in the array?
- What happens if you try to access an element outside the array's bounds?
- How would you insert an element in the middle?

```
array[89]
```

- When the compiler sees this bit of code, what does it do?

Strings

In C, there is no “string” type

What is a string made up of?

Example

```
char student_name[30];
```

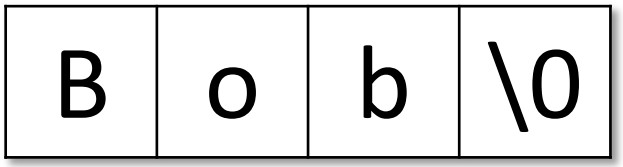
- Can I store the string "Wolfeschlegelsteinhausenbergerdorff" (35 chars)?
- Can I store the string "Matthew" (7 chars)?
- How does `printf` know to stop after the 'w'?

Null-terminated

- A string does not have to take up all of the allocated space.
- Must end with the null character:
  - '\0'
  - ascii value 0

Example

```
char name[] = "Bob";
```



What happens if you forget to end a string with ‘\0’?

C String Functions

Name	Syntax	Purpose
strcpy	`strcpy(char *s1, const char *s2)`	Copies the string pointed to by s2 into the character array pointed to by s1 (including the null terminator byte). s2 must be null-terminated and the programmer must ensure that the character array pointed to by s1 is large enough to accomodate the string in s2.
strncpy	`strncpy(char *s1, const char *s2, int n)`	Copies at most n characters from s2 into the character array s1. The null byte will be included in the copy.
strcat	`strcat(char *s1, const char *s2)`	appends string s2 to the end of character array s1. The first character of s2 overwrites the null character at the end of s1.

Name	Syntax	Purpose

strncat	`strncat(char *s1,`
	`  const char *s2,`
	`  int n)`
		appends at most n characters of the string s2 to the end of character array s1. The first character of s2 overwrites the null character at the end of s1.

strcmp	`int strcmp(const char *s1,` `  const char *s2)`	compares the string s1 to the string s2. If the strings are identical, the function returns 0. If s1 is*lexically less than* s2, then a number < 0 is returned. If s1 is *lexcially greater than* s2, then a value > 0 is returned

strncmp	`int strncmp(const char *s1,`
	`  const char *s2,`
	`  int n)`
		same as strcmp except than only up to n characters are compared.

strlen	`int strlen(const char *s)`	returns the number of characters in the string (not including the null character)

Example

```
strcat(char *s1, const char *s2)
{
    int i = 0;
    int j = 0;

    /* Find the end of the first string */
    while (s1[i] != '\0')
    {
        i++;
    }

    /* Starting there, add the contents of the second */
    while (s2[j] != '\0')
    {
        s1[i] = s2[j];

        i++;
        j++;
    }

    /* Make sure the string is null-terminated */
    s1[i] = '\0';
}
```

Exercise

- Create a function called ninjify that adds an extra space character in between each word.
  - "`Is there a ninja in my program?`" would become: "`Is  there  a  ninja  in  my  program?`"
- The function signature should be:

```
void ninjify(char *s1, const char *s2)
```

Lecture 03 Summary

- Arrays
- In-class Exercises
- Strings

Next Class

- Pointers and Indirection