C Programming II
2023 Spring
Homework 01

Instructor: Po-Wen Chi

Due: 2023.03.21 PM 11:59

Policies:

- **Zero tolerance** for late submission.
- **Plagiarism is not allowed.** Both source and copycat will be **zero**.
- You need to prepare a README file about how to make and run your program. Moreover, you need to provide your name and your student ID in the README file.
  - Your Name and Your ID.
  - The functional description for each code.
  - Anything special.
- Please pack all your submissions in one zip file.
- For convenience, your executable programs must be named following the rule hwXXYY, where the red part is the homework number and the blue part is the problem number. For example, hw0102 is the executable program for homework #1 problem 2.
- I only accept **PDF**. MS Word is not allowed.
- Do not forget your Makefile. For convenience, each assignment needs only one Makefile.

1 My String Library (20 pts)

In this class, I have shown you some standard string functions. Of course, you should use them when coding. However, sometimes you may need some functions that are not included in the standard string library. Do not worry, you can implement on your own. For your practice, I want you to implement some existing string functions in your own way.

```
char *mystrchr(const char *s, int c);
char *mystrrchr(const char *s, int c);
size_t mystrspn(const char *s, const char *accept);
size_t mystrcspn(const char *s, const char *reject);
char *mystrpbrk(const char *s, const char *accept);
char *mystrstr(const char *haystack, const char *needle);
char *mystrtok(char *str, const char *delim);
```

The usage of these functions should be the same with the standard version, including their return values. All requirements are in the manual. You need to prepare **mystring.h**, and TA will prepare **hw0101.c**. Of course, Makefile is your own business. Do not forget to make **hw0101.c** in your Makefile.

You cannot call the corresponding standard functions directly in your implementation.

2 String Insertion (20 pts)

Please develop the following function.

```
#include <stdint.h>

// Insert pStr2 into pStr1's location.
// For example
// char *pStr = NULL;
// strinsert( &pStr, "Happy", 1, "ABC" );
// The output string should be HABCappy
int32_t strinsert( char **ppResult, const char *pStr1, int32_t location, const
    char *pStr2 );
```

You need to prepare **insert.h** and TA will prepare **hw0102.c**. Of course, Makefile is your own business. Do not forget to make **hw0102.c** in your Makefile.

3 Abacus 算盤 (20 pts)

Do you know what abacus is and how to use it? You can see Fig. 1 for example. If you do not know how to use it, you can see the following link. You may wonder why I do not provide the url of wikipedia. The reason is that there are too many different Abacuses. For your simplicity, I use Japanese Abacus, soroban.

`https://knews.cc/zh-tw/collect/gvbqzry.html`

This time, I want you to develop a structure and related functions for the Japanese Abacus. The structure and functions are defined as follows.

```
#include <stdint.h>

typedef struct _sAbacus
{
  uint8_t number;     // 0-255
  uint8_t *pUpperRod; // Each element is in {0,1}
  uint8_t *pLowerRod; // Each element is in {0,1,2,3,4}
}sAbacus;

```

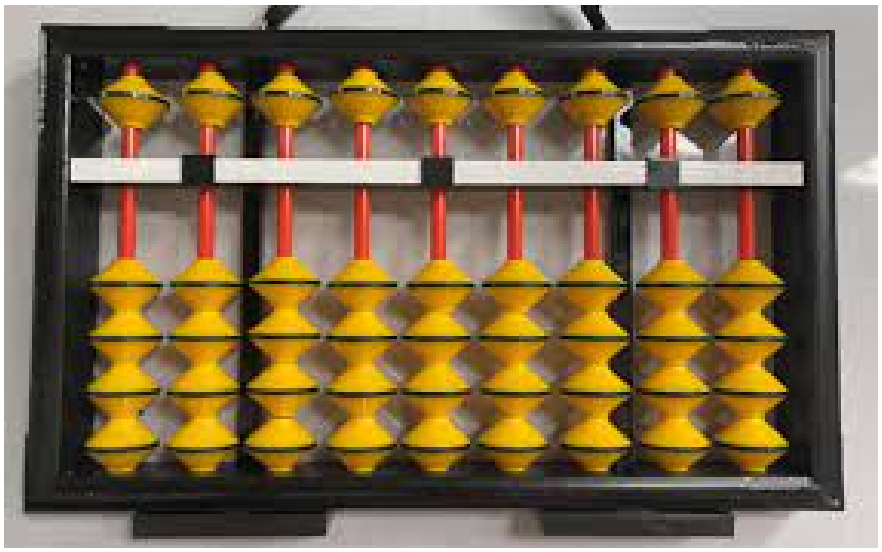

Figure 1: The soroban (算盤, そろばん, counting tray) is an abacus developed in Japan.

```
sAbacus * abacus_init( void );
void abacus_free( sAbacus * );

// Set pA according to pStr
// For example, pStr is "123456789"
// pA -> number = 9;
// pA -> pUpperRod: {0,0,0,0,1,1,1,1,1}
// pA -> pLowerRod: {1,2,3,4,0,1,2,3,4}
// if the length of pStr > 255 or the string contains alphabets or special
    character or NULL, return -1.
// Otherwise, return 0
int32_t abacus_set( sAbacus *pA, char *pStr );

// a = b + c
// Successful: return 0; otherwise, return -1
int32_t abacus_add( sAbacus *pA, sAbacus b, sAbacus c );

// a = b - c
// Successful: return 0; otherwise, return -1
int32_t abacus_del( sAbacus *pA, sAbacus b, sAbacus c );

// EX: "172", you should print
// * *
//  *
// ---
// ***
//  **
// *
// ***
// ***
// Successful: return 0; otherwise, return -1
int32_t abacus_print( sAbacus a );
```

The color of the upper elements should be red while the color of the lower elements should be yellow You need to prepare **abacus.h**, and TA will prepare **hw0103.c**. Of course, Makefile is your own business. Do not forget to make **hw0103.c** in your Makefile.

4 JSON Reader (20 pts)

JSON is a popular text format for information exchange. You can see the following link to study this format.

`https://zh.wikipedia.org/wiki/JSON`

Now, I want you to develop a JSON reader. The user can input a JSON string which contains at most 8192 characters. Then the user will give a key and you need to return its value.

```
$ ./hw0104
Please enter the JSON string:
{ "firstName": "John", "lastName": "Smith", "sex": "male", "age": 25, "address
    ": { "streetAddress": "21 2nd Street", "city": "New York", "state": "NY",
    "postalCode": "10021" }, "phoneNumber": [ { "type": "home", "number": "212
     555-1234" },{ "type": "fax", "number": "646 555-4567" } ] }
Choice (0:Exit,1:Get) : 1
Key: firstName
Value: John
Choice (0:Exit,1:Get) : 1
Key: address.city
Value: New York
Choice (0:Exit,1:Get) : 1
Key: phoneNumber[0].number
Value: 212 555-1234
Choice (0:Exit,1:Get) : 0
Bye
```

For your simplicity, you can

- treat **space** as the only white space, which means the input string will not contain **newline** except the last character.
- skip the JSON format check.
- ignore the case that the key has **dot**.

BTW, you cannot use any existing JSON library.

5 Potato Ball(20 pts)

There's some rumor that jw910731(TA Wu) will fry sweet potato balls in front of the NTNU CSIE department office, which is untrue. In fact, Smallten(TA Yu) summons magical potatoes from another world, and Subarya(TA Su) farms some normal potatoes. These potatoes then get fried by themself and become potato balls, the process does not change the weight of the potato (after frying a potato weight x, you get a potato ball with weight x). The

magical potatoes have magical effect on them, which is kept after frying. Now, you need to fry two different types of potato balls in the same way. To simulate this scenario mentioned above, TA Wu has already prepared a header file that describes the situation, you need to implement the functions declared in the header file. The TA will prepare `hw0105.c`, it may look like this:

```c
#include "oop.h"
#include <string.h>

int main() {
    struct PotatoProducer smallten, subarya;
    init_smallten(&smallten);
    init_subarya(&subarya);

    // smallten summon potato and fry potato ball
    struct Potato *p = smallten.produce(&smallten);
    p->print(p);
    struct PotatoBall *pb = p->fry(&p); // p pointer set to NULL
    pb->print(pb);
    pb->dtor(&pb); // pb pointer set to NULL

    // subarya produce potato and fry potato ball
    p = subarya.produce(&subarya);
    p->print(p);
    pb = p->fry(&p); // p pointer set to NULL
    pb->print(pb);
    strncpy(subarya.name, "Handsome Subarya", 31);
    subarya.name[31] = 0;
    pb->print(pb);
    pb->dtor(&pb); // pb pointer set to NULL
    return 0;
}
```

And it's output should look like this:

```
Magical Potato:
  weight: 807
  produced by: Smallten
  magical effect: Slowness
  magic level: 3
Magical Potato Ball:
  weight: 807
  produced by: Smallten
  magical effect: Slowness
  magic level: 3
Potato:
  weight: 658
  produced by: Subarya
Potato Ball:
  weight: 658
  produced by: Subarya
Potato Ball:
  weight: 658
  produced by: Handsome Subarya
```

You may ask what is the formula to get those number and attribute, just do it randomly. You may need `rand()` function to accomplish random number generating.
It is forbidden to rewrite the header file because it is carefully crafted so that you can learn patterns and techniques to handle complex situations elegantly.

P.S. Sweet potato ball is a lie.

6 Bonus: wchar_t (5 pts)

In this class, I said that I prefer to keep **char** as a character type. As you know, it is impossible to encode all Chinese words in **char**. Actually, in C standard library, there is a type called **wchar_t**, which implies **Wide Character**. It was first defined in C90 as an integral type whose range of values can represent distinct codes for all members of the largest extended character set specified among the supported locales.

It sounds great, right? However, when considering **portability**, **you should not use wchar_t**. Why? Please write down your reason and provide some references.