COMP310/ECSE427 Lab2
C Basic

Jason Zixu Zhou

Slides are revised from Murray Kornelsen

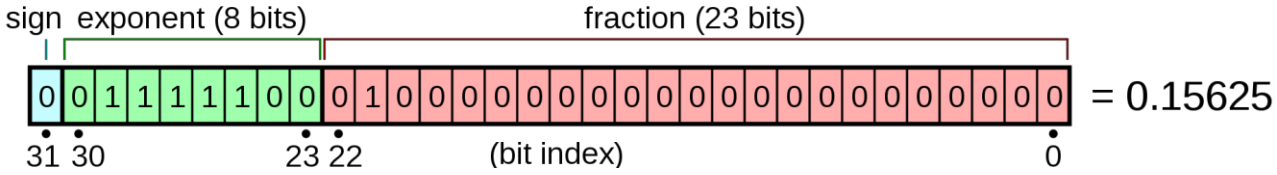
Integer Types

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
```

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.

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.

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
```