# Assignment 1 – Basic C Datatypes

Version 1.01 (last update: Sept. 13, 13:00)
Changes highlighted in yellow
Due date: Tue, Sep 21, 11:59 PM

## Summary and Purpose

For this assignment, you will be writing a small C library that implements a number of functions that will help you to understand the basic data types in C.

## Deliverables

You will be submitting:

1) A file called **`a1.h`** that contains your function prototypes (see below).
2) A file called **`a1.c`** that contains your function definitions.
3) A **`makefile`** that contains the instructions to compile your code into an object file called **`a1.o`.**

You will submit all of your work via git to the School's gitlab server. (As per instructions in the labs.)

This is an individual assignment. Any evidence of code sharing will be investigated and, if appropriate adjudicated using the University's Academic Integrity rules.

## Setting up your work environment and using git:

Start the SoCS VM. While running in the VM, begin by cloning an empty directory for your assignment using the command:

**`git clone https://gitlab.socs.uoguelph.ca/2520F21/skremer/A1`**

But, use your own login ID instead of **`skremer`**.

Then, **`cd A1`**, to move to the assignment directory.

Work in the A1 directory. Use the command:

**`git add filename`**

For each file file that you create as part of your code (one .c file, one .h file, and the makefile).

Every so often (especially if you get something working), use the command:

```
git commit -a
```

to commit your changes.

And then make sure to use:

```
git push --all
```

to push everything to the server.

If you want to check out what's on the server you can rename the A1 directory to A1.day1 and then use

```
git clone https://gitlab.socs.uoguelph.ca/2520F21/skremer/A1
```

to get a copy of exactly what is currently in the repo.

## Function prototypes and descriptions for your assignment

```
unsigned char checksum( char *string );
```

A checksum is a value that can be calculated for a given sequences of values and is often used to confirm the accurate transmission or transcription of information.

https://en.wikipedia.org/wiki/Checksum

Characters in the C programming language are a numeric type (i.e. numbers) that can be displayed as symbols using the **`"%c"`** format specifier in **`printf`** statements. This means it is possible to perform mathematical operations on characters. This function should loop over all characters in the string until the null terminator (**`'\0'`**) is reached). Characters that are not alphabetic uppercase or lowercase letters (in the English alphabet) will be ignored. All letters will be treated as integers equal to the letter position relative to **`'a'`** or **`'A'`** in the alphabet for lower- and upper-case letters, respectively. I.e. **`'d'`** should be treated as 3, and **`'G'`** as 6, because they are 3 and 6 letters later than the letter **`'a'`** or **`'A'`**. A sum of all values modulo (division remainder) 26 should be computed and returned. E.g. the string **`"yF"`** should return a value of 3 because **`'y'`** is represented by 24 and **`'F'`** is represented by 5 and (24+5)%26=3. Note that **`checksum`** should never return a value greater than 25.

You may use **`isupper`** and **`islower`** in your code.

```
void caesar( char *string, int rshift );
```

A Caesar cipher is a very simple encryption technique in which each letter in the plain text is replaced by a letter some fixed number of positions down in the alphabet.

https://en.wikipedia.org/wiki/Caesar_cipher

This function will process a given string which will contain only uppercase letters or spaces followed by the null character. It will replace each letter in the string by a letter that occurs **`rshift`** places further in the alphabet. Note that positions "wrap around" after the **`'Z'`** character. Spaces will be left alone. See the Wikipedia page for an example of **`rshift=23`**.

```
void char2bits( char c, unsigned char bits[8] );
```

This function will call the **`get_bits8`** function, supplied by the instructor, 8 times to retrieve bits 0, through 7 from the character, **`c`** and store the bit values in the array **`bits`**.

```
void bits2str( int bitno, unsigned char *bits, char *bitstr );
```

This function will convert the first **`bitno`** numbers stored in **`bits`** into a string in **`bitstr`**. It will do this by copying each number from its index in **`bits`** to the same index in **`bitstr`** and add the value **`'0'`**. Finally, it will place the null terminator value **`'\0'`** in the final position at the index **`bitno`** in **`bitstr`** so that the string is null terminated and can be printed using **`printf`** and the **`"%s"`** format specifier. You may assume that the pointer **`bits`** points to **`bitno`** bytes of allocated memory and that **`bitstr`** points to **`bitno+1`** bytes of allocated memory.

```
void ushort2bits( unsigned short s, unsigned char bits[16] );
```

This function will call the **`get_bits16`** function, supplied by the instructor, 16 times to retrieve bits 0, through 15 from the unsigned short integer, **`s`** and store the bit values in the array **`bits`**.

```
void short2bits(short s, unsigned char bits[16] );
```

This function will call the **`get_bits16`** function, supplied by the instructor, 16 times to retrieve bits 0, through 15 from the short integer, **`s`** and store the bit values in the array **`bits`**.

```
short bits2short( char *bits );
```

This function should read characters from the string **`bits`**, and process up to, but non including, the terminating null character. The characters processed with be either the character **`'0'`** or the character **`'1'`**. The function should return a **`short`** that is equal to the *two's complement* binary number that it read, character by character, from the highest valued column, all the way to the 1s column. The value of a two's complement number can be calculated by multiplying each binary digit by its column value, just like a binary number, except that the highest column value is negative. You may call the function **`strlen`** in this fuction.

E.g.

| column value | $-2^7$=-128 | $2^6$=64 | $2^5$=32 | $2^4$=16 | $2^3$=8 | $2^2$=4 | $2^1$=2 | $2^0$=1 |
|---|---|---|---|---|---|---|---|---|
| -8 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 120 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| -128 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 127 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Pointers

## The Last 10%

The above, constitutes 90% of the assignment. If you complete it, you can get a grade up to 90% (Excellent). The rest of the assignment is more challenging and will allow you to get a grade of 90-100% (Outstanding). Make sure you complete the first part well, before proceeding to the following additional part.

Read the Wikipedia pages on:

- "Single-precision floating-point format" (https://en.wikipedia.org/wiki/Single-precision_floating-point_format)

Write the following function:

```
void spff( char *sign, char *exponent, char *significand, float *src );
```

This function should retrieve individual groups of bits from the floating point number at the address given by **`src`**. It should retrieve the sign bit (1), the exponent bits (8) and the significand bits (23). You can use the get_bits8 function supplied by the instructor and the bits2str function that you developed earlier in implementing this function.

It is the caller's responsibility to ensure that **`sign`**, **`exponent`**, and **`significand`** have 1+1, 8+1 and 23+1 writable characters. It is the function's responsibility to write the correct characters ('0's and '1's) into those location and terminate each string with a null character ('\0').

***You can write additional helper functions as necessary to make sure your code is modular, readable, and easy to modify.***

## Testing

You are responsible for testing your code to make sure that it works as required. The CourseLink web-site contains 7 test programs to get you started. However, we will use a different set of test programs to grade your code, so you need to make sure that your code performs according to the instructions above by writing more test code.

Your assignment will be tested on the standard SoCS Virtualbox VM (http://socs.uoguelph.ca/SoCSVM.zip) which will be run using the Oracle Virtualbox software (https://www.virtualbox.org/wiki/Downloads). If you are developing in a different environment, you will need to allow yourself enough time to test and debug your code on the target machine. We will NOT test your code on YOUR machine/environment.

Full instructions for using the SoCS Virtualbox VM can be found at: https://wiki.socs.uoguelph.ca/students/socsvm.

## Makefile

You will create a makefile that supports the following target:

**`a1.o:`** this target should generate your object file.

**`clean:`** this target should delete all **`.o`** and executable files.

All compilations and linking must be done with the **`-Wall -pedantic -std=c99`** flags and compile and link **without any warnings or errors**.

## Git

You must submit your .c, .h and makefile using git to the School's git server. Only code submitted to the server will be graded. Do ***not*** e-mail your assignment to the instructor. We will only grade one submission; we will only grade the last submission that you make to the server and apply any late penalty based on the last submission. So once your code is complete and you have tested it and you are ready to have it graded make sure to commit and push all of your changes to the server, and then do not make any more changes to the A1 files on the server.

## Academic Integrity

Throughout the entire time that you are working on this assignment. You must not look at another student's code, now allow your code to be accessible to any other student. You can share additional test cases (beyond those supplied by the instructor) or discuss what the correct outputs of the test programs should be, but do not share ANY code with your classmates.

Also, do your own work, do not hire someone to do the work for you.

## Grading Rubric

| | |
|---|---|
| checksum | 2 |
| caesar | 2 |
| char2bits | 2 |
| bits2str | 2 |
| ushort2bits | 2 |
| short2bits | 2 |
| bits2short | 2 |
| style | 2 |
| makefile | 2 |
| spff | 2 |
| Total | 20 |

## Ask Questions

The instructions above are intended to be as complete and clear as possible. However, it is YOUR responsibility to resolve any ambiguities or confusion about the instructions by asking questions in class, via the discussion forums, or by e-mailing the course e-mail.