McGill

C Lab #2: C Basics

COMP 310 / ECSE 427 : Winter 2023

TA: Murray Kornelsen

Contents

- Hello World
- Data Types
- Variables
- Functions
- Header Files
- Arrays and Pointers
- Strings
- Structs

C Hello World

```c
#include <stdio.h>

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

```
murray@DESKTOP-LFSQ6CL$ gcc HelloWorld.c
murray@DESKTOP-LFSQ6CL$ ./a.out
Hello World!
```

- "main" is the function that is called when your program is launched.
- Can take command line input with argc & argv.
- Can also be defined as "int main(void) {…}"

Fundamental Data Types

- Integer Types
- Floating Point

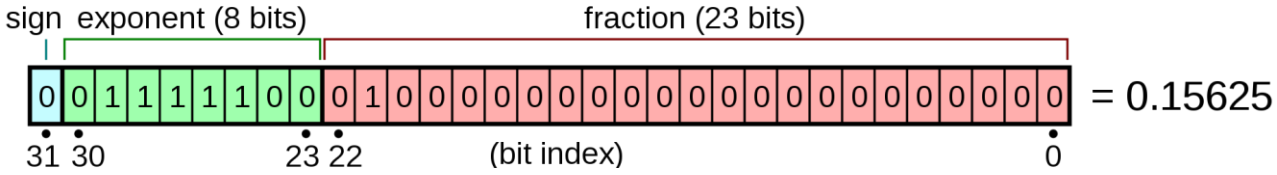
Integer Types

Type	Size	Range
char unsigned char	1 byte (8 bits)	-128 to 127 0 to 255
short unsigned short	2 bytes	-32768 to 32767 0 to 65535
int unsigned int	4 bytes	-2^{31} to $2^{31}-1$ 0 to $2^{32}-1$
long unsigned long	8 bytes	-2^{63} to $2^{63}-1$ 0 to $2^{64}-1$

- Note that types can differ between compilers.
  - Example: MSVC defines long as 4 bytes and “long long” as 8 bytes.
- This table applies for 64-bit gcc.

Float Types

Type	Size
float	4 bytes
double	8 bytes



- More complex type which stores numbers as a sign, exponent, and fraction.
- Probably not needed for this course.

Variables

- Syntax
  - <data type> <name>;
    - Value undefined.
  - <data type> <name> = <value>;

```c
#include <stdio.h>

int main(void) {

    int a;
    short b = 5;
    char c = 'x';
    float f = 3.5f;

    printf(
        "a = %d\n"
        "b = %d\n"
        "c = %c\n"
        "f = %f\n"
        , a, b, c, f);
}
```

```
murray@DESKTOP-LFSQ6CL$ gcc Variables.c
murray@DESKTOP-LFSQ6CL$ ./a.out
a = 0
b = 5
c = x
f = 3.500000
```

Functions

- Syntax
  - <return type> <name> (<parameters>);
  - <return type> <name> (<parameters>) {
    <statements>
    return <retval>;
    }
  - Parameters defined as <datatype> <name>

- First syntax can be used to “forward declare” a function.
- Second syntax defines behavior.

Exercise 1

- Write and call a function that adds two integers and returns the result.
- Try placing your function both above and below “main”.

- To print a number, use printf(“%d\n”, x);

Exercise 1

- Write and call a function that adds two integers and returns the result.
- Try placing your function both above and below “main”.

```c
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}

int main(void) {
    int sum = add(12, 45);
    printf("%d\n", sum);
    return 0;
}
```

```
murray@DESKTOP-LFSQ6CL$ gcc Exercise1.c
murray@DESKTOP-LFSQ6CL$ ./a.out
57
```

Exercise 1

- Write and call a function that adds two integers and returns the result.
- Try placing your function both above and below “main”.

```c
#include <stdio.h>

int add(int a, int b);

int main(void) {
    int sum = add(12, 45);
    printf("%d\n", sum);
    return 0;
}

int add(int a, int b) {
    return a + b;
}
```

```
murray@DESKTOP-LFSQ6CL$ gcc Exercise1.c
murray@DESKTOP-LFSQ6CL$ ./a.out
57
```

Separate C Files

- To organize your code, you may want to put functions in different files.
- To call a function in a different file, you need a forward declaration.
  - Just tells the compiler “this function exists somewhere”.
  - Compiler works top to bottom.

File1.c

```c
void greeting();

int main(void) {
    greeting();
    return 0;
}
```

File2.c

```c
#include <stdio.h>

void greeting() {
    printf("Hello from File2\n");
}
```

```
murray@DESKTOP-LFSQ6CL$ gcc File1.c File2.c
murray@DESKTOP-LFSQ6CL$ ./a.out
Hello from File2
```

Header Files

- Placing forward declarations in every C file will not scale well.
- Header files hold function declarations (and more), for convenient use with #include.

Funcs.h

```
#ifndef FUNCS_H
#define FUNCS_H

void foo();
void bar();

#endif
```

Include Guard

Funcs.c

```
#include <stdio.h>
#include "Funcs.h"

void foo() {
    printf("Hello from foo!\n");
}

void bar() {
    printf("Hello from bar!\n");
}
```

Main.c

```
#include "Funcs.h"

int main(void) {
    foo();
    bar();
    return 0;
}
```

```
murray@DESKTOP-LFSQ6CL$ gcc Main.c Funcs.c
murray@DESKTOP-LFSQ6CL$ ./a.out
Hello from foo!
Hello from bar!
```

Types of .h files

- System headers
  - #include <stdio.h>
  - #include <stdlib.h>
  - #include <string.h>
- User made headers
  - #include “Funcs.h”

- <…> used when including system headers.
- “…” searches local directories first and should be used with your headers.

Arrays

- You can also have arrays for every data type.
- Syntax
  - <data type> <name> [<number of elements>];
  - <data type> <name> [<numel>] = { <initializers> };
- Access with <name> [<idx>]

```c
#include <stdio.h>

int main(void) {
    int arr[5] = { 1, 2, 3, 4, 5 };
    for (int i = 0; i < 5; i++) {
        printf("%d\n", arr[i]);
    }
    return 0;
}
```

```
murray@DESKTOP-LFSQ6CL$ gcc Arrays.c
murray@DESKTOP-LFSQ6CL$ ./a.out
1
2
3
4
5
```

Pointers

- A pointer is just a 64-bit unsigned integer that represents a memory address.
- Can store the address of a large block of memory (malloc).
- You can define pointers for any type.
  - Including pointers to pointers.
  - Another C lab will go in depth later.
  - “*” is used to declare and dereference pointers.
  - “&” is used to get the address of a variable.
- Syntax:
  - <datatype> * <name> = &<variable>

```c
#include <stdio.h>

int main(void) {
    int a = 5;
    printf("a = %d\n", a);

    int *b = &a;
    *b = 10;
    printf("a = %d\n", a);
}
```

```
murray@DESKTOP-LFSQ6CL$ gcc Pointer.c
murray@DESKTOP-LFSQ6CL$ ./a.out
a = 5
a = 10
```

Arrays and Pointers

- In C, pointers and arrays are interchangeable in many ways.
  - One important difference:
    - sizeof(ptr) = 8
    - sizeof(array) = sizeof(dtype) * length
  - When you pass an array to a function, it will be handled as a pointer.
- You can index a pointer in the same way as an array using [<idx>].

String Creation

- C strings are just arrays of char terminated with the **null** character '\0'.
- Initialization
  - Array: char str[] = "Hello";
  - Pointer: char *str = "Hello";
  - Specified size array: char str[100] = "Hello";

- Important: if you use the pointer method, you cannot modify any characters in the string.
  - This creates a pointer into read-only memory.
  - Can use malloc or strdup to get a modifiable string.

String Functions

- The standard library contains various useful string functions.
- #include <string.h>

Function	Description
int strlen(char *str)	Returns the length of a string.
int strcmp(char *a, char *b)	Returns 0 if strings are identical. Otherwise returns some “difference”.
char *strcat(char *dst, char *src)	Appends string src to dst.
char *strcpy(char *dst, char *src)	Copy src into dst buffer.
char *strdup(char *src)	Allocates memory and copies src to it. Need to free the returned pointer.

String Examples

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

int main(void) {
    char str1[] = "This is a string";

    int len = strlen(str1);
    printf("str1 length = %d\n", len);

    char *str2 = strdup(str1);
    int cmp = strcmp(str1, str2);
    printf("strcmp(str1, str2) = %d\n", cmp);

    char str3[128] = "Hello ";
    strcat(str3, str2);
    printf("%s\n", str3);
}
```

```
murray@DESKTOP-LFSQ6CL$ gcc Strings.c
murray@DESKTOP-LFSQ6CL$ ./a.out
str1 length = 16
strcmp(str1, str2) = 0
Hello This is a string
```

Exercise 2

- Try using some string functions.
  - #include <string.h>
  - Declare a string as char [].
  - Declare a string as char *.
    - Try printing sizeof(str) for each of these.
  - Print the result of strlen on one of your strings.
  - Concatenate strings with strcat and print the result.
    - Create a large array for the dst parameter.

Function	Description
int strlen(char *str)	Returns the length of a string.
int strcmp(char *a, char *b)	Returns 0 if strings are identical. Otherwise returns some “difference”.
char *strcat(char *dst, char *src)	Appends string src to dst.
char *strdup(char *src)	Allocates memory and copies src to it. Need to free the returned pointer.

Exercise 2

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

int main(void) {
    char a[128] = "This is string a";
    char *b = "This is string b";

    printf("sizeof(a) = %ld\n", sizeof(a));
    printf("sizeof(b) = %ld\n", sizeof(b));

    printf("strlen(a) = %ld\n", strlen(a));

    strcat(a, b);
    printf("strcat(a, b) = %s\n", a);

    return 0;
}
```

```
murray@DESKTOP-LFSQ6CL$ gcc Exercise2.c
murray@DESKTOP-LFSQ6CL$ ./a.out
sizeof(a) = 128
sizeof(b) = 8
strlen(a) = 16
strcat(a, b) = This is string aThis is string b
```

Structs

- Structs can be used to create groups of data.
  - Simpler than “objects”.
- Values can then be accessed by name.

```c
struct HelloStruct {
    int a;
    float b;
    char c[16];
    char *d;
};
```

```c
struct HelloStruct hs;
hs.a = 10;
strcpy(hs.c, "Hello");

printf("hs.a = %d\n", hs.a);
printf("hs.c = %s\n", hs.c);
```

```c
struct HelloStruct hs2 = {
    .a = 10,
    .c = "Hello"
};
```

```
murray@DESKTOP-LFSQ6CL$ gcc Structs.c
murray@DESKTOP-LFSQ6CL$ ./a.out
hs.a = 10
hs.c = Hello
```

Exercise 3

- Create a struct that contains four different data types.
  - Idea: struct Animal containing name, weight, …
- Initialize the struct with some data.
- Print the data with printf.
  - %d – integer
  - %f – float
  - %s – string

Exercise 3

```c
#include <stdio.h>

struct Animal {
    char *name;
    short height;
    float weight;
    int population;
};

int main(void) {

    struct Animal animal = {
        "Dog",
        40,
        15.0f,
        3000000
    };

    printf("name = %s\n", animal.name);
    printf("height = %d cm\n", animal.height);
    printf("weight = %f lbs\n", animal.weight);
    printf("population = %d\n", animal.population);
    return 0;
}
```

```
murray@DESKTOP-LFSQ6CL$ gcc Exercise3.c
murray@DESKTOP-LFSQ6CL$ ./a.out
name = Dog
height = 40 cm
weight = 15.000000 lbs
population = 3000000
```

Exercise 4

- Implement a simple sorting algorithm.
  - I suggest selection sort or bubble sort.

- Declare an array of length N.
  - int arr[10] = { <val1>, <val2>, … }

- for loop syntax:
  - for (int i = 0; i < 10; i++) {
    <statements>
    }

- Extra challenges:
  - Instead of just implementing this in main, write a function and pass your array as a pointer.
  - Separate your code into two files: sort.c and main.c
  - #include <stdlib.h> and use rand() to fill your array.

Solution 1 – Selection Sort

```c
#include <stdio.h>

int main(void) {
    int arr[10] = {5, 2, 9, 1, 0, 3, 2, 8, 9, 7};

    for (int i = 0; i < 10; i++) {
        int minidx = i;
        for (int j = i; j < 10; j++) {
            if (arr[j] < arr[minidx]) {
                minidx = j;
            }
        }
        int tmp = arr[minidx];
        arr[minidx] = arr[i];
        arr[i] = tmp;
    }

    for (int i = 0; i < 10; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

Solution 2 – Selection Sort Function

```c
#include <stdio.h>

void sort(int *arr, int len) {
    for (int i = 0; i < len; i++) {
        int minidx = i;
        for (int j = i; j < len; j++) {
            if (arr[j] < arr[minidx]) {
                minidx = j;
            }
        }
        int tmp = arr[minidx];
        arr[minidx] = arr[i];
        arr[i] = tmp;
    }
}

int main(void) {
    int arr[10] = {5, 2, 9, 1, 0, 3, 2, 8, 9, 7};

    sort(arr, 10);

    for (int i = 0; i < 10; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```