Programming in C
Today, you’ll begin your very first program in C!

- You already know how to program in C++! 🎉
- Programming in C is a simplification of the C++ programming.

1. Program Starting Point of **ALL** C PROGRAMS:

2. Printing Using **printf()** (and include `<stdio.h>`):

```c
03/printf.c

int main() {
    int i = 42;
    char *s = "Hello, world!";
    float f = 3.14;
    printf("%d  %s  %f
", i, s, f);  // First argument
    printf("%d
", s[0]);  // Additional arguments
    printf("%d
", s);
    printf("%d
", f);
    return 0;
}
```

3. Pointers:

4. Heap Memory Allocation:

```c
03/malloc.c

typedef struct _myObject {
    int value;
    char *s;
} myObject;

int main() {
    char *s = malloc(10);
    myObject *obj = malloc(sizeof(myObject));
    obj->value = 3;
    printf("%s %s %s
", s, obj, obj->value);
    return 0;
}
```

5. **Strings** – There is no “data type” in C known as a string. Instead, we refer to “C Strings” as a sequence of characters:
- A “C string” is just a character pointer: ________.
- The string continues until it reaches a ________ byte.
- C will automatically include the NULL byte **ONLY** when using double quotes in your code (**not** counted as part of the length, but **does require memory – extremely tricky!**)

```c
03/string.c

char *s = malloc(6);
strcpy(s, "cs340");
printf("s[0]: 0x%x == %d == %c
", s[0], s[0], s[0]);
printf("s[4]: 0x%x == %d == %c
", s[4], s[4], s[4]);
printf("s[5]: 0x%x == %d == %c
", s[5], s[5], s[5]);
printf("s == "%s", strlen(s): %ld
", s, strlen(s));
```

```c
03/utf8.c

char *s = malloc(5);
s[0]=0xF0; s[1]=0x9F; s[2]=0x8E; s[3]=0x89; s[4]=0x00;
char *s1 = "\xF0\x9F\x8E\x89";
char *s2 = "🎉";
char *s3 = "\U0001f389";  // \U - must be 8 bytes
printf("%s %s %s %s
", s, s1, s2, s3);
printf("strlen(): %ld %ld %ld %ld
", strlen(s), strlen(s1), strlen(s2), strlen(s3));
```

Some extremely useful built in string functions:
- `strcmp(char *s1, char *s2)` – Compares two strings
- `strcat(char *dest, char *src)` – Concatenate two strings
- `strcpy(char *dest, char *src)` – Copies a string
- `strlen(char *s)` – Returns the length of the string
Logic Gates and Truth Tables
We can begin to define the building blocks of the CPU by basic instructions with input bits and output bits through logical gates.
  - By convention, you will see that the input bits are labeled A and B by default.

Logic Gate #1:

Logic Gate #2:

Logic Gate #3:

Logic Gate Challenge: \( A \ XOR \ B \)

We can also express this in a table known as a truth table:

<table>
<thead>
<tr>
<th>Op.</th>
<th>Binary</th>
<th>Math</th>
<th>Example Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( x )</td>
<td></td>
<td>1100 110011 101</td>
</tr>
<tr>
<td>( B )</td>
<td>( y )</td>
<td></td>
<td>1010 11 010</td>
</tr>
<tr>
<td>AND</td>
<td>( A \land B )</td>
<td>( xy )</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>( A \lor B )</td>
<td>( x + y )</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>( A \oplus B )</td>
<td>( x \ XOR \ y )</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>( \neg A )</td>
<td>( x' )</td>
<td></td>
</tr>
</tbody>
</table>

Truth Table: Half Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>( A + B )</th>
<th>SUM</th>
<th>CARRY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Truth Table for a Half Adder

Circuit Diagram for a “Half Adder”:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>CARRY\text{\small{in}}</th>
<th>SUM</th>
<th>CARRY\text{\small{out}}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Truth Table for a Full Adder

Circuit Diagram for a “Full Adder”:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>CARRY\text{\small{in}}</th>
<th>SUM</th>
<th>CARRY\text{\small{out}}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Truth Table for a Full Adder

Chaining Circuits Together: ________________________________

Disadvantages: