CBasic Compiler

- a tribute to the first language we learnt -

Message

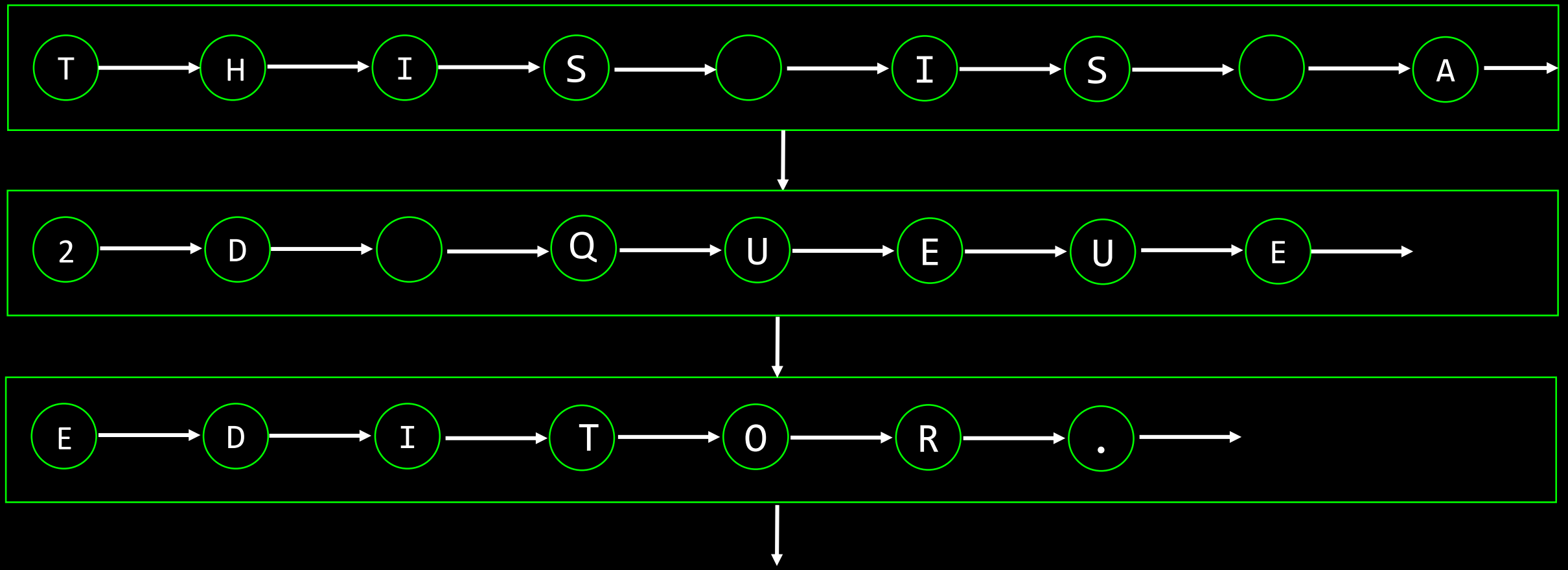
» Press Any Key To Continue...

Commands1.cqb [x]

Some Supported Commands:

+ print
+ if-then-else
+ cls
+ rem
+ return
+ sub
+ canvas
+ bkcolor
+ delay
+ pset
+ inarrow$
+ input
+ for-next-step
+ sleep
+ assignment (=)
+ include
+ dim
+ color
+ circle
+ draw
+ midy$
+ midx$

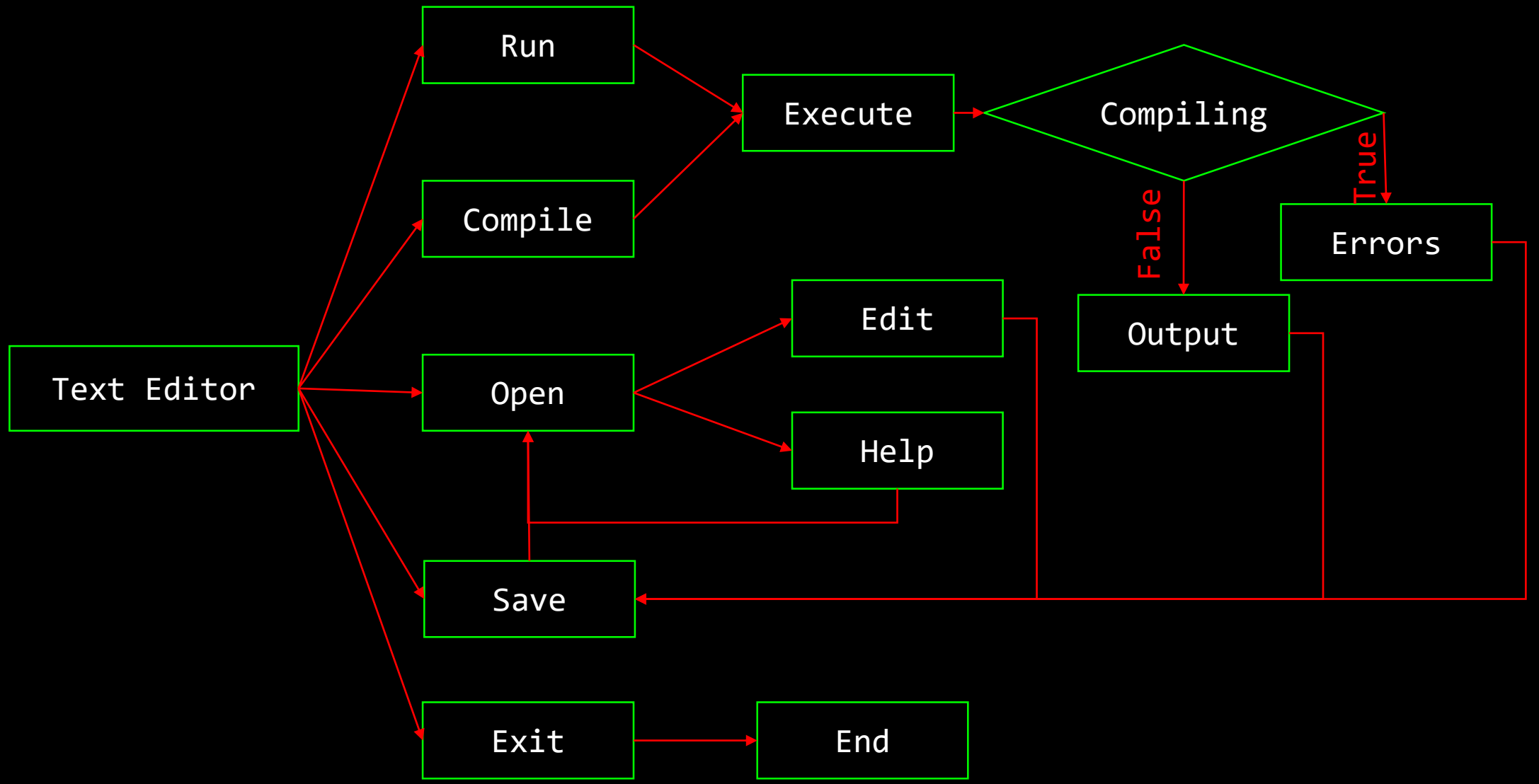
Prog_Edit.cqb [x]



Message

» The above structure helps us to present you with a fully functional writing space that let's you edit the program in all the ways a notepad provides. *(Minus clipboard)*

Structure.cqb [x]




[X]
Welcome to MS-DOS CBasic compiler,
Copyright (c) The Void Club
All rights reserved.
Press any key to continue..._
1:1
Message

help.txt [X]

B.12) Canvas
description and syntax:
open canvas - stars the graphics mode
clear canvas - clears the canvas
close canvas - brings back text mode by closing graphics mode

B.13) Color
description:
- sets the current drawing color
- an intiger between 1 and 15

0 - BLACK	5 - MAGENTA	10 - LIGHTCYAN
1 - BLUE	6 - BROWN	11 - LIGHTRED
2 - GREEN	7 - DARKGRAY	12 - LIGHTMAGNETA
3 - CYAN	8 - LIGHTBLUE	13 - YELLOW
4 - RED	9 - LIGHTGREEN	14 - WHITE

syntax: color var_name

164:1 Message

<F1 - Help> <F2 - Save> <F3 - Open> <F5 - Run> <F9 - Compile> <F10 - Exit>

prog16.txt [X]

```
sub int foo(dim a int)
print a
return a
endsub

sub void main()
cls
dim x int
dim faltu int
print "Enter x: ";faltu
input x_p
x = foo(x)
x = foo2(x)
dim y int
if y > 9 then
print "hey"
else
```

1:1 Message

```
line 11: undefined variable: x_p
line 13: function foo2 should have a prototype
line 21: undefined symbol boom
line 22: undefined variable: t
```

<F1 - Help> <F2 - Save> <F3 - Open> <F5 - Run> <F9 - Compile> <F10 - Exit>

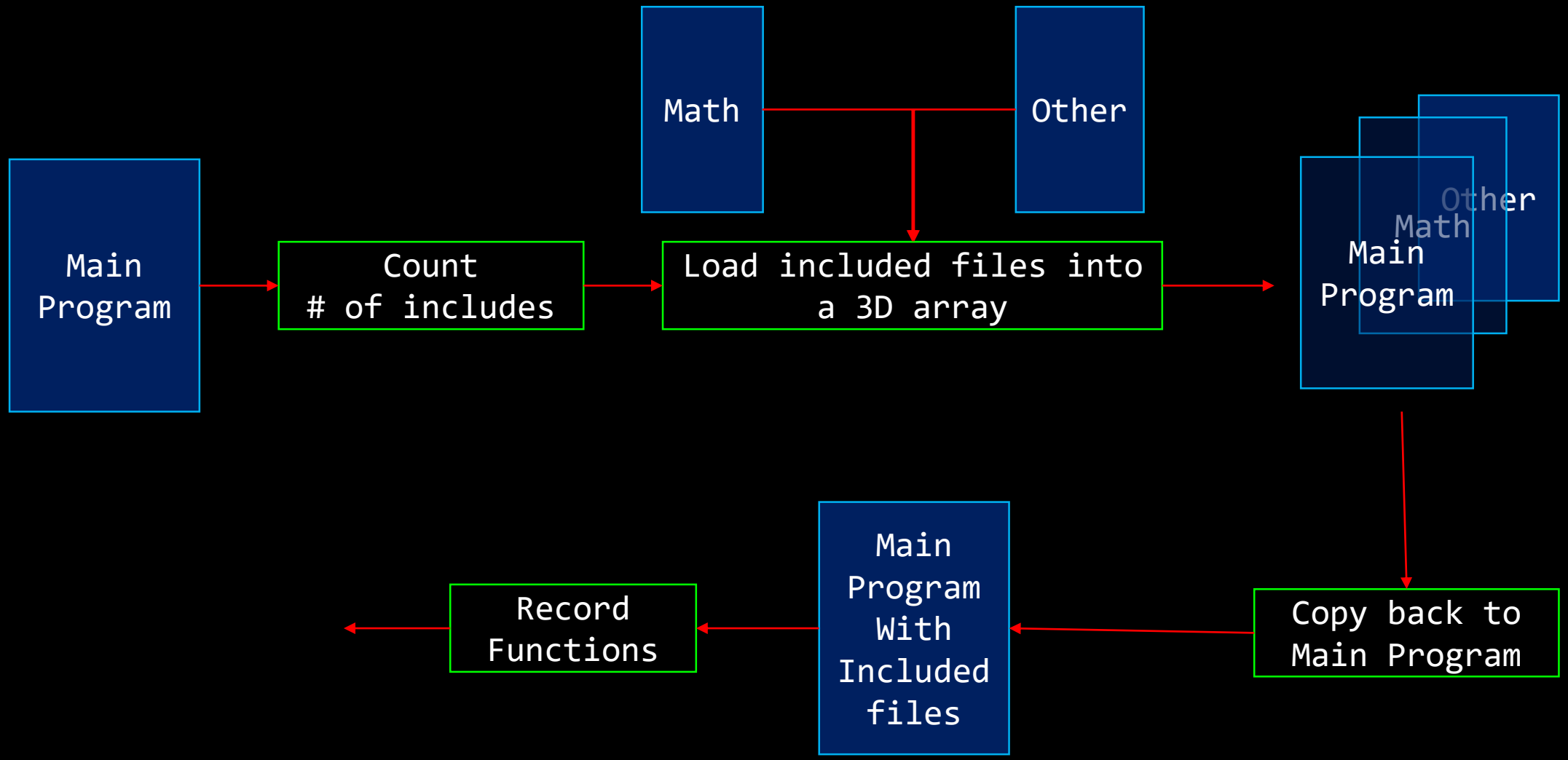
prog15.txt [X]

```
include "math.txt"

sub void main()
dim s int
dim i int
print "this is a edit from text editor"
print newline
print "Enter the number to not find factorial of: "
input i
s = factorial(i)
print i;"! = ";s;newline
endsub
```

1:1 Message

<F1 - Help> <F2 - Save> <F3 - Open> <F5 - Run> <F9 - Compile> <F10 - Exit>



How does ‘include’ work?

prog15.txt [X]

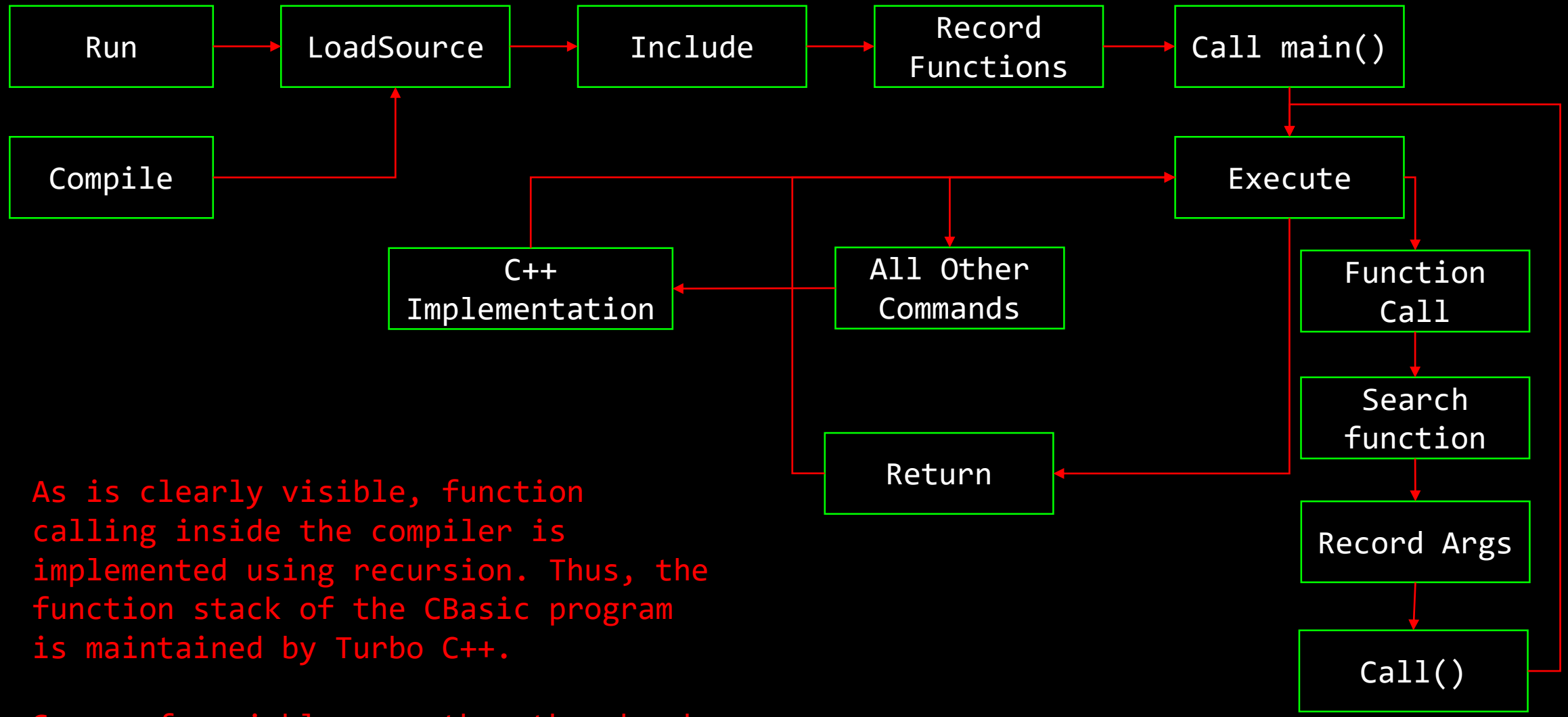
```
include "math.txt"

sub void main()
dim s int
dim i int
print "this is a edit from text editor"
print newline
print "Enter the number to not find factorial of: "
input i
s = factorial(i)
print i;"! = ";s;newline
endsub
```

1:1 Message

<F1 - Help> <F2 - Save> <F3 - Open> <F5 - Run> <F9 - Compile> <F10 - Exit>

Structure.cqb [x]



As is clearly visible, function calling inside the compiler is implemented using recursion. Thus, the function stack of the CBasic program is maintained by Turbo C++.

Scope of variables, on the other hand, is managed by us using nothing but a stack. (More on this later)

How does return statement work?

Return statement works with void* at its heart.

It stores the actual location of the variable being *returned* and returns it with a reference telling whether it is safe to use this value or not.

```
this is a edit from text editor
Enter the number to not find factorial of: 7
7! = 5040



Press any key to continue...
```

```
class function_class
{
  private:
        char return_type[10];
        char name[20];
        char arg_list[40];
        int start;
        int end;
        symbol_table priv_vars;
        friend class function_table;
  public:
        ...
};
```

```
class symbol_table
{

        private:
                entery* variables_list;
                //Discussed before^
                int state;
                int index;
                stack_class stack_1;
        public:
                ...

}
```

What happens when a function calls it self? Well, as all the functions are maintained in an array, there is nothing like “new function_class...”. I just shift the current executing line to that of the called function. Everything works perfectly until you realize the two “instances” of the functions share the same symbol_table which in turn messes all the base cases in recursive algorithms. Solution? Turn over ;)

```
class function_class
{
  private:
        char return_type[10];
        char name[20];
        char arg_list[40];
        int start;
        int end;
        symbol_table priv_vars;
        friend class function_table;
  public:
        ...
};
```

```
class symbol_table
{
        private:
                entery* variables_list;
                //Discussed before^
                int state;
                int index;
                stack_class stack_1;
        public:
                ...
}
```

Whenever a function is called, it's priv_vars are initialized which in turn do a *new entery[num_of_variables]*. We exploited this. Now, as soon as this happens, the address return by the above is pushed to a stack. Now, when the function "returns" after execution, it knows which *symbol_table* to point to using just a simple pop() operation. Cool, isn't it?

PS. Factorial works on it.

prog17.txt [X]

```
for x = 100 to 200 step 1
clear canvas
color x
car_x = x-80
pset(car_x,200)
draw "r35l35u30r50e30r40f30r50d30l35"
mid_x = x-15
pset(mid_x,200)
draw "r70"
wheel1_x = x-30
wheel2_x = x+70
circle(wheel1_x,200,15)
circle(wheel2_x,200,15)
delay 2
next x
close canvas
endsub
```

31:1 Message

<F1 - Help> <F2 - Save> <F3 - Open> <F5 - Run> <F9 - Compile> <F10 - Exit>

```
sub int foo(dim a int)
print a
return a
endsub

sub void main()
open canvas
rem bkcolor 8
dim x int
dim wheel1_x int
dim wheel2_x int
dim car_x int
dim mid_x int
pset(20,200)
for x = 100 to 200 step 1
clear canvas
color x
```

1:1 — Message

<F1 - Help> <F2 - Save> <F3 - Open> <F5 - Run> <F9 - Compile> <F10 - Exit>

How do we manage to create an array of **CB**asic variables of different data types?

MagicHere.cqb [x]

```
template <class datatype> class
variable
{
   private:
    char name[20];
    void* location;
    datatype value;
   public:
     ...
}
```

Demo

That's it.

(for now)

More information regarding all of the previous slides and other syntax delicacies can be found in help.txt and readme.txt. You are really encouraged to go through the source code to get a better understanding of this project.

~ The Void Club