

CMPT 125

**Introduction to Computing Science
and Programming II**

September 29, 2021

Assignment 1

- Your code needs to compile with the provided makefile
 >> make
 >> .test1
- Currently makefile compiles your code using
 gcc -Wall -std=c99 -o test1 test1.c **assignment1.c**
- You need to submit exactly one file **assignment1.c**
- If you submit a file with a different name, it will not compile → you get zero
- If you include main(), it will not compile → you get zero
- Don't use printf("enter number") scanf ("...").
 The grader will not enter anything → you get zero
- No printf! No scanf!

Execution stack and scope of variables

Execution stack and scope of variables

An execution stack (call stack, run-time stack...) is a data structure that stores the information about the functions called during the execution of a program.

For each function the stack stores the following information:

- parameters of the function


- local variables

- return value


- return address: When a function completes,
it returns control to the function that called it.

Execution stack and scope of variables


```
□ int bar(int size) {  
    int i = size+1;  
    return i;  
}
```



```
int foo(int n) {  
    int ret = 3;  
    bar(4);  
    bar(8);  
    return ret;  
}
```



```
int main() {  
    foo(5);  
    return 0;  
}
```



bar()
params: int size = 8;
variables: int size = 8; int i = 9;
return value: ?
return address: ...

foo()
params: int n = 5;
variables: int n = 5; ret = 3;
return value:
return address: 0x9378ad

main()
params: --
variables: ...
return value: ?
return address: 0x128fbad

Execution stack and scope of variables

All local variables of a function are stored in the corresponding stack-frame. When the function completes, the variables become unavailable.

Variables obtained from `malloc()` are stored in "global memory", and do not die when the function completes.

We must free the memory when we don't need them anymore.

Terminology:

Local variables are stored on the *stack*.

Global variables are stored on the *heap*.

Stack (data structure)

Stack is a data structure with two principal operations:

Push(element) – adds an element to the collection

Pop() - removes the most recently added element from the collection

Example: Stack of blocks

we add and remove only to/from the top.

Same with function calls:

Each function call adds a block to the stack.

When finished, we remove it from the stack and continue to the next function on top.



Functions and structs

Functions and Structs

Note that struct is treated similarly to any other type (int, char)

Example:

Consider the function

```
person get_person() {  
    person p;  
    ...  
    return p;  
}
```

It returns the struct by copying p.data1, p.data2... to the caller.

More examples in struct_ret.c and struct_ret_ptr.c

Pointers to Functions

Function Pointers

In C functions are stored in the memory just as any other data.

And we can use pointers to refer to them. [A crazy idea, but why not]

The syntax is:

```
int foo(char c, int n, char* s) {...}
```

```
int (*my_function_ptr)(char, int, char*);
```

```
my_function_ptr = foo;
```

```
my_function_ptr('A', 10, str);
```

See `function_pointers.c`

We did not cover...

What we did not cover

Union – similar to struct except that all data fields share the same memory. For example:

```
union op {  
    char op_c;  
    int op_a;  
    int op_b[3];  
};
```

will have the size of 3 ints. If you modify op_a it may change op_c

File I/O – Your assignment 2 will have a question about file I/O.

You will have to read about it yourself.

We are done with the syntax of C!!!

*We missed some minor topics, like switch or some useful function (getc, putc, gets...) and some libraries.

But they are quite straightforward to learn by examples.

https://www.tutorialspoint.com/c_standard_library/stdio_h.htm

Binary Encodings

Binary encodings of integers

The values of digits in a number are positional:

Decimal numbers: $582 = 500 + 80 + 2 = 5 \cdot 10^2 + 8 \cdot 10^1 + 2 \cdot 10^0$

Binary numbers: $10110 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$

Exercises:

- Convert 10011011 from binary to decimal
- Convert 29 from decimal to binary

Binary encodings of integers

Convert 10011011 from binary to decimal:

- $2^7 + 2^4 + 2^3 + 2^1 + 2^0 = 128 + 16 + 8 + 2 + 1 = 155$

Binary encodings of integers

Convert 29 from decimal to binary:

29 is odd, hence the binary should end with 1. *****1

Let's subtract 1, we are left with 28.

Let's divide 28 by 2, and get 14.

14 is even, so the next digit will be 0.... -> *****01

Let's divide 14 by 2, we get 7.

7 is off, so the next digit is 1 → ****101

Binary encodings of integers

Convert 29 from decimal to binary:

Let's try to write 29 as sum of powers of 2.

29 is odd, hence the binary should end with 1. *****1

Let's subtract 1, we are left with 28.

Note $28 = 16 + 12 = 2^4 + 12 = 2^4 + 8 + 4 = 2^4 + 2^3 + 2^2$

So $29 = 2^4 + 2^3 + 2^2 + 2^0$, In binary this is 11101

Binary encodings of integers

Convert 29 from decimal to binary:

ANSWER: 29 in decimal = 11101

Algorithm:

1. $i = 0$
2. while ($N > 0$)
 - 2.1 if N is even
 - 2.1.1 set the 2^i -th digit = 0
 - 2.1.2 set $N = N/2$
 - 2.2 else
 - 2.2.1 set the 2^i -th digit = 1
 - 2.2.2 $N = (N-1)/2$
 - 2.3 $i++$

$i = 0$: $N = 29$ is odd

set $1 \cdot 2^0 \rightarrow ****1$
set $N = (29-1)/2 = 14$

$i = 1$: $N = 14$ is even

set $0 \cdot 2^1 \rightarrow ***01$
set $N = 14/2 = 7$

$i = 2$: $N = 7$ is odd

set $1 \cdot 2^2 \rightarrow **101$
set $N = (7-1)/2 = 3$

$i = 3$: $N = 3$ is odd

set $1 \cdot 2^3 \rightarrow *1101$
set $N = (3-1)/2 = 1$

$i = 4$: $N = 1$ is odd

set $1 \cdot 2^4 \rightarrow 11101$
set $N = (1-1)/2 = 0$

Fixed width encoding

Simple data types are usually fixed in width

`int` is usually 4 byte = 32 bits

Range of `int` is $[-2^{31}, 2^{31}-1]$ (one of the digits is for the sign)

That is between -2,147,483,648 and 2,147,483,647

Larger numbers will result in an *overflow*

Try it in C

Non integer arithmetic

Two common decimal numbers:

$$\frac{1}{3} = 0.3333333333$$

$$\frac{2}{3} = 0.6666666666$$

Cannot write the decimal with finite number of digits.

So we round them

Scientific Notation

A convention to express numbers by their magnitude:

Speed of light = 2.99792458×10^8 m/s

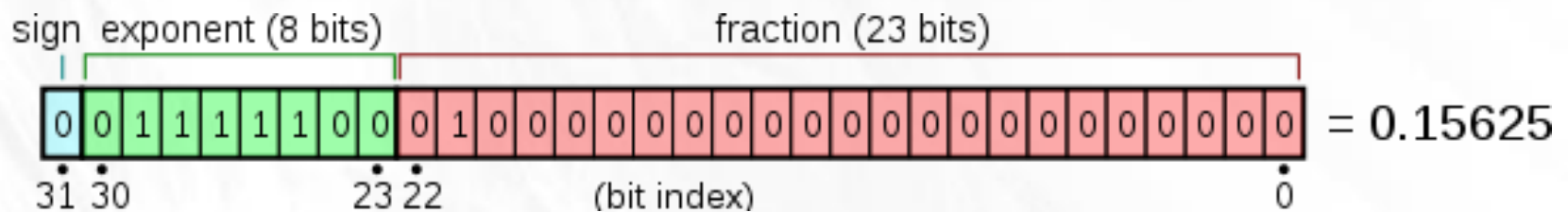
One gigabyte = 1.073741824×10^9 bytes

$\frac{1}{3} = 3.333333333333 \times 10^{-1}$

Same conventions are used for binary

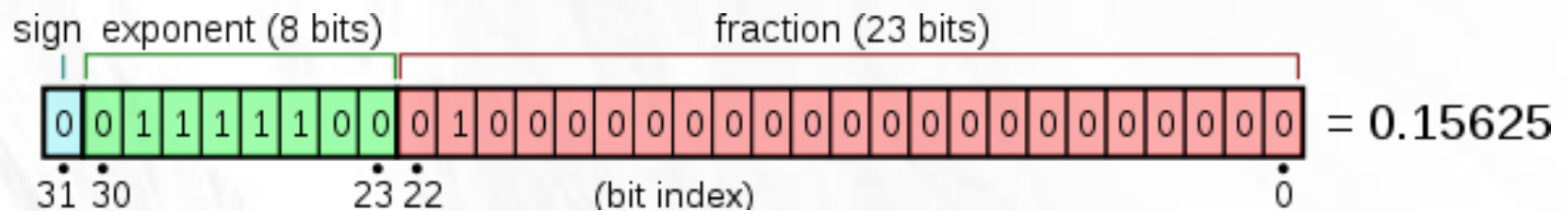
Floating point encoding

A float is composed of 32 bits:



- 1 bit for sign (0 – positive, 1 – negative)
- 23 bits for the significand (*significant digits* of the number)
 - $1.b_{22}b_{21}\dots b_0$
 - b_{22} represents the digit of $\frac{1}{2}$
 - b_{21} represents the digit of $\frac{1}{4}$...
- 8 bits for the exponent – ranges from -127 to 128
- Range of the representation: $1.b_{22}b_{21}\dots b_0 \times 2^{(\text{exp}-127)}$
 - between $\approx 2^{128} \approx 3.40 \times 10^{38}$ and $\approx 2^{-127} \approx 1.17 \times 10^{-38}$

Example



The number is $(-1)^{\text{sign}} \times (1 + \text{fraction}) \times 2^{\text{exp}-127}$

•In this example:

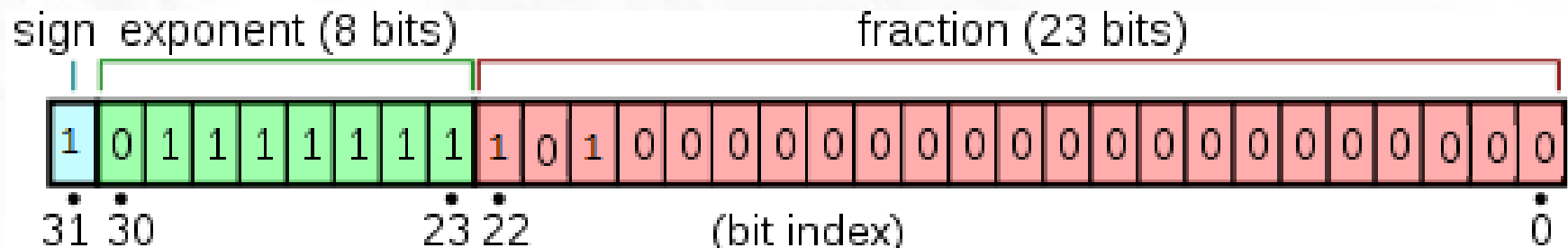
$$1 \times (1 + 1/4) \times 2^{124-127}$$

$$= 1.25 \times 2^{-3}$$

$$= 1.25/8 = 0.15625$$

Example

$$-1.625 = -(1 + 5/8) \times 2^0$$



- $5/8 = 1/2 + 1/8$
- Set 1 for the minus sign
- Set $b_0 = 1$ (for $1/2$) and $b_2 = 1$ (for $1/8$)
- The 8 bits of exponent are 127
- $-1.625 = - (1 + 1/2 + 1/8) \times 2^0$

Questions?
Comments?