Middle East Technical University
Department of Computer Engineering

CENG 140

Spring 2019

Take Home Exam 3
[**Version: 1.1**]

REGULATIONS

Due date: 10 June 2019, Monday [23:59] *(Not subject to postpone)*

Submission: Electronically. You will be submitting your program source code written in a file which you will name as `the3.c`. Follow news group for submission method. Resubmission is allowed (till the last moment of the due date), The last will replace the previous.

Team: There is **no** teaming up. The take home exam has to be done/turned in individually.

Cheating: **This is an exam:** all parts involved (source(s) and receiver(s)) get zero+both parts will be subject to disciplinary action.

INTRODUCTION

Files do not have to be homogeneous (containing the same type of data). When this is the case, it becomes complicated to investigate the content. In this exam you are going to write a file investigator. You will take three inputs

- A File name
- A File layout description
- Fields that we want to be printed.

The idea is to navigate on the file, go to the exact position where the data we are interested in is located at, fetch the data, display it and then move to the next data field we are interested in.

To get it straight from the start, you are not expected nor allowed to read any array data into the memory. When you are asked for any such item you will get it by a file access. On the contrary, you are allowed (and encouraged) to fetch from disk and store into memory any non-array (aka scalar) data.

PROBLEM

A lay out description is as follows.

$$\begin{array}{rcl}
\langle\text{layout description}\rangle & ::= & \langle\text{data description}\rangle \mid \langle\text{data description}\rangle\ \textbf{,}\ \langle\text{layout description}\rangle \\
\langle\text{data description}\rangle & ::= & \langle\text{integral data description}\rangle \mid \langle\text{floating data description}\rangle \\
\langle\text{integral data description}\rangle & ::= & \langle\text{integral type}\rangle\ \langle\text{integral data}\rangle \\
\langle\text{floating data description}\rangle & ::= & \langle\text{floating type}\rangle\ \langle\text{floating data}\rangle \\
\langle\text{integral data}\rangle & ::= & \langle\text{integral identifier}\rangle \mid
\end{array}$$

		⟨integral identifier⟩ `[` ⟨atomic⟩ `]` |
		⟨integral identifier⟩ `[` ⟨atomic⟩ `]` `[` ⟨atomic⟩ `]`
⟨floating data⟩	::=	⟨floating identifier⟩ |
		⟨floating identifier⟩ `[` ⟨atomic⟩ `]` |
		⟨floating identifier⟩ `[` ⟨atomic⟩ `]` `[` ⟨atomic⟩ `]`
⟨integral type⟩	::=	`char` | `uchar` | `int` | `uint` | `long`
⟨floating type⟩	::=	`float` | `double`
⟨integral identifier⟩	::=	⟨identifier⟩
⟨floating identifier⟩	::=	⟨identifier⟩
⟨query⟩	::=	⟨integral data⟩ | ⟨floating data⟩
⟨query list⟩	::=	⟨query⟩ | ⟨query⟩ `,` ⟨query list⟩
⟨atomic⟩	::=	⟨natural number⟩ | ⟨integral identifier⟩
⟨natural number⟩	::=	`0` | `1` | ... | `100000000`
⟨identifier⟩	::=	⟨letter⟩ | ⟨letter⟩ ⟨letternums⟩
⟨letternums⟩	::=	⟨letter⟩ | ⟨digit⟩ | ⟨letter⟩ ⟨letternums⟩ | ⟨digit⟩ ⟨letternums⟩
⟨letter⟩	::=	`a` | `b` | ... | `z`
⟨digit⟩	::=	`0` | `1` | ... | `9`

In addition to this

- A ⟨*layout description*⟩ can contain any number of blanks provided that no ⟨*atomic*⟩ is split.
- Some clarifying examples for a ⟨*data description*⟩ in the ⟨*layout description*⟩ line:

  `int s1x` : means at that position of the file an `int` data exists which we will name as `s1x` in this run.

  `float qq` : means at that position of the file a `float` data exists which we will name as `qq` in this run.

  `int sq23[20]` : means at that position of the file 20 `int` data exist and we will refer to them as a one dimensional array with the name `sq23` in this run.

  `uint baba[s1x]` : means at that position of the file `s1x` many `unsigned int` data exist. `s1x` has appeared before we have arrived at this ⟨*data description*⟩. That integer value in the file that corresponds to `s1x` is used for the definition of the size of the `baba` array.

  `uint abba[s1x][s1x]` : means at that position of the file a two dimensional array exists where the size of both dimensions will come from the integer identifier `s1x`, priorly defined. The ordering of the elements is exactly the same as of C.

- Some clarifying examples for a ⟨*query*⟩ in the ⟨*query list*⟩ line:

  `s1x` : print on a new line the (integer) value that was corresponding to `s1x` in the file.

  `qq` : print on a new line the (floating point) value that was corresponding to `qq` in the file.

  `sq23[11]` : print on a new line the 12^{th} (integer) value of 20 (integer) values in the file which we referred to as an array with the name `sq23`. Note that the index starts at 0 (similar to C).

  `sq23[s]` : print on a new line the $(s+1)^{th}$ (integer) value of 20 (integer) values in the file which we referred to as an array with the name `sq23`. The value `s` is referring to must be < 20.

- No '`[0]`' can appear in the ⟨*layout description*⟩ line.[1]
- In the ⟨*layout description*⟩ any ⟨*identifier*⟩ used to define the size of an array (i.e. appear in a pair of square brackets) will have been defined in a prior data description (prior means a data description to the left of the usage) [2]
- Any ⟨*atomic*⟩ in a query that is used in an array index has a valid value as far as the index range is concerned.

SPECIFICATIONS

- The data types comply with Intel Processor, 64 bit Linux. (all department PC's are installed with this version). Do not worry about the differences of internal data representation as far as processors and C compilers are concerned. Your program has not to be portable. It is expected to run on a Intel Processor, 64 bit Linux, being compiled by gcc.
- The input is 3 lines for the standard input.
  - The first line is a file name (can contain a path) of Linux.
  - The second line contains the ⟨*layout description*⟩
  - The third line contains a ⟨*query list*⟩
- While your program is run for grading your program and heap space will be limited to an amount that will disable you from reading in all the data and then process it. Do not even think of reading in any array.
- The use of period operators (to access subfields of structures, if you define any) is forbidden except in define macros.
- An ⟨*identifier*⟩ has at most 20 characters.
- There will be at most 100 ⟨*data description*⟩s.
- In the grade testing of your program there will be no erroneous input. *(This is a big asset that simplifies your program)*
- The use of any other file throughout your program is forbidden (why shall you need it anyway!).

GOLDEN HINT

- Create an identifier table where you keep for each variable: name (`strdup()`'ed while reading), data type, (1-dim/2-dim) array or not, the position in file (as a value suitable to go along with `SEEK_SET`), and the value if it is not an array variable. Make use of the `union` construct for alternatives that can occur here.
- Process the three input lines straight from the stdin. Do not read in the whole input lines and then start processing it. You will only mess up your code and spend unnecessary memory. The problem definition allows one-pass processing of input.

[1] *This and the above item could have been expressed also in BNF notation but that would severely injure readability.*

[2] *This is to simplify the exam*

1 EXAMPLE

Assume you were given a file `sample.dat` that was created by the run of the following program.

```
#include<stdio.h>
FILE *f;

int n=3, m=7, b[3][7];
unsigned int q=882334423;
float ff=3.1415926, a[10];
double s, c[3];
long x=1234567901011;

main()
{ int i,j;
  f = fopen("sample.dat","w");
  for(i=1, a[0]=1.5; i<10; i++) a[i] = 1.5*a[i-1];
  for(i=1; i<4; i++) c[i-1] = ((double)i*10)/7.0 + i*i;
  for(i=0; i<3; i++) for (j=0; j<7; j++) b[i][j] = 200 - (i*j*100+i+j);
  fwrite(&n,sizeof(int),1,f);
  fwrite(&ff,sizeof(float),1,f);
  fwrite(&m,sizeof(int),1,f);
  fwrite(a,sizeof(float),10,f);
  fwrite(c,sizeof(double),n,f);
  fwrite(&x,sizeof(long),1,f);
  fwrite(&q,sizeof(unsigned int),1,f);
  fwrite(b,sizeof(int),n*m,f);
  fwrite(&s,sizeof(double),1,f);
  fclose(f); }
```

Now assume the input lines to the `the3.c` program are as follows[3]

```
sample.dat
int n, float fiko, int kek, float ali[10], double s[n],long w,uint ka3x1,int p12[n][fiko], double z
fiko,n,   p12[1][n],ali[1],ali[kek],ka3x1,    kek,  w,s[2]
```

The expected output of the program is:

```
3.141593
3
-104
2.250000
25.628906
882334423
7
1234567901011
13.285714
```

[3]Left margin is displayed back intended for line fitting purpose only.

MAKE UP QUESTION FOR THE2

THE2 was about the implementation of Dijkstra's algorithm of infix expression evaluation. If you have not got a full grade then you have a chance to makeup.

Assume the BNF notation for the problem above was updated to be:

⟨layout description⟩	::=	⟨data description⟩ | ⟨data description⟩ `,` ⟨layout description⟩
⟨data description⟩	::=	⟨integral data description⟩ | ⟨floating data description⟩
⟨integral data description⟩	::=	⟨integral type⟩ ⟨integral data⟩
⟨floating data description⟩	::=	⟨floating type⟩ ⟨floating data⟩
⟨integral data⟩	::=	⟨integral identifier⟩ |
		⟨integral identifier⟩ `[` ⟨expression⟩ `]` |
		⟨integral identifier⟩ `[` ⟨expression⟩ `]` `[` ⟨expression⟩ `]`
⟨floating data⟩	::=	⟨floating identifier⟩ |
		⟨floating identifier⟩ `[` ⟨expression⟩ `]` |
		⟨floating identifier⟩ `[` ⟨expression⟩ `]` `[` ⟨expression⟩ `]`
⟨integral type⟩	::=	`char` | `uchar` | `int` | `uint` | `long`
⟨floating type⟩	::=	`float` | `double`
⟨integral identifier⟩	::=	⟨identifier⟩
⟨floating identifier⟩	::=	⟨identifier⟩
⟨expression⟩	::=	⟨expression⟩ ⟨operator⟩ ⟨expression⟩ |
		`(` ⟨expression⟩ `)` | ⟨atomic⟩
⟨query⟩	::=	⟨integral data⟩ | ⟨floating data⟩
⟨query list⟩	::=	⟨query⟩ | ⟨query⟩ `,` ⟨query list⟩
⟨atomic⟩	::=	⟨natural number⟩ | ⟨integral identifier⟩
⟨operator⟩	::=	`+` | `-` | `*` | `/` | `\%`
⟨natural number⟩	::=	`0` | `1` | ... | `100000000`
⟨identifier⟩	::=	⟨letter⟩ | ⟨letter⟩ ⟨letternums⟩
⟨letternums⟩	::=	⟨letter⟩ | ⟨digit⟩ | ⟨letter⟩ ⟨letternums⟩ | ⟨digit⟩ ⟨letternums⟩
⟨letter⟩	::=	`a` | `b` | ... | `z`
⟨digit⟩	::=	`0` | `1` | ... | `9`

The only change is that now an index value (both in a ⟨*data description*⟩ as well as in a ⟨*query*⟩) can be an ⟨*expression*⟩. If you are able to incorporate this change into your program you will get a bonus that will compensate for your loss in THE2.

Additional specifications are:

- Precedences (greater value means performed first), associativity and arity of the operators are

Operator	Precedence	Associativity	arity
`+ -`	1	left	binary
`* / %`	2	left	binary

- It is possible that ⟨*expression*⟩ contains unnecessary pairs of parenthesis.

MAKEUP GRADING

Let us assume you get a grade G (out of 100) from the program that implements 'expressions' and obtained H (out of 100) from THE2 evaluation. Then you will get an increase in your THE2 grade that is equivalent to

$$\min(G/2, 100 - H)$$

The THE3 evaluation is done independent of this:
If you chose to submit for makeup, you are supposed to submit two programs. In addition to the program which is the solution to the above defined problem (the simple one) that you named as `the3.c` you will submit a second program (solution to the 'expression' problem) that you will name as `the3expr.c`.
Even if both of your programs are exactly the same you are supposed to make the two submissions.
The submission procedure of the TWO programs will be announced on the course news group.

HAYROLA NEVZAT?
ABİ CENG 140'A DÜŞTÜM BEN!!
E KALKSANA..?
HERKES AYNI ŞEYİ SÖYLÜYOR. DUR BEN DE DÜŞEYİM NEVZATÇIM DİYEN YOK..
KAFANI MI VURDUN SEN?
PEKMEZİ FALAN HEP AKTI GİTTİ...