Compiler Project

1 CONSTANTS

integers, chars and immutable strings.

test file:constant.sc

assembly file:constant.as

```
    push    'c'
    push    sp[-1]
//variable x not defined, saved at fp[0]
    pop fp[0]
    push    27
    push    sp[-1]
//variable i not defined, saved at fp[1]
    pop fp[1]
    push    "hello, world!"
    push    sp[-1]
//variable s not defined, saved at fp[2]
    pop fp[2]
    push    fp[0]
    putc
    push    fp[1]
    puti
//fp now at 3
    push    fp[2]
    puts
//fp now at 3
```

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ make c6c
gcc -o c6c lex.yy.c y.tab.c c6c.c
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./c6c tests/constant.sc>constant.
as
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas constant.as
c
27
hello, world!
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$
```

Above is a screenshot of the code running.

2 VARIABLES

variables are alphanumeric, up to 12 chars long and have no particular type.

test file: var.sc

assembly file:var.as

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./c6c tests/var.sc>var.as
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas var.as
200
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$
```

variables can be either local or global. You can refer to a global variable inside a function with "@VAR"

test file:var2.sc

assembly file:var2.as

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./c6c tests/var2.sc>var2.as
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas var2.as
16
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$
```

3 ARRAYS

We support multidimensional arrays.

Arrays of integers and chars can be initialized with a single value. Arrays of chars can also be initialized by a string.

The syntax for declaring a multidimensional array is: `array matrix[5,4,3]`

test file:array.sc

assembly file:array.as

Note that this source file combines many tests.

tests for multi-dimensional arrays:

```
puts("multiple dimensional array");
array a[10] = 8, arr[2,3,4], c[3], d[2,2,3] = 7;
```

tests for multiple i/o:

```
geti(c[1], c[2]); // multiple geti
puti(c[1], c[2]); // multiple puti
```

tests for obtaining array elements:

```
arr[k, k, index[2] + k] = 100;
puti(a[index[0]], a[d[1,1,1]], c, arr[index[2], index[2] , index[
2] + k]);
```

printing out an integer array:

```
puts("print integer array");
array nums[5] = 3;
puti(nums);
```

arrays of immutable strings and printing them out with format:

```
puts("array of nas strings");
array strs[3];
strs[0] = "\hello\t\"world\"\n";
```

arrays of chars (real strings):

```
puts("real string array");
array str [10] = "66666666\n";
```

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./c6c tests/array.sc>array.as
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas array.as
multiple dimensional array
1
2
1
2
8
8
3712
100
print integer array
33333
array of nas strings
Hello
how
Hello
how
This is a nas string: [     \hello      "world"
]
This is a nas string: [              Hello]
This is a nas string: [             literal]
real string array
tomorrow
t
o
to666666
```

4 FUNCTIONS

Functions accept none or multiple arguments. They must appear before they are used and they must return with a value

test file:prime.sc(this is a function that tests whether the input number is a prime)

assembly file:prime.as

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./c6c tests/prime.sc>prime.as
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas prime.as
1
1 is not a prime
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas prime.as
78
78 is not a prime
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas prime.as
113
113 is a prime
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$
```

Another test for recursive functions:

test file:recursive.sc(a function that calculates the Fibonacci sequence numbers recursively)

assemble file:recursive.as

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./c6c tests/recursive.sc>recursiv
e.as
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas recursive.as
1
55
832040
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$
```

5 REAL STRINGS

We can initialize a char array with a string, or output a real string.

We also support escape characters like \n \\ \t.

test file:string.sc

assemble file:string.as

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas string.as
m
This is a nas string: [      hello      "world"
]merry Christmas


vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$
```

6 INPUT, OUTPUT

input syntax: read, geti,getc,gets (read is the same as geti for a single variable)

output syntax:print, puti,putc,puts(print is the same as puti for a single variable)

We support multiple gets and multiple puts.

We also support printing with a format string for the basic types (not including arrays).We assume that if the first argument is a string, it is the format string.

We can also print out integer arrays. The numbers will be printed out one by one without spaces in between.

test file:io.sc

assembly file:io.as

```
ntu-trusty-64:/vagrant/c6$ ./c6c tests/io.sc>io.as
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas io.as
12
44
564
output in reverse order:

the next one is 564
the next one is 44
the next one is 12
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$
```

7 PASS BY REFERENCE

Function parameters can be passed by reference.

'&' needs to be used in the function definition and the function call.

test files: ref.sc swap.sc ref2.sc strlen.sc

```
change (&a){
    a = a+1;
    puti(a);
    return 1;
}

x =1;
change(&x);
puti(x);
```

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./c6c tests/ref.sc>ref.as
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas ref.as
2
2
```

```
swap(&a,&b){
    c = a;
    a = b;
    b = c;
    return 1;
}

x=10;
y=1;
puti(y);
swap(&x,&y);
puti(y);
```

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas swap.as
1
10
```

```
alter(&a[10]){
    a[0] =5;
    return 1;
}

sum(&a[10]){
    s=0;
    for (i=0;i<10;i=i+1;)
    {
        s= s+a[i];
        puti(a[i]);
    }
    return s;
}

array nums[10] = 3;
alter(&nums);
puti(sum(&nums));
puti(nums);
```

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./c6c ref2.sc>ref2.as
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas ref2.as
5
3
3
3
3
3
3
3
3
3
32
5333333333
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$
```

This also shows outputting integer arrays.

```
length(&s[200]){
    l=0;
    while (s[l]!=0){
        l=l+1;
    }
    return l;
}

array greetings[100]="Good morning!\n";
puti(length(&greetings));
```

```
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$ ./nas tests/strlen.as
14
vagrant@vagrant-ubuntu-trusty-64:/vagrant/c6$
```

8 APPLICATION

1. 3-puzzle game

You can find the readme file of this game in test-files.zip/3-puzzle/readme. Compiling and playing instructions are given in that file. Below is the output of the game controller, which includes all game logics and controlling instructions to be sent to the graphic module, which is implemented in C with OpenGL and GLUT.

```
BEGIN
1
DRAW 132
STEP 0
1
DRAW 1032
STEP 1
?
DRAW 1230
STEP 2
SUCCESS
A
BEGIN
2
DRAW 3102
STEP 0
?
DRAW 132
STEP 1
?
DRAW 1032
STEP 2
?
DRAW 1230
STEP 3
SUCCESS
E
END
```

"BEGIN" tells the graphic module to show the welcome message and the prompt for the user to input a lucky number. Then the user inputs a lucky number in the console.

"DRAW xxx" tells the graphic module to update the gameboard to state xxx.

"STEP x" tells the graphic module to update the step counter to x.

If the user inputs a movable slice number, the action will be performed and a DRAW instruction will be generated. If the following input is a questionmark ('?'), a best move will be automatically made and a DRAW instruction will also be generated.

On reaching the goal state, a "SUCCESS" will be generated to tell the graphic module to show a congratulating message. If the user inputs 'A', the game will restart; otherwise, an "END" will be generated to close the graphic module and exit this game.

By running `../../nas 3-puzzle.as | ./3-puzzlegui`, a player can input into the game controller and see the graphic gameboard in a window:

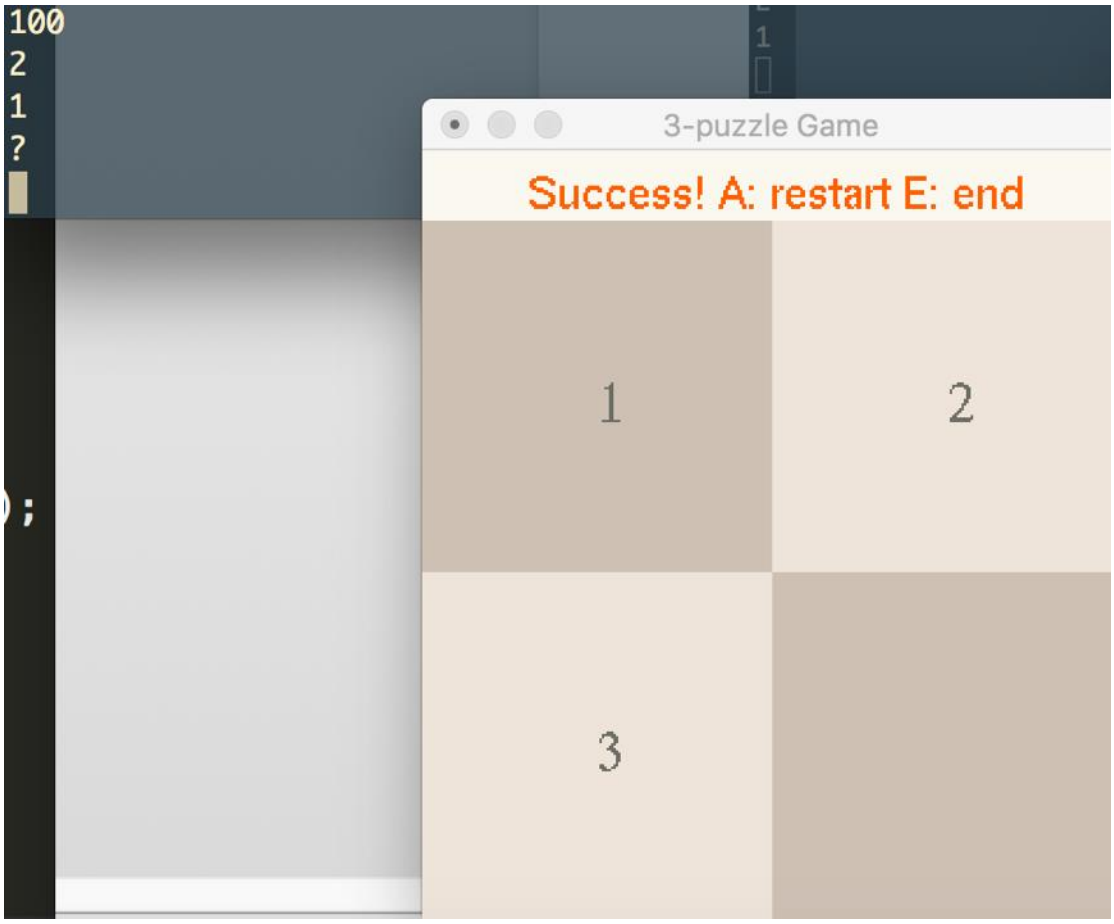
Here we give an example of a complete game circle:

(1) Input a lucky number: 100

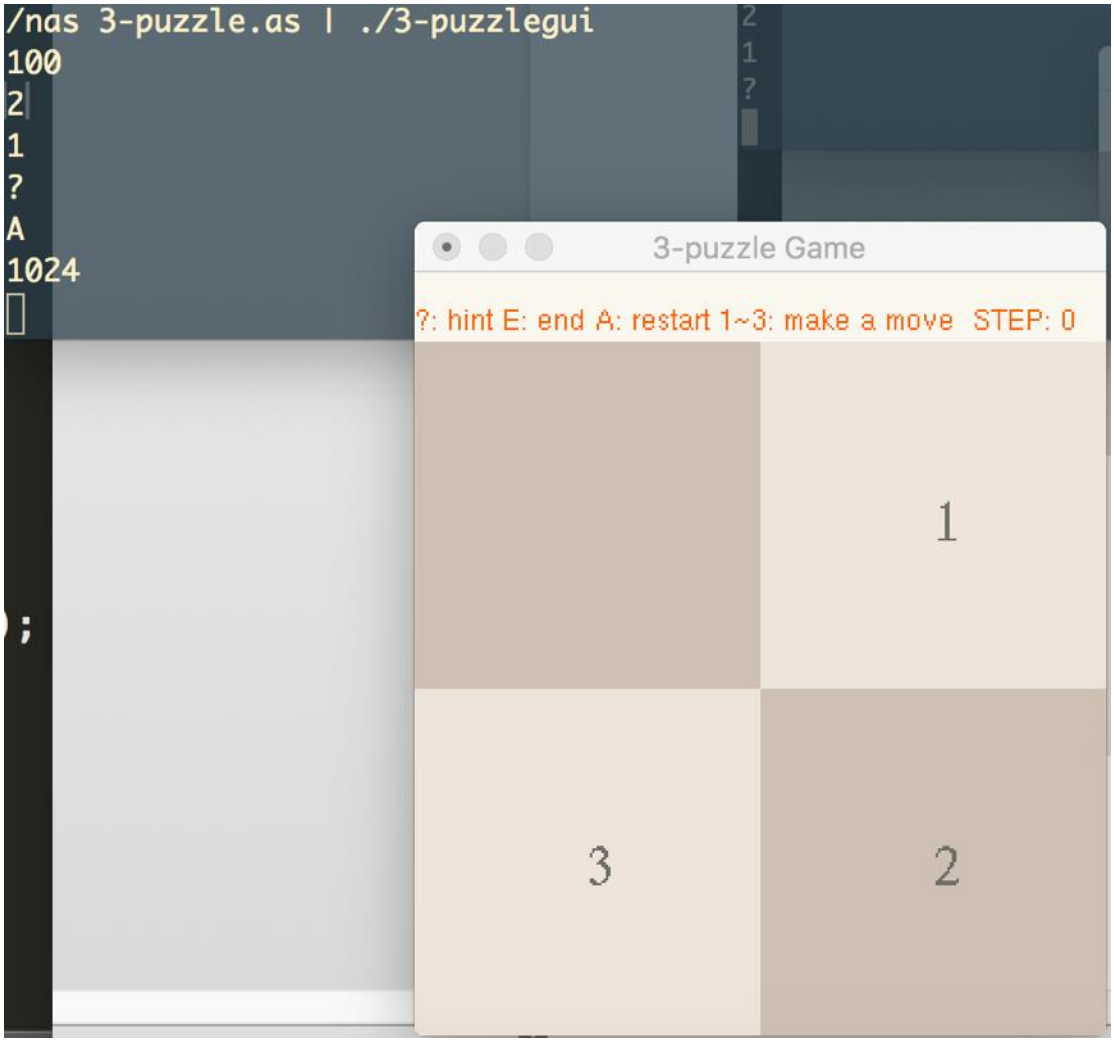
(2) Try to input an illegal move: 2

(3) Input a legal move: 1 (step counter +1)

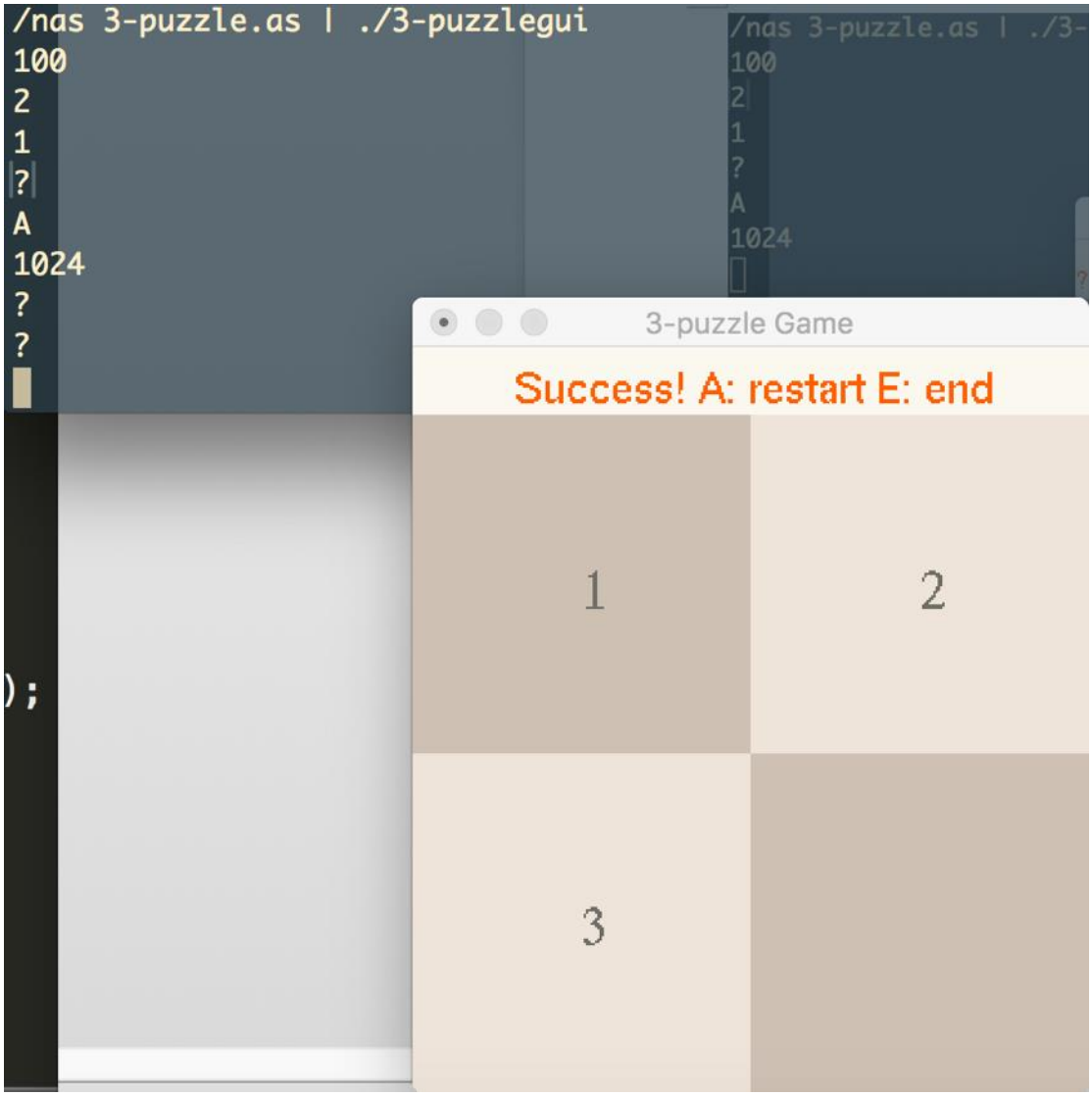
(4) Ask for a intelligent move: ? (move 2 and reaches goal)



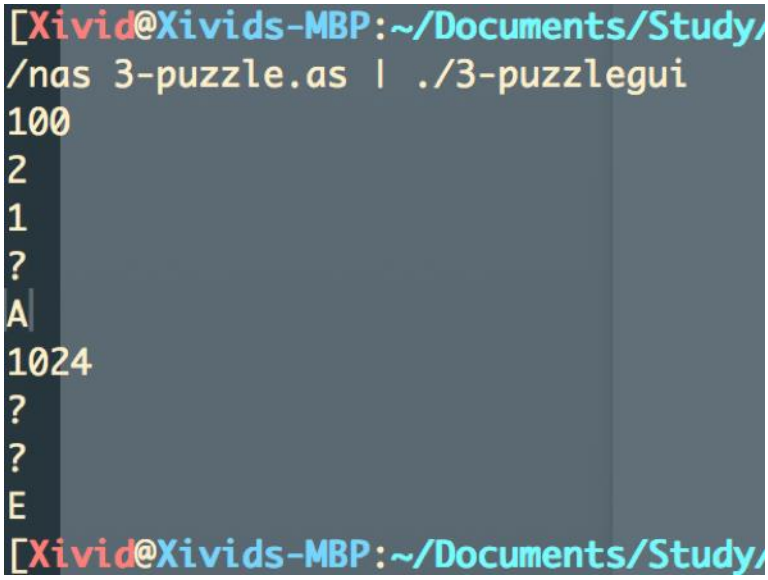
(5) play again: A and input a lucky number 1024



(6) make two intelligent moves: ? and ?



(7) end game: E (exited successfully)



2. sortplot

A very simple application that repeatedly receives an integer as input and output the sorted list of all integers and draw them in a line graph (with matplotlib.pyplot and numpy in Python3).

Usage (working directory: tests/sortplot/, requires matplotlib, numpy and python3):

../../nas sortplot.as | python3 receiver.py

input any number except -1: draw the sorted data since startup

input -1: exit