Lab 1: C

CS 105
Variables

- Declaration
  ```c
  int myVariable;
  ```

- Assignment
  ```c
  myVariable = 47;
  ```

- Declaration and assignment
  ```c
  int myVariable = 47;
  ```

<table>
<thead>
<tr>
<th>C Data Type</th>
<th>x86-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4</td>
</tr>
<tr>
<td>unsigned long</td>
<td>8</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
</tbody>
</table>
Operations

• Bitwise Operations: &, |, ^, ~
  ```
  int x = 47;
  int y = ~x;
  y = x & y;
  ```

• Logical Operations: &&, ||, !
  ```
  int x = 47;
  int y = !x;
  y = x && y;
  ```

• Arithmetic Operations: +, -, *, /, %
  ```
  int x = 47;
  int y = x + 13;
  y = (x * y) % 5;
  ```

• Boolean Operators: ==, !=, >, >=, <, <=
  ```
  int x = (13 == 47);
  ```
Control Flow

Conditionals

int x = 13;
int y;
if (x == 47){
y = 1;
} else {
y = 0;
}

While Loops

int x = 47;
while (x > 0){
x = x - 1;
}

Do-While Loops

int x = 47;
do {
x = x - 1;
} while (x > 0);

For Loops

int x = 0;
for (int i=0; i < 47; i++){
x = x + i;
}
Functions

Declaring a Function

```c
int myFunction(int x, int y){
    int z = x - 2*y;
    return z * x;
}
```

Calling a Function

```c
int a;

a = myFunction(47, 13);
```
Exercise 1

• Create a file called part1.c (don’t worry about running it yet)

• In that file, define a function that takes two integers and returns an integer.

• If the second integer argument is greater than (or equal to) the first, it returns the sum of the integer values between those two numbers (inclusive).

• Otherwise, it returns -1.
Hello World

```c
#include<stdio.h>

int main(int argc, char** argv){
    printf("Hello world!\n");
    return 0;
}
```
Aside: Printing

```c
printf("Hello world!\n");

printf("%d is a number\n", 13);

printf("%d is a number greater than %f\n", 47, 3.14);
```
Compilation

compiler output name filename
- gcc -o hello hello.c

Source program (text)
- hello.c
  - Pre-processor (cpp)
    - hello.i
      - Modified source program (text)
      - Compiler (cc1)
        - hello.s
          - Assembly program (text)
          - Assembler (as)
            - hello.o
              - Relocatable object programs (binary)
              - Linker (ld)
                - hello
                  - Executable object program (binary)

#include<stdio.h>

int main(int argc,
          char ** argv){
  printf("Hello world!\n");
  return 0;
}

... int printf(const char * restrict, ...
    _attribute__((__format__ (__printf__, 1, 2)));
    ...
    int main(int argc,
              char ** argv){
      printf("Hello world!\n");
      return 0;
    }

pushq  %rbp
movq $32, %rsp
lea L_str(%rip), %rax
movl $0, -4(%rbp)
movl %edi, -8(%rbp)
movq %rax, %rdi
movb $0, %al
callq _printf
xorl %ecx, %ecx
movl %eax, -20(%rbp)
movl %ecx, %eax
addq $32, %rsp
popq %rbp
retq

printf.o

55 48 89 e5
48 83 ec 20
48 8d 05 25 00 00 00
c7 45 fc 00 00 00 00
89 7d f8
48 89 75 f0
48 89 c7
b0 00
e8 00 00 00 00
31 c9
89 45 ec
89 c8
48 83 c4 20
5d
c3
Running a Program

- hello.c demo
Exercise 1b

- Add a main function to your part1.c file that calls your function with some hardcoded arguments and prints the value it returns. Then compile and run your program.
Arrays

• Contiguous block of memory
• Random access by index
  • Indices start at zero

• Declaring an array:

```plaintext
int array1[5]; // array of 5 ints named array1
char array2[47]; // array of 47 chars named array2
int array3[7][4]; // two dimensional array
```

• Accessing an array:

```plaintext
int x = array1[0];
```

• The array variable stores the address of the first element in the array
Pointers

- Pointers are addresses in memory (i.e., indexes into the array of bytes)

- Most pointers declare how to interpret the value at (or starting at) that address

Examples:

```c
int * ptr = &myVariable;
char * ptr2 = (char *) ptr;
```

Dereferencing pointers:

```c
int var2 = *ptr
char c = *ptr2;
```

& and * are inverses of one another
Pointer Arithmetic

- Location of `ptr+k` depends on the type of `ptr`
- adding 1 to a pointer `p` adds `1*sizeof(*p)` to the address

- `array[k]` is the same as `*(array+k)`

```c
int * ptr = &myVariable;
ptr += 1;

char * ptr2 = (char *) ptr;
ptr2 += 1;
```
Strings

- Strings are just arrays of characters
- End of string is denoted by null byte `\0`

- generally declared as type `char *`
Aside: Main Function Parameters

```c
#include<stdio.h>

int main(int argc, char** argv){
    printf("Hello world!\n");
    return 0;
}
```
Exercise 2

• Open a file called part2.c. In that file, write a program that computes the pair of unsigned integers x, y such that the array [x, y] has the same binary representation as the string “CS 105!”

Hint: you can do this entirely with pointer arithmetic and casts. You don’t need to compute it by hand using the ASCII table (although you can!)
**Structs**

- Heterogeneous records, like objects
- Typical linked list declaration:
  ```
  typedef struct cell {
    int value;
    struct cell *next;
  } cell_t;
  ```

- Usage:
  ```
  cell_t c;
  c.value = 42;
  c.next = NULL;
  ```

- Usage with pointers:
  ```
  cell_t *p;
  p->value = 42;
  p->next = NULL;
  ```

  `p->next` is an abbreviation for `(*p).next`
Exercise 3

- Implement a linked list