Assignment name: argo

subject.txt

Exp. files: argo.c

Allowed functions: getc, ungetc, printf, malloc, calloc, realloc, free, isdigit, fscanf, write

- - - - - -

Write a function argo that will parse a json file in the structure declared in argo.h:

```
int argo(json *dst, FILE *stream);
        dst    is the pointer to the AST that you will create.
        stream is the file to parse (man FILE)
```

Your function will return 1 for success and -1 for failure.
If an unexpected token is found you will print the following message in stdout:
"Unexpected token '%c'\n" or "Unexpected end of input\n" if the token is EOF.

In case of doubt how to parse json, read rfc 8259. But you won't need it as the format is simple.

Tested with the main, the output should be exactly the same as the input (except for errors).

There are some functions in argo.c that might help you.

Examples that should work

echo -n input | ./argo /dev/stdin | cat -e	output
'1'	1$
'"bonjour"'	"bonjour"$
'"escape! \" "'	"escape! \" "$
'{"tomatoes":42,"potatoes":234}'	{"tomatoes":42,"potatoes:234}$
'{"recursion":{"recursion":{recursion":{"recursion":{"recursion"}}}}'	{"recursion":{"recursion":{"recursion":{"recursion":{"recursion"}}}}$
'"unfinished string'	Unexpected end of input $
'"unfinished string 2\"'	- " -
'{"no value?":}'	Unexpected token '}'$

argo.h

```
# include <stdio.h>   , <stdbool.h> , <ctype.h> ,<string.h> , <malloc.h>
typedef  struct  json {
         enum {
                  MAP,
                  INTEGER,
                  STRING
         } type;
         union {
                  struct {
                           struct pair * data;
                           size_t          size ;
                  } map;
                  int      integer;
                  char     *string;
         };
}        json ;


typedef  struct  pair {
           char  *key ;
           json  value;
} pair ;


void    free_json(json j) ;
int     argo( json *dst, FILE * stream),
# include "argo.h"                                                    argo.c

int peek (FILE *stream)
{   int c = getc (stream);
    ungetc (c, stream);
    return c;
}

void   unexpected (FILE *stream)
{
    if (peek (stream)  != EOF)
        printf( " Unexpected token '%c'\n" , peek (stream));
    else
        printf( " Unexpected end of input \n" );
}
int  accept ( FILE  *stream  ,  char c)
{
      if (peek (stream) == c)
      {
          (void)  getc (stream);
           return 1;
      }
      return 0;
}

int  expect ( FILE  *stream , char c)
{   if (accept (stream, c))
          return 1;
    unexpected (stream);
    return 0;

}
```

main.c

```c
int main (int argc, char *argv)
{
    if (argc != 2)
        return 1;
    char *filename = argv[1];
    FILE *stream = fopen(filename, "r");
    json file;
    if (argo(&file, stream) != 1)
    {
        free-json(file);
        return 1;
    }
    serialize(file);
    printf("\n");
}


void serialize (json j)
{
  switch (j.type) {
        case INTEGER:
            printf("%d", j.integer);
            break;
        case STRING:
            putchar('"');
            for (int i=0; j.string[i]; i++)
            {
                if (j.string[i] = '\\' || j.string[i] = '"')
                    putchar('\\');
                putchar(j.string[i]);
            }
            putchar('"');
            break
        case MAP:
            putchar('{');
            for (size_t i=0; i < j.map.size; i++)
            {
                if (i != 0)
                    putchar(',');
                serialize((json){.type = STRING, .string = j.map.data[i].key});
                putchar(':');
                serialize(j.map.data[i].value);
            }
            putchar('}');
            break;
  }
}
```

```c
void free-json (json j) {
    switch (j.type)
    {
        case MAP:
            for (size_t i=0; i < j.map.size; i++)
            {
                free(j.map.data[i].key);
                free-json(j.map.data[i].value);
            }
            free(j.map.data);
            break;
        case STRING:
            free(j.string);
            break;
        default:
            break;
    }
}
```