

Table of Contents:

[Intro to MasC++](#)

[MasC++ command line arguments](#)

[Your First Console MasC++ program](#)

[Your first Windows MasC++ program](#)

[MasC++ Instruction Set](#)

[Compiling MasC++](#)

[Compiling under Windows](#)

[Compiling under Linux](#)

[The Source Code](#)

Intro to MasC++ by Jared Bruni (jaredbruni@gmail.com)

MasC++ is a cross platform object oriented scripting language. Its instruction set is based off Assembly Language/C++/ and Pascal. However it doesn't support complex expressions, but instead series of instructions, and streams. What can MasC++ be used for? Well its compatible with apache as cgi for windows and linux. You can recompile the source with your own function "callbacks" which can in turn call MasC++ procedures. So basically you can mix C++ and MasC++ to script up applications for the web, or write your own custom applications (games, windows appz, etc..). MasC++ has a default standard library which handles FILE I/O , Windows Application Programming and more. Its is in very alpha stages so the standard library at this time is very minimal. But soon it will be expanded. What are the standard library names? So far I have written console.msrc, file.msrc, and windows.msrc. You can include them into your application with the insert keyword (preprocessor directive) and they will be included into your app. MasC++ (finally) has a syntax analyzer will print out errors and the line that they appear on. I tried to think of every possible error I could come up with but there still might be some that will not be caught and will cause a runtime error. By Runtime error I mean a invalid instruction or the like will be called and the program will printout a debug error message preceded by **** to let you know theres something wrong. Thanks to Dustin Bowers for helping me test the syntax analyzer and catching alot of the problems there was initially with it. What doesn't the language support? As of right now it doesn't support instantiation of objects only inheritance of the objects from the "master" object. Instantiation will come in version 2.02. Also does not support multi-dimensional arrays or vectors. This is also a feature I am working on implementing. I am also thinking about including a expression parser for simplicity sake so people who are used to using complex expressions rather than instruction statements can do so. These are all things I am working on and will come in time. I have spent quite a long time working on this application and from the ashes of my brain have created 4 versions from scratch. Each time adding more features and improving the language. So what does the language support the following is a list of its features:

Full Compatibility with C++ with No C style casts, heavy use of STL

The ability to call MasC++ procedures from C++ code

The ability to call C++ functions from MasC++ code

A Symbol Table with easy access to variables via the [] operator for passing information to and from the code.

A Stack to pass variables to and from procedures

Instructions with one operand

Instructions with two operands

Instructions with stream operators

Mas Object code blocks which can contain virtual functions **implement** keyword and also external C++ functions **extern** keyword. Mas Objects have constructors/destructors which are called via the **uses** block in the **master** object. A constructor is defined by a begin statement followed by a block. Same for destructor except its a end statement followed by a block. Has a **var** block which contains the objects variables. Variable names are used by the name of the object preceded with the . operator Variable declarations are the following: **var&** name; (for integer type) **var#** floatname; (for floating point type) **var\$ stringname;** (for string type) variable declarations can have a initial value for the variable by using the = operator. So you could say **var\$** str = "";

master Object code block. The "master" object is the main object of the program its begin {} block is called when the program begins execution. And its end block is called when the programs execution terminates. It also contains a uses{} block for inheritance and multiple inheritance of other objects with virtual functions & procedures. comments are

statement;// this is a comment

A following is a example of a simple program using the MasC++ standard library

```
#insert "console.msrc";

master MyProg {

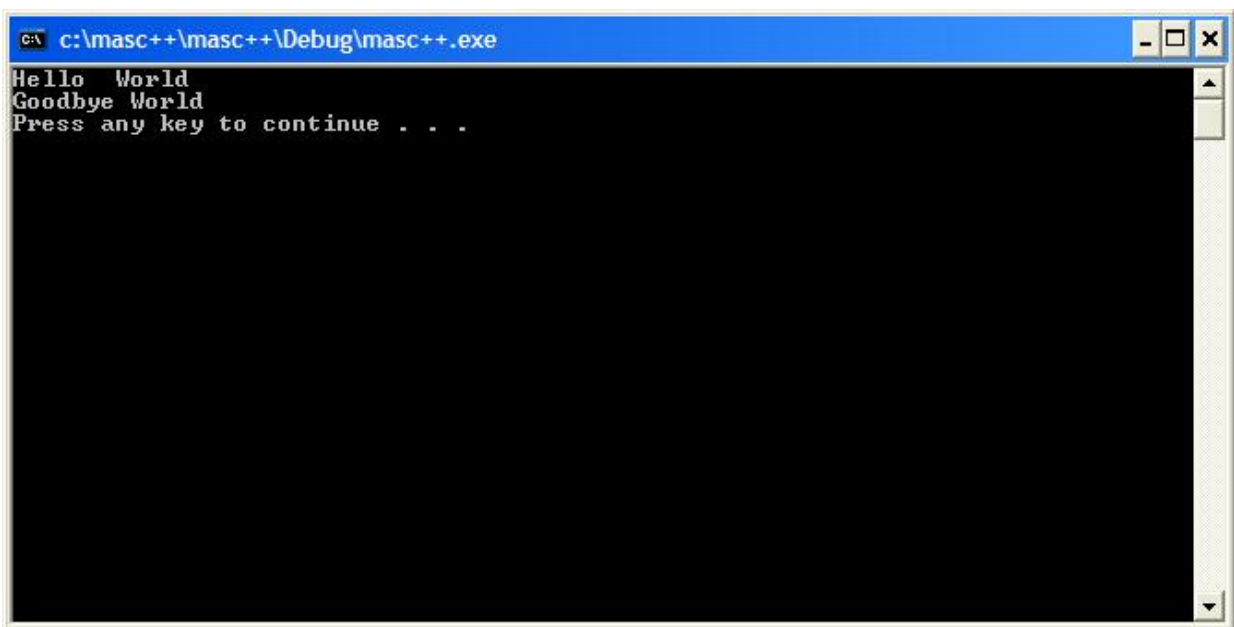
uses {console;}

var { var$ str = "Hello "; }

begin {
print << master.str << " World\n";
}

end {
print << "Goodbye World\n";
call console.pause;
}
}
```

The following program will give this output in windows:

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "c:\masc++\masc++\Debug\masc++.exe". The window has standard Windows window controls (minimize, maximize, close) on the right. The command prompt area is black with white text. The output of the program is displayed as follows:
Hello World
Goodbye World
Press any key to continue . . .
The cursor is positioned at the end of the third line.

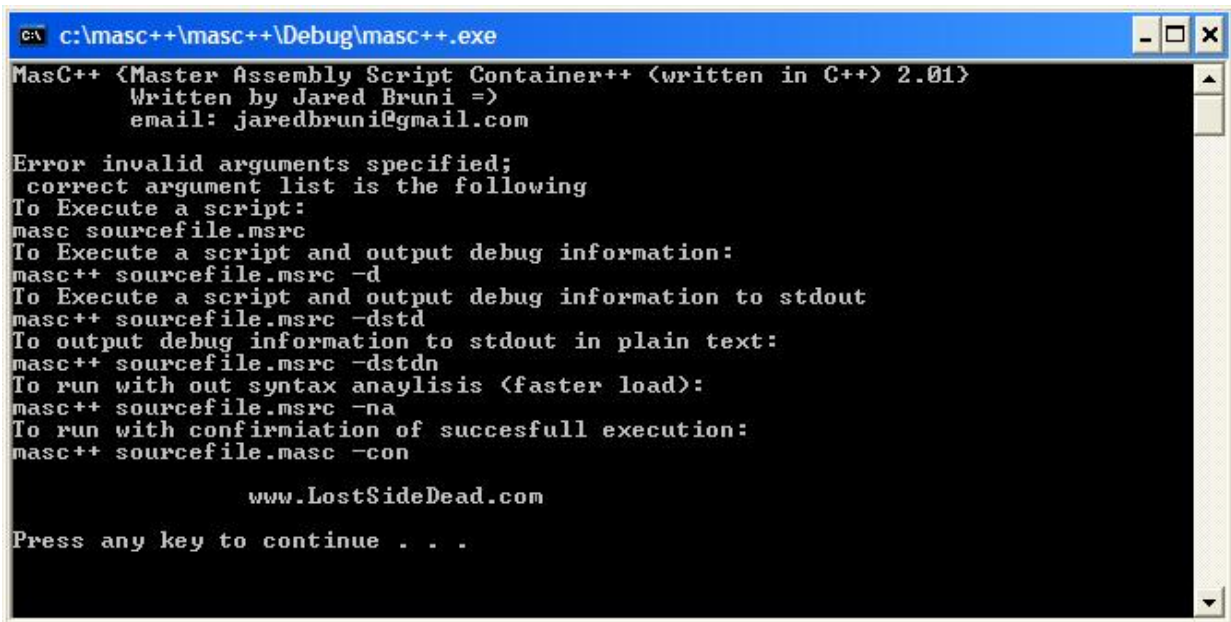
You may notice the variable name is master.str this is because the variable str belongs to the master object and can be called from any other object that the master object inherits. So say in the console mas object wants to use the master objects variable but doesn't know what the master object looks like. It can do so by using the master keyword preceded with a . operator.

That concludes this section of the documentation.

- **Jared Bruni** ⇒

[Next Step](#)

MasC++ has the following command line arguments: to invoke MasC++ do so from the command line.



```
c:\masc++\masc++\Debug\masc++.exe
MasC++ <Master Assembly Script Container++ <written in C++> 2.01>
Written by Jared Bruni =>
email: jaredbruni@gmail.com

Error invalid arguments specified;
correct argument list is the following
To Execute a script:
masc sourcefile.msrm
To Execute a script and output debug information:
masc++ sourcefile.msrm -d
To Execute a script and output debug information to stdout
masc++ sourcefile.msrm -dstd
To output debug information to stdout in plain text:
masc++ sourcefile.msrm -dstdn
To run with out syntax anaylisis <faster load>:
masc++ sourcefile.msrm -na
To run with confirmation of succesfull execution:
masc++ sourcefile.masc -con

www.LostSideDead.com

Press any key to continue . . .
```

To just run your script call it like this:

masc++ script.msrm

To run your script with debug information in HTML format do so with the following command.

masc++ script.msrm -d

This will output the intermediate code in a human readable HTML format for your to view in whatever your master object is named plus "_debug.html" So say you have master Jared {} defined it will output Jared_debug.html

The following command will output the debug information to the stdout (incase your running MasC++ from apache and want the debug information of your script in the browser when its called kinda like <?phpinfo()?>)

masc++ script.msrm -dstd

To print out the debug information in plain-text to do with the following command.

masc++ script.msrm -dstdn

To run the script without syntax anaylis (not recommend unless you have fully tested your script and are sure its working for faster load) to so with the following command

masc++ script.msrm -na

If your having problems with your program just coming up with a blank screen make sure its

successfully completed execution with the following command:

masc++ script.msrc -con

[Next Step](#) [Previous Step](#)

Your first MasC++ program:

```
#insert "console.msrc";

master MyProg {

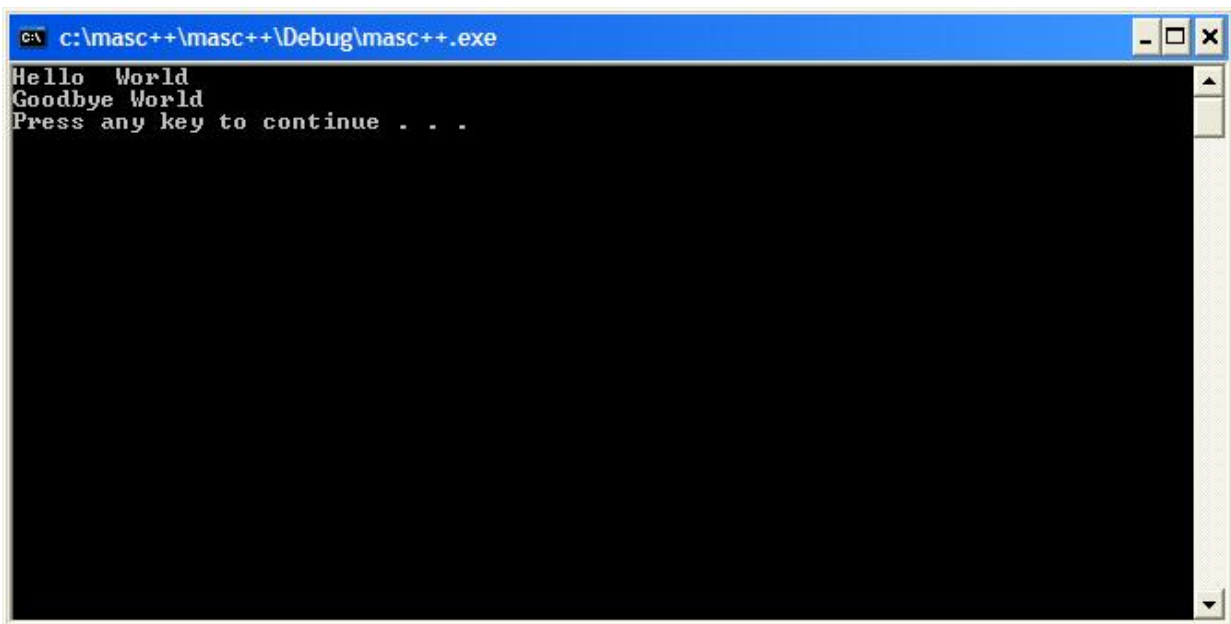
uses {console;}

var { var$ str = "Hello "; }

begin {
print << master.str << " World\n";
}

end {
print << "Goodbye World\n";
call console.pause;
}
}
```

Will produce the following output:

A screenshot of a Windows command prompt window. The title bar is blue and contains the text "C:\ c:\masc++\masc++\Debug\masc++.exe". The command prompt area is black with white text. The output of the program is displayed as follows:
Hello World
Goodbye World
Press any key to continue . . .
The cursor is positioned at the end of the third line.

To run the script execute:

masc++ myprog.msrc

The following is the HTML debug output for the first example:

Tokenized Code

File/Line	Byte Code	Symbols
proggie.msrc:2	#	#
proggie.msrc:2	Identifier	insert
proggie.msrc:2	String	console.msrc
proggie.msrc:2	;	;
proggie.msrc:3	Identifier	master
proggie.msrc:3	Identifier	MyProg
proggie.msrc:3	{	{
proggie.msrc:4	Identifier	uses
proggie.msrc:4	{	{
proggie.msrc:4	Identifier	console
proggie.msrc:4	;	;
proggie.msrc:4	}	}
proggie.msrc:6	Identifier	var
proggie.msrc:6	{	{
proggie.msrc:6	Identifier	var\$
proggie.msrc:6	Identifier	str
proggie.msrc:6	=	=
proggie.msrc:6	String	Hello
proggie.msrc:6	;	;
proggie.msrc:6	}	}
proggie.msrc:8	Identifier	begin
proggie.msrc:8	{	{
proggie.msrc:9	Identifier	print
proggie.msrc:9	<	<
proggie.msrc:9	Identifier	master.str
proggie.msrc:9	<	<
proggie.msrc:9	String	World
proggie.msrc:9	;	;
proggie.msrc:10	}	}

proggie.msrc:12	Identifier	end
proggie.msrc:12	{	{
proggie.msrc:13	Identifier	print
proggie.msrc:13	<	<
proggie.msrc:13	String	Goodbye World
proggie.msrc:13	;	;
proggie.msrc:14	Identifier	call
proggie.msrc:14	Identifier	console.pause
proggie.msrc:14	;	;
proggie.msrc:15	}	}
proggie.msrc:16	}	}
console.msrc:2	Identifier	insert
console.msrc:2	String	file.msrc
console.msrc:2	;	;
console.msrc:4	Identifier	mas
console.msrc:4	Identifier	console
console.msrc:4	{	{
console.msrc:6	Identifier	var
console.msrc:6	{	{
console.msrc:6	Identifier	var\$
console.msrc:6	Identifier	buffer
console.msrc:6	=	=
console.msrc:6	String	
console.msrc:6	;	;
console.msrc:6	}	}
console.msrc:8	Identifier	begin
console.msrc:8	{	{
console.msrc:8	}	}
console.msrc:10	Identifier	end
console.msrc:10	{	{
console.msrc:10	}	}
console.msrc:12	Identifier	extern
console.msrc:12	Identifier	proc
console.msrc:12	Identifier	clear

console.msrc:12	;	;
console.msrc:13	Identifier	extern
console.msrc:13	Identifier	proc
console.msrc:13	Identifier	pause
console.msrc:13	;	;
console.msrc:14	Identifier	extern
console.msrc:14	Identifier	proc
console.msrc:14	Identifier	input
console.msrc:14	;	;
console.msrc:15	Identifier	extern
console.msrc:15	Identifier	proc
console.msrc:15	Identifier	output
console.msrc:15	;	;
console.msrc:16	Identifier	extern
console.msrc:16	Identifier	proc
console.msrc:16	Identifier	flush
console.msrc:16	;	;
console.msrc:18	}	}
file.msrc:1	Identifier	mas
file.msrc:1	Identifier	file
file.msrc:1	{	{
file.msrc:2	Identifier	var
file.msrc:2	{	{
file.msrc:2	Identifier	var\$
file.msrc:2	Identifier	file_name
file.msrc:2	=	=
file.msrc:2	String	
file.msrc:2	;	;
file.msrc:2	Identifier	var\$
file.msrc:2	Identifier	buffer
file.msrc:2	=	=
file.msrc:2	String	
file.msrc:2	;	;
file.msrc:2	Identifier	var\$

file.msrc:2	Identifier	char
file.msrc:2	=	=
file.msrc:2	String	
file.msrc:2	;	;
file.msrc:2	}	}
file.msrc:4	Identifier	begin
file.msrc:4	{	{
file.msrc:4	}	}
file.msrc:5	Identifier	end
file.msrc:5	{	{
file.msrc:5	}	}
file.msrc:7	Identifier	extern
file.msrc:7	Identifier	proc
file.msrc:7	Identifier	read
file.msrc:7	;	;
file.msrc:8	Identifier	extern
file.msrc:8	Identifier	proc
file.msrc:8	Identifier	write
file.msrc:8	;	;
file.msrc:9	Identifier	extern
file.msrc:9	Identifier	proc
file.msrc:9	Identifier	bread
file.msrc:9	;	;
file.msrc:11	}	}

Intermediate Code

OpCode	Operand 1	Operand 2
Code Block (255)	master.begin	
print (32)	master.strÿ World Ÿ	
Code Block (255)	master.end	
print (32)	Goodbye World Ÿ	
call (30)	console.pause	
Code Block (255)	console.begin	
Code Block (255)		

	console.end	
Code Block (255)	file.begin	
Code Block (255)	file.end	

Symbol Table

String Name	Value	Type
console.buffer		String
file.buffer		String
file.char		String
file.file_name		String
master.str	Hello	String

External Callbacks

Procedure Name	Address
console.clear	00437A36
console.flush	00437220
console.input	00436C71
console.output	00436F4B
console.pause	00437B44
file.bread	004376E4
file.read	00437A8B
file.write	004365B9
window.create	004365AF
window.disp	004367DF
window.drawtext	00438094
window.loop	0043712B
window.msgbox	00437153
window.redraw	00437649
window.trans	00436BD1
window.waitmsg	00436424

[Next Step](#) [Previous Step](#)

The following is using the MasC++ windows standard library which is not yet complete it does contain basic functionality however. You can expand it on your own using external callbacks or by waiting for the next release. The MasC++ windows standard library source file is called windows.msrc simply include it with the insert keyword its layout is the following. Its important to know since it uses the implement keyword which stands for virtual functions that **MUST** be implemented for the script to execute.:

```
mas window {

var {
    var$ cls_name = ""; var$ win_title = ""; var& win_x = 0; var& win_y = 0; var& win_w = 640; var& win_h = 480;
    var& mouse_x = 0, var& mouse_y = 0, var& key = 0;
}

begin {}

end {}

extern proc create;
extern proc loop;
extern proc msgbox;
extern proc waitmsg;
extern proc trans;
extern proc disp;
extern proc drawtext;
extern proc redraw;
implement proc key_press;
implement proc paint;
implement proc mousedown;
implement proc mouseup;
implement proc mousemove;

}
```

This file includes functions for messagebox's windows Message Loop , creating a window drawing to the screen , redrawing the screen. Virtual functions for key's pressed, paint, mousedown, mouseup, and mousemove. It is still very primitive but you get the idea

The following source code will create a window, Loop for messages , when the escape key is pressed exit, and when a mouse is clicked/moved print message to the stdout.

```
insert "console.msrc";
insert "windows.msrc";

master MyWindow {
var {
    var& app_over = 0;
    var$ key_pressed = "";
    var$ mouse_clicked = "";
}

uses {window;}
begin {
print << "Initalizing Window...\n";
scat master.key_pressed << " Waiting for key... ";
scat window.cls_name << "Win_Cls";
scat window.win_title << "Press any Key";
mov window.win_x,0;
mov window.win_y,0;
mov window.win_w,640;
mov window.win_h,480;
call window.create;
win_loop:
call window.loop;
cmp master.app_over,0;
jne win_loop;
}
end {
print << "Destroying Window...";
```

```

push "Goodbye!";
push "Bye Bye!";
call window.msgbox;
}
proc key_press {
scat master.key_pressed << "You pressed the " << window.key << " Key!";
print << master.key_pressed << "\n";
push master.key_pressed;
push "Y0";
call window.redraw;
call window.msgbox;
cmp window.key,27;
je yep;
ret;
yep:
mov master.app_over,0;
push "You pressed Escape Were outta here..";
push "Laterz";
call window.msgbox;
}

proc paint {
push 0;
push 0;
push master.key_pressed;
call window.drawtext;
}

proc mousedown {
scat master.mouse_clicked << "Mouse Clicked at Position (" << window.mouse_x << " : " << window.mouse_y << ")";
print << master.mouse_clicked << "\n";
push "You clicked the mouse!";
push "Y0 y0 yo hommie Y()";
call window.msgbox;
}

proc mouseup {

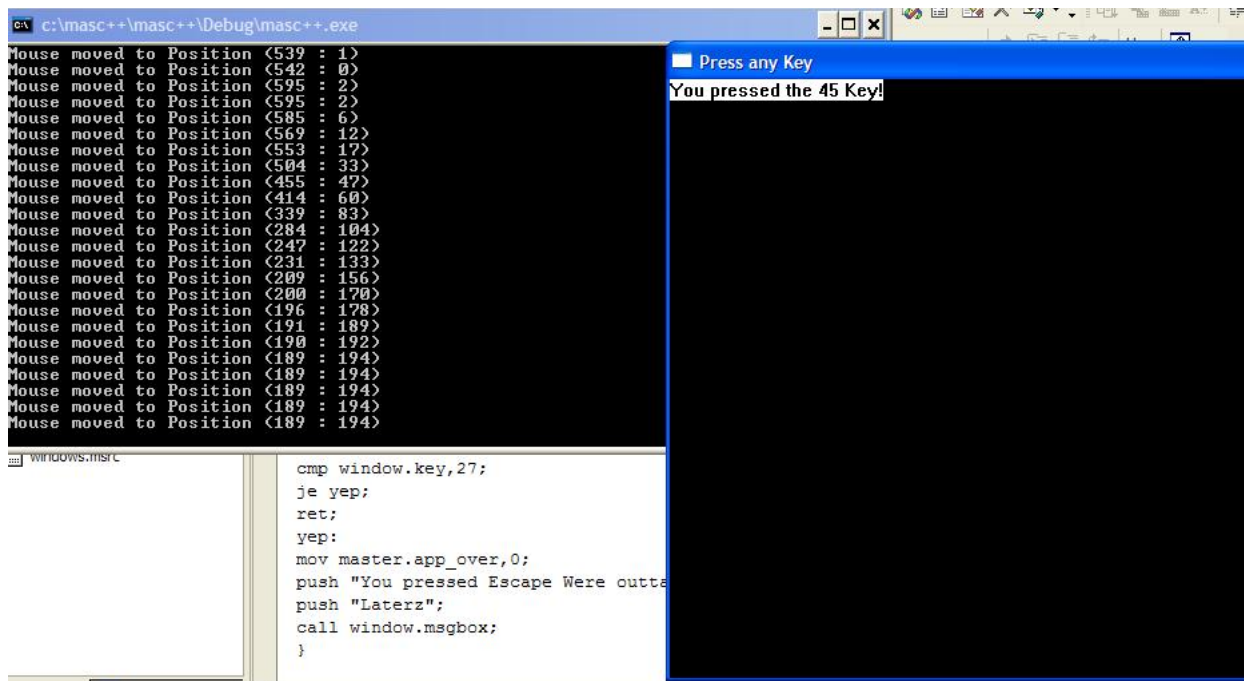
}

proc mousemove {
scat master.mouse_clicked << "Mouse moved to Position (" << window.mouse_x << " : " << window.mouse_y << ")";
print << master.mouse_clicked << "\n";
}

}

```

The following code will produce this output:



The following is the HTML Debug Output for the application

Tokenized Code

File/Line	Byte Code	Symbols
test.msrc:3	Identifier	insert
test.msrc:3	String	console.msrc
test.msrc:3	;	;
test.msrc:4	Identifier	insert
test.msrc:4	String	windows.msrc
test.msrc:4	;	;
test.msrc:7	Identifier	master
test.msrc:7	Identifier	MyWindow
test.msrc:7	{	{
test.msrc:8	Identifier	var
test.msrc:8	{	{
test.msrc:9	Identifier	var&
test.msrc:9	Identifier	app_over
test.msrc:9	=	=
test.msrc:9	Digit	0
test.msrc:9	;	;
test.msrc:10	Identifier	var\$
test.msrc:10	Identifier	key_pressed
test.msrc:10	=	=
test.msrc:10	String	
test.msrc:10	;	;

test.msrc:11	Identifier	var\$
test.msrc:11	Identifier	mouse_clicked
test.msrc:11	=	=
test.msrc:11	String	
test.msrc:11	;	;
test.msrc:12	}	}
test.msrc:14	Identifier	uses
test.msrc:14	{	{
test.msrc:14	Identifier	window
test.msrc:14	;	;
test.msrc:14	}	}
test.msrc:15	Identifier	begin
test.msrc:15	{	{
test.msrc:16	Identifier	print
test.msrc:16	<	<
test.msrc:16	String	Initializing Window...
test.msrc:16	;	;
test.msrc:17	Identifier	scat
test.msrc:17	Identifier	master.key_pressed
test.msrc:17	<	<
test.msrc:17	String	Waiting for key...
test.msrc:17	;	;
test.msrc:18	Identifier	scat
test.msrc:18	Identifier	window.cls_name
test.msrc:18	<	<
test.msrc:18	String	Win_Cls
test.msrc:18	;	;
test.msrc:19	Identifier	scat
test.msrc:19	Identifier	window.win_title
test.msrc:19	<	<
test.msrc:19	String	Press any Key
test.msrc:19	;	;
test.msrc:20	Identifier	mov
test.msrc:20	Identifier	window.win_x
test.msrc:20	,	,
test.msrc:20	Digit	0
test.msrc:20	;	;
test.msrc:21	Identifier	mov
test.msrc:21	Identifier	window.win_y
test.msrc:21	,	,
test.msrc:21	Digit	0
test.msrc:21	;	;
test.msrc:22	Identifier	mov
test.msrc:22	Identifier	window.win_w
test.msrc:22	,	,
test.msrc:22	Digit	640

test.msrc:22	;	;
test.msrc:23	Identifier	mov
test.msrc:23	Identifier	window.win_h
test.msrc:23	,	,
test.msrc:23	Digit	480
test.msrc:23	;	;
test.msrc:24	Identifier	call
test.msrc:24	Identifier	window.create
test.msrc:24	;	;
test.msrc:25	Code Label	win_loop
test.msrc:25	:	:
test.msrc:26	Identifier	call
test.msrc:26	Identifier	window.loop
test.msrc:26	;	;
test.msrc:27	Identifier	cmp
test.msrc:27	Identifier	master.app_over
test.msrc:27	,	,
test.msrc:27	Digit	0
test.msrc:27	;	;
test.msrc:28	Identifier	jne
test.msrc:28	Identifier	win_loop
test.msrc:28	;	;
test.msrc:29	}	}
test.msrc:30	Identifier	end
test.msrc:30	{	{
test.msrc:31	Identifier	print
test.msrc:31	<	<
test.msrc:31	String	Destroying Window...
test.msrc:31	;	;
test.msrc:32	Identifier	push
test.msrc:32	String	Goodbye!
test.msrc:32	;	;
test.msrc:33	Identifier	push
test.msrc:33	String	Bye Bye!
test.msrc:33	;	;
test.msrc:34	Identifier	call
test.msrc:34	Identifier	window.msgbox
test.msrc:34	;	;
test.msrc:35	}	}
test.msrc:36	Identifier	proc
test.msrc:36	Identifier	key_press
test.msrc:36	{	{
test.msrc:37	Identifier	scat
test.msrc:37	Identifier	master.key_pressed
test.msrc:37	<	<
test.msrc:37	String	You pressed the
test.msrc:37	<	<

test.msrc:37	Identifier	window.key
test.msrc:37	<	<
test.msrc:37	String	Key!
test.msrc:37	;	;
test.msrc:38	Identifier	print
test.msrc:38	<	<
test.msrc:38	Identifier	master.key_pressed
test.msrc:38	<	<
test.msrc:38	String	
test.msrc:38	;	;
test.msrc:39	Identifier	push
test.msrc:39	Identifier	master.key_pressed
test.msrc:39	;	;
test.msrc:40	Identifier	push
test.msrc:40	String	Y0
test.msrc:40	;	;
test.msrc:41	Identifier	call
test.msrc:41	Identifier	window.redraw
test.msrc:41	;	;
test.msrc:42	Identifier	call
test.msrc:42	Identifier	window.msgbox
test.msrc:42	;	;
test.msrc:43	Identifier	cmp
test.msrc:43	Identifier	window.key
test.msrc:43	,	,
test.msrc:43	Digit	27
test.msrc:43	;	;
test.msrc:44	Identifier	je
test.msrc:44	Identifier	yep
test.msrc:44	;	;
test.msrc:45	Identifier	ret
test.msrc:45	;	;
test.msrc:46	Code Label	yep
test.msrc:46	:	:
test.msrc:47	Identifier	mov
test.msrc:47	Identifier	master.app_over
test.msrc:47	,	,
test.msrc:47	Digit	0
test.msrc:47	;	;
test.msrc:48	Identifier	push
test.msrc:48	String	You pressed Escape Were outta here..
test.msrc:48	;	;
test.msrc:49	Identifier	push
test.msrc:49	String	Laterz
test.msrc:49	;	;
test.msrc:50	Identifier	call

test.msrc:50	Identifier	window.msgbox
test.msrc:50	;	;
test.msrc:51	}	}
test.msrc:53	Identifier	proc
test.msrc:53	Identifier	paint
test.msrc:53	{	{
test.msrc:54	Identifier	push
test.msrc:54	Digit	0
test.msrc:54	;	;
test.msrc:55	Identifier	push
test.msrc:55	Digit	0
test.msrc:55	;	;
test.msrc:56	Identifier	push
test.msrc:56	Identifier	master.key_pressed
test.msrc:56	;	;
test.msrc:57	Identifier	call
test.msrc:57	Identifier	window.drawtext
test.msrc:57	;	;
test.msrc:58	}	}
test.msrc:60	Identifier	proc
test.msrc:60	Identifier	mousedown
test.msrc:60	{	{
test.msrc:61	Identifier	scat
test.msrc:61	Identifier	master.mouse_clicked
test.msrc:61	<	<
test.msrc:61	String	Mouse Clicked at Position (
test.msrc:61	<	<
test.msrc:61	Identifier	window.mouse_x
test.msrc:61	<	<
test.msrc:61	String	:
test.msrc:61	<	<
test.msrc:61	Identifier	window.mouse_y
test.msrc:61	<	<
test.msrc:61	String)
test.msrc:61	;	;
test.msrc:62	Identifier	print
test.msrc:62	<	<
test.msrc:62	Identifier	master.mouse_clicked
test.msrc:62	<	<
test.msrc:62	String	
test.msrc:62	;	;
test.msrc:63	Identifier	push
test.msrc:63	String	You clicked the mouse!
test.msrc:63	;	;
test.msrc:64	Identifier	push
test.msrc:64	String	Y0 y0 yo hommie Y()
test.msrc:64	;	;

test.msrc:65	Identifier	call
test.msrc:65	Identifier	window.msgbox
test.msrc:65	;	;
test.msrc:66	}	}
test.msrc:68	Identifier	proc
test.msrc:68	Identifier	mouseup
test.msrc:68	{	{
test.msrc:70	}	}
test.msrc:72	Identifier	proc
test.msrc:72	Identifier	mousemove
test.msrc:72	{	{
test.msrc:73	Identifier	scat
test.msrc:73	Identifier	master.mouse_clicked
test.msrc:73	<	<
test.msrc:73	String	Mouse moved to Position (
test.msrc:73	<	<
test.msrc:73	Identifier	window.mouse_x
test.msrc:73	<	<
test.msrc:73	String	:
test.msrc:73	<	<
test.msrc:73	Identifier	window.mouse_y
test.msrc:73	<	<
test.msrc:73	String)
test.msrc:73	;	;
test.msrc:74	Identifier	print
test.msrc:74	<	<
test.msrc:74	Identifier	master.mouse_clicked
test.msrc:74	<	<
test.msrc:74	String	
test.msrc:74	;	;
test.msrc:75	}	}
test.msrc:77	}	}
console.msrc:2	Identifier	insert
console.msrc:2	String	file.msrc
console.msrc:2	;	;
console.msrc:4	Identifier	mas
console.msrc:4	Identifier	console
console.msrc:4	{	{
console.msrc:6	Identifier	var
console.msrc:6	{	{
console.msrc:6	Identifier	var\$
console.msrc:6	Identifier	buffer
console.msrc:6	=	=
console.msrc:6	String	
console.msrc:6	;	;
console.msrc:6	}	}

console.msrc:8	Identifier	begin
console.msrc:8	{	{
console.msrc:8	}	}
console.msrc:10	Identifier	end
console.msrc:10	{	{
console.msrc:10	}	}
console.msrc:12	Identifier	extern
console.msrc:12	Identifier	proc
console.msrc:12	Identifier	clear
console.msrc:12	;	;
console.msrc:13	Identifier	extern
console.msrc:13	Identifier	proc
console.msrc:13	Identifier	pause
console.msrc:13	;	;
console.msrc:14	Identifier	extern
console.msrc:14	Identifier	proc
console.msrc:14	Identifier	input
console.msrc:14	;	;
console.msrc:15	Identifier	extern
console.msrc:15	Identifier	proc
console.msrc:15	Identifier	output
console.msrc:15	;	;
console.msrc:16	Identifier	extern
console.msrc:16	Identifier	proc
console.msrc:16	Identifier	flush
console.msrc:16	;	;
console.msrc:18	}	}
windows.msrc:1	Identifier	mas
windows.msrc:1	Identifier	window
windows.msrc:1	{	{
windows.msrc:3	Identifier	var
windows.msrc:3	{	{
windows.msrc:4	Identifier	var\$
windows.msrc:4	Identifier	cls_name
windows.msrc:4	=	=
windows.msrc:4	String	
windows.msrc:4	;	;
windows.msrc:4	Identifier	var\$
windows.msrc:4	Identifier	win_title
windows.msrc:4	=	=
windows.msrc:4	String	
windows.msrc:4	;	;
windows.msrc:4	Identifier	var&
windows.msrc:4	Identifier	win_x
windows.msrc:4	=	=
windows.msrc:4	Digit	0
windows.msrc:4	;	;

windows.msrc:4	Identifier	var&
windows.msrc:4	Identifier	win_y
windows.msrc:4	=	=
windows.msrc:4	Digit	0
windows.msrc:4	;	;
windows.msrc:4	Identifier	var&
windows.msrc:4	Identifier	win_w
windows.msrc:4	=	=
windows.msrc:4	Digit	640
windows.msrc:4	;	;
windows.msrc:4	Identifier	var&
windows.msrc:4	Identifier	win_h
windows.msrc:4	=	=
windows.msrc:4	Digit	480
windows.msrc:4	;	;
windows.msrc:5	Identifier	var&
windows.msrc:5	Identifier	mouse_x
windows.msrc:5	=	=
windows.msrc:5	Digit	0
windows.msrc:5	,	,
windows.msrc:5	Identifier	var&
windows.msrc:5	Identifier	mouse_y
windows.msrc:5	=	=
windows.msrc:5	Digit	0
windows.msrc:5	,	,
windows.msrc:5	Identifier	var&
windows.msrc:5	Identifier	key
windows.msrc:5	=	=
windows.msrc:5	Digit	0
windows.msrc:5	;	;
windows.msrc:6	}	}
windows.msrc:8	Identifier	begin
windows.msrc:8	{	{
windows.msrc:8	}	}
windows.msrc:10	Identifier	end
windows.msrc:10	{	{
windows.msrc:10	}	}
windows.msrc:12	Identifier	extern
windows.msrc:12	Identifier	proc
windows.msrc:12	Identifier	create
windows.msrc:12	;	;
windows.msrc:13	Identifier	extern
windows.msrc:13	Identifier	proc
windows.msrc:13	Identifier	loop
windows.msrc:13	;	;
windows.msrc:14	Identifier	extern

windows.msrc:14	Identifier	proc
windows.msrc:14	Identifier	msgbox
windows.msrc:14	;	;
windows.msrc:15	Identifier	extern
windows.msrc:15	Identifier	proc
windows.msrc:15	Identifier	waitmsg
windows.msrc:15	;	;
windows.msrc:16	Identifier	extern
windows.msrc:16	Identifier	proc
windows.msrc:16	Identifier	trans
windows.msrc:16	;	;
windows.msrc:17	Identifier	extern
windows.msrc:17	Identifier	proc
windows.msrc:17	Identifier	disp
windows.msrc:17	;	;
windows.msrc:18	Identifier	extern
windows.msrc:18	Identifier	proc
windows.msrc:18	Identifier	drawtext
windows.msrc:18	;	;
windows.msrc:19	Identifier	extern
windows.msrc:19	Identifier	proc
windows.msrc:19	Identifier	redraw
windows.msrc:19	;	;
windows.msrc:20	Identifier	implement
windows.msrc:20	Identifier	proc
windows.msrc:20	Identifier	key_press
windows.msrc:20	;	;
windows.msrc:21	Identifier	implement
windows.msrc:21	Identifier	proc
windows.msrc:21	Identifier	paint
windows.msrc:21	;	;
windows.msrc:22	Identifier	implement
windows.msrc:22	Identifier	proc
windows.msrc:22	Identifier	mousedown
windows.msrc:22	;	;
windows.msrc:23	Identifier	implement
windows.msrc:23	Identifier	proc
windows.msrc:23	Identifier	mouseup
windows.msrc:23	;	;
windows.msrc:24	Identifier	implement
windows.msrc:24	Identifier	proc
windows.msrc:24	Identifier	mousemove
windows.msrc:24	;	;
windows.msrc:26	}	}
file.msrc:1	Identifier	mas
file.msrc:1	Identifier	file
file.msrc:1	{	{

file.msrc:2	Identifier	var
file.msrc:2	{	{
file.msrc:2	Identifier	var\$
file.msrc:2	Identifier	file_name
file.msrc:2	=	=
file.msrc:2	String	
file.msrc:2	;	;
file.msrc:2	Identifier	var\$
file.msrc:2	Identifier	buffer
file.msrc:2	=	=
file.msrc:2	String	
file.msrc:2	;	;
file.msrc:2	Identifier	var\$
file.msrc:2	Identifier	char
file.msrc:2	=	=
file.msrc:2	String	
file.msrc:2	;	;
file.msrc:2	}	}
file.msrc:4	Identifier	begin
file.msrc:4	{	{
file.msrc:4	}	}
file.msrc:5	Identifier	end
file.msrc:5	{	{
file.msrc:5	}	}
file.msrc:7	Identifier	extern
file.msrc:7	Identifier	proc
file.msrc:7	Identifier	read
file.msrc:7	;	;
file.msrc:8	Identifier	extern
file.msrc:8	Identifier	proc
file.msrc:8	Identifier	write
file.msrc:8	;	;
file.msrc:9	Identifier	extern
file.msrc:9	Identifier	proc
file.msrc:9	Identifier	bread
file.msrc:9	;	;
file.msrc:11	}	}

Intermediate Code

OpCode	Operand 1	Operand 2
Code Block (255)	master.begin	
print (32)	Initializing Window... ħ	
scat (31)	master.key_pressed	Waiting for key... ħ
scat (31)	window.cls_name	Win_Clsĥ
scat (31)	window.win_title	Press any Keyĥ
mov (7)	window.win_x	0
mov (7)	window.win_y	0

mov (7)	window.win_w	640
mov (7)	window.win_h	480
call (30)	window.create	
call (30)	window.loop	
cmp (20)	master.app_over	0
jne (22)	win_loop	
Code Block (255)	master.end	
print (32)	Destroying Window...ÿ	
push (16)	Goodbye!	
push (16)	Bye Bye!	
call (30)	window.msgbox	
Code Block (255)	master.key_press	
scat (31)	master.key_pressed	You pressed the ÿwindow.keyÿ Key!ÿ
print (32)	master.key_pressedÿ ÿ	
push (16)	master.key_pressed	
push (16)	Y0	
call (30)	window.redraw	
call (30)	window.msgbox	
cmp (20)	window.key	27
je (23)	yep	
ret (28)	;	
mov (7)	master.app_over	0
push (16)	You pressed Escape Were outta here..	
push (16)	Laterz	
call (30)	window.msgbox	
Code Block (255)	master.paint	
push (16)	0	
push (16)	0	
push (16)	master.key_pressed	
call (30)	window.drawtext	
Code Block (255)	master.mousedown	
scat (31)	master.mouse_clicked	Mouse Clicked at Position (ÿwindow.mouse_xÿ : ÿwindow.mouse_yÿ)ÿ
print (32)	master.mouse_clickedÿ ÿ	
push (16)	You clicked the mouse!	
push (16)	Y0 y0 yo hommie Y()	
call (30)	window.msgbox	
Code Block (255)	master.mouseup	
Code Block (255)	master.mousemove	
scat (31)	master.mouse_clicked	Mouse moved to Position (ÿwindow.mouse_xÿ : ÿwindow.mouse_yÿ)ÿ
print (32)	master.mouse_clickedÿ ÿ	
Code Block (255)	console.begin	
Code Block (255)	console.end	
Code Block (255)	window.begin	
Code Block (255)	window.end	
Code Block (255)	file.begin	
Code Block (255)	file.end	

Symbol Table

String Name	Value	Type
console.buffer		String
file.buffer		String
file.char		String
file.file_name		String
master.app_over	0	Integer
master.key_pressed		String
master.mouse_clicked		String
window.cls_name		String
window.key	0	Integer
window.mouse_x	0	Integer
window.mouse_y	0	Integer
window.win_h	480	Integer
window.win_title		String
window.win_w	640	Integer
window.win_x	0	Integer
window.win_y	0	Integer

External Callbacks

Procedure Name	Address
console.clear	00437A36
console.flush	00437220
console.input	00436C71
console.output	00436F4B
console.pause	00437B44
file.bread	004376E4
file.read	00437A8B
file.write	004365B9
window.create	004365AF
window.disp	004367DF
window.drawtext	00438094
window.key_press	CCCCCCCC
window.loop	0043712B
window.mousedown	CCCCCCCC
window.mousemove	CCCCCCCC
window.mouseup	CCCCCCCC
window.msgbox	00437153
window.paint	CCCCCCCC
window.redraw	00437649
window.trans	00436BD1
window.waitmsg	00436424

[Next Step](#) [Previous Step](#)

MasC++ supports the following instructions they are defined in masc.h and can easily add new instructions simply by adding them to the defined enumerated constants , and character array of keyword tokens in the correct order: They look like this:

```
enum TOK_TYPE { TOK_INSERT, TOK_MAIN, TOK_USES, TOK_MAS, TOK_VAR, TOK_BEGIN, TOK_END, TOK_MOV, TOK_ADD, TOK_SUB, TOK_MUL,
static char *tok_array[] = {"insert", "master", "uses", "mas", "var", "begin", "end", "mov", "add", "sub", "mul", "div", "or", "xor"
```

Each keyword/instruction will be explained statements are separeated with a semi-colon and comments are terminated from the point of // to the end of the line.

-Keywords:

insert "sourcefile.msrc";

- This keyword will insert one source file directly into another

master ObjectName {}

- This keyword is for creating the main object thats constructor is called first.

uses{object;object2;}

-The uses keyword is for inhertiance of **mas** objects it can inherit as many objects as resources allow and executes constructors/destructors from right to left unlike C++ which is left to right

mas ObjectName{}

-This is used for objects that contain procedures,external procedures,virtual procedures (implement) variables, and a constructor/destructor

var {}

-the **var** keyword is a block of code which contains a objects variable declerations

var&

- this stands for integer variable decleration.

var\$

-this stands for string variable declerations

var#

-this stands for double precesion floating point variables

* Note: Variable declerations can have initalization values the variable name followed by the equal sign (=) and a value.

begin

-The **begin** {} keyword is where a object starts its execution this cannot be left out however the **end** keyword is not nessicary

end

-The **end** {} keyword is for the objects destructor.

proc

-the proc keyword is for defining a member procedure for a specific **mas** or **master** object. if its a mas object the keyword can be preceeded by **extern** for external (functions in C++) or **implement** (which is virtual procedures to be implemented) keywords in the **master** object.

-Instructions

Double operand instructions syntax is the following:

instruction operand1,operand2;

Single operand instructions is the following:

instruction operand1;

The instructions:

*Double operand instructions

mov

-The mov instruction will move data from one variable to another. Or can move constant data into a variable:

add sub mul div xor or and

-the add/sub/mul/div/xor/or/and instructions will do the specific operation (the name of the instruction) on the operands.

cmp

-The cmp instruction (logical compare) will compare to variables and set the variable flag registers with the results of the operation for use with the looping instructions:

jne -(jump if not equal)

je -(jump if equal)

jl -(jump if less than)

jle -(jump if less than or equal to)

jg -(jump if greater than)

jge -(jump if greater than or equal to)

instructions for loops syntax is the following

start:

```
cmp master.var1 ,master.var2;
```

```
jne start;
```

*Single operand instructions:

inc dec not

- the inc/dec/not instructions will do the specific operation (name of the instruction) on the operands.

push

- push instruction will push variable or constant into the interpreters stack

pop

- pop instruction will pop variable from the interpreters stack into the variable specified by operand1

*Special Instructions (streams)

unlike the other instructions the **print** instruction takes two forms. First is like a regular instruction taking one operand. example:

```
print "hello world";
```

The second is using the stream operator <<

Example:

```
print << "x = " << master.x << " y = " << master.y << " equals x & y variables \n";
```

* Note instructions containing streams must be terminated with a semi-colon so they can expand multiple lines.

The other special instruction is **scat** which stands for string concatenation.

Its syntax is the following:

```
scat master.variable << "This is streamed into string variable. " << master.x << "\n";
```

than you can do whatever you want with the string including writing it to a file, passing it to a C++ function etc...

Well that concludes the documentation on the instructions / keywords. I know its very basic right now but this is my first language so give me some time

-Jared Bruni =>

[Previous](#)

First download the MasC++ source code package from
<http://www.lostsidedead.com/masc++.zip> Then extract it to a directory. Then choose
your platform and compile out with the following instructions:
for [windows](#)
for [linux/unix/osx/cygwin](#)

First download the MasC++ source code package from <http://www.lostsidedead.com/masc++.zip> Then extract it to a directory. Second click on the .vcproj file inside the extracted_directory/masc++_alpha1/masc++_vc/masc++/masc++.vcproj

Third if you wish to compile it for apache as a cgi program uncomment the line in main.cpp

```
//#define MASC_APACHE
```

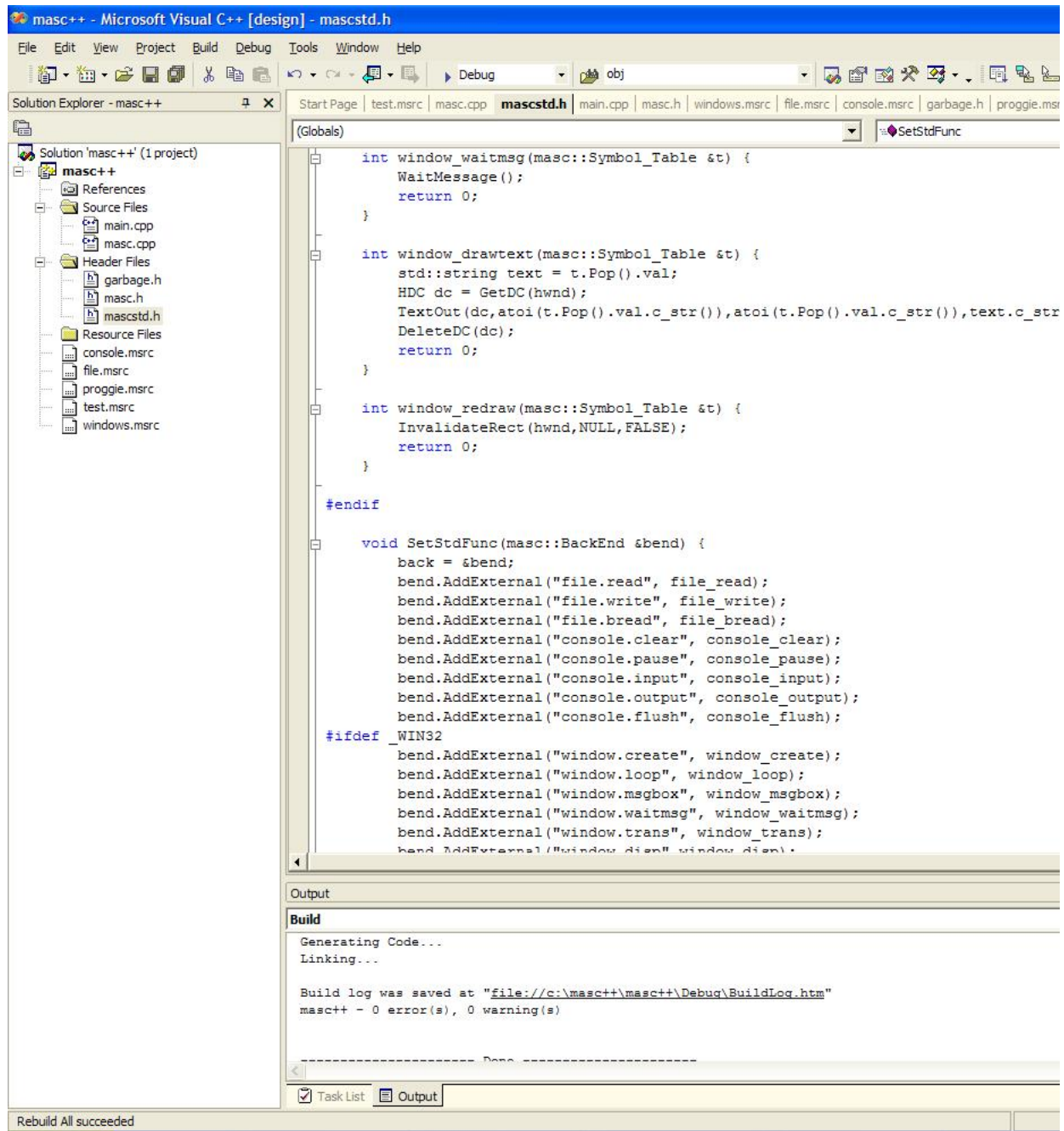
Third click The Build Menu Rebuild All (Build->Rebuild All) It should compile out with no warnings or errors. Second if you wish to add your own external functions to the stdlib

Open mascstd.h and add your functions in the correct format which is the following.

```
int functionname(masc::Symbol_table &t)
{
    return 0; // return 0 if succesfull other wise return 1 to break execution
}
```

Next add to the line that says bend.AddExternal the name of the mas object that contains the externed procedure and the procedure name seperated with a period (.), (example: "file.read") if your mas object is named file and the procedure name is read.The second paramter is the function name you wish to add.

After you do this Once agian rebuild the application. Heres a screen shot of Visual Studio .NET 2003 Enterprise Architect Succesfully compiling MasC++



Your all done. You can start using MasC++.

First download the MasC++ source code package from <http://www.lostsidedead.com/masc++.zip> Then extract it to a directory.

if you wish to compile it for apache as a cgi program uncomment the line in main.cpp

```
//#define MASC_APACHE
```

Third click The Build Menu Rebuild All (Build->Rebuild All) It should compile out with no warnings or errors. Second if you wish to add your own external functions to the stdlib

Open mascstd.h and add your functions in the correct format which is the following.

```
int functionname(masc::Symbol_table &t)
{
return 0; // return 0 if succesfull other wise return 1 to break execution
}
```

Next add to the line that says bend.AddExternal the name of the mas object that contains the externed procedure and the procedure name seperated with a period (.), (example: "file.read") if your mas object is named file and the procedure name is read. The second paramter is the function name you wish to add.

Open up the command prompt. Navigate to extracted_directory/masc++_alpha1/masc++_unix/
third type

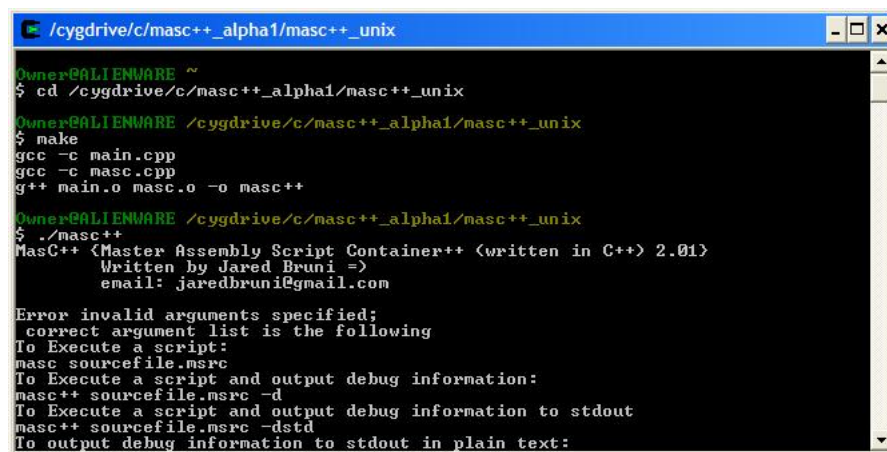
make

next type

./masc++

to test your build if you see a message print out like the following it worked!. you must have gnu make or a similar tool to use the Makefile.

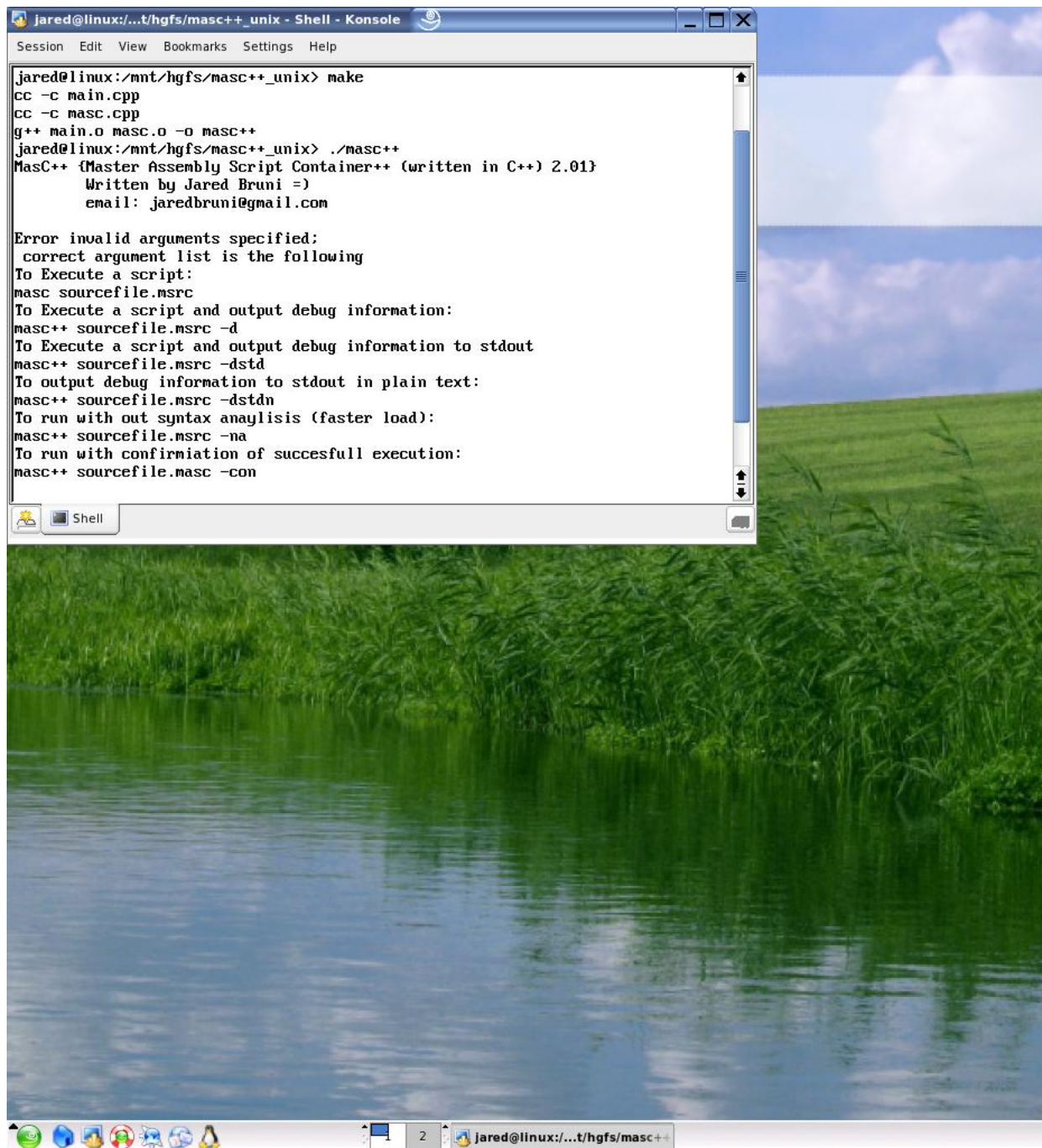
If your running cygwin you should see the following:



```
Owner@ALIENWARE ~
$ cd /cygdrive/c/masc++_alpha1/masc++_unix
Owner@ALIENWARE /cygdrive/c/masc++_alpha1/masc++_unix
$ make
gcc -c main.cpp
gcc -c masc.cpp
g++ main.o masc.o -o masc++
Owner@ALIENWARE /cygdrive/c/masc++_alpha1/masc++_unix
$ ./masc++
MasC++ (Master Assembly Script Container++ (written in C++) 2.01)
      Written by Jared Bruni =>
      email: jaredbruni@gmail.com

Error invalid arguments specified;
correct argument list is the following
To Execute a script:
masc sourcefile.msrc
To Execute a script and output debug information:
masc++ sourcefile.msrec -d
To Execute a script and output debug information to stdout
masc++ sourcefile.msrec -dstd
To output debug information to stdout in plain text:
```

If your running linux then you should see something like this:



You have succesfully compiled MasC++ !!!

The source code contains the following files:

[masc.h](#)

[mascstd.h](#)

[masc.cpp](#)

[garbage.h](#)

[main.cpp](#)

```

#ifndef MASC_H_
#define MASC_H_

#include<iostream>
#include<fstream>

#include<map>
#include<vector>
#include<string>
#include<list>
#include<cstdio>
#include "garbage.h"

namespace masc {

    const double version = 2.01;

    char *mysprintf(char *src, const char *sptr, ...);

    int myprintf(const char *src, ...);

    struct symb {

        std::string val,name;
        int type;
        symb() { val = name = ""; type = 0; }

    };

    class BackEnd;
    class Parser;

    class Symbol_Table {

    public:
        void Set(int type, std::string val, std::string name);

        symb &Get(std::string name);
        void DebugPrint() const;

        symb &operator[]( std::string str ) { return vars[str]; }

        symb &operator() ( size_t pos );
        symb operator--();
        std::list<symb>::iterator operator*() { return st.begin(); }

        void operator+=(const symb &s);
        std::list<symb> st;

        symb Pop();
        void PopAll();
        void Push(symb &s);

    protected:
        std::map<std::string, symb> vars;

        friend class BackEnd;
        friend class Parser;

    };

    class MFile {

    public:
        std::string cur_line;

        MFile(std::string s) {

            file = 0;
            Open(s);

        }

        ~MFile() {
            if(opened == true)

                file->close();
            delete file;
        }

        void Open(std::string s) {

```

```

        if(file && file->is_open())
            file->close();

        if(file) delete file;
        file = new std::fstream;

        file->open(s.c_str(),std::ios::in);
        if(!file->is_open()) { opened = false; return; }

        opened = true;
        file_name = s;
        curline = 0;
    }

    bool GetLine() {
        if(file->eof()) {
            file->close();

            return true;
        }
        std::getline(*file,cur_line);
        ++curline;

        return false;
    }

protected:
    std::fstream *file;

    bool opened;
    std::string file_name;
    int curline;
};

struct Token {
    unsigned char type;
    std::string token;

    int line_num;
    std::string source_file;
};

enum { IDENTIFIER = 1, DIGIT, STRING, CODELABEL };

class Parser : public MFile {
public:

    friend class BackEnd;
    Parser(std::string s);
    void Begin();

    void Parse(std::string str);
    void DebugTokens();
    Token &operator[](size_t pos) { if(pos >= 0 && pos < tok_vec.size()) return tok_vec[pos]; else return

    bool LexAnalyze();
    void StartParse(std::string str);
    inline bool pBounds(size_t pos) {
        if(pos >= 0 && pos < tok_vec.size()) {
            current_line = tok_vec[pos].line_num;

            current_file = tok_vec[pos].source_file;
            return true;
        }

        return false;
    }
    void ErrorMsg(const char *str, ...);

private:
    Token cur;
    int err,current_line;
    std::string current_file;

    std::vector<Token> tok_vec;
    std::vector<std::string> insert_vec;
};

struct Instruct {
    unsigned char op_code;
    std::string opl;
};

```

```

std::string op2;
Instruct(unsigned char op_code, std::string op1, std::string op2) {

    this->op_code = op_code;
    this->op1 = op1;

    this->op2 = op2;
}
Instruct() { op_code = 0; op1 = op2 = ""; }
};

struct External_Callback {
    int (*f) (Symbol_Table &t);

    std::string name, other_name;
    bool isdef, imp;
    External_Callback() {

        name = "";
        isdef = false;
        imp = false;
    }
};

enum TOK_TYPE { TOK_INSERT, TOK_MAIN, TOK_USES, TOK_MAS, TOK_VAR, TOK_BEGIN, TOK_END, TOK_MOV, TOK_ADD, TOK_SUB,

static char *tok_array[] = {"insert", "master", "uses", "mas", "var", "begin", "end", "mov", "add", "sub", "mul", "div", "

enum PROC_RETURN { PROC_CONT, PROC_JMP, PROC_RET };

struct Code_Label {
    std::string label_name;
    std::string proc;

    size_t pos;
    Code_Label(std::string label_name, std::string proc, size_t pos) {

        this->label_name = label_name;
        this->pos = pos;

        this->proc = proc;
    }
    Code_Label() {
        label_name = "";

        pos = 0;
        proc = "";
    }
};

class BackEnd {
    Parser *p;
    std::vector<std::vector<Instruct> > ins;

    std::string prog_name, current_obj;
    std::vector<std::string> uses;

    std::list<std::pair<size_t, size_t> > code;
public:

    Symbol_Table symbols;

    BackEnd(Parser *p) {
        this->p = p;

        memset(reg, 0, sizeof(reg));
        back = this;
    }
    ~BackEnd() {

        if(p)
            delete p;
    }
    void Convert();

    void Execute(bool analyze);

    inline void Debug() { std::cout << "Program Name: " << prog_name << '\n'; p->DebugTokens(); symbols.Debug()

    inline bool Bounds(size_t pos) { if(pos >= 0 && pos < p->tok_vec.size()) return true; return false; }

```

```

    size_t Find(std::string what);
    void AddExternal(std::string name, int (*f)(Symbol_Table &t));

    void DebugHTML(std::string src, bool sout);
    std::string GetName() { return prog_name; }

    void CallProcedure(std::string proc_name);
    static class BackEnd *back;

protected:
    void AddIncBlock(std::string block_name, size_t start);

    void DebugInc();
    PROC_RETURN ProcInc(size_t block, size_t &inc);

    bool IsVar(std::string &name);
    std::string byteToString(unsigned char type);

    std::string tokenTypeToString(TOK_TYPE t);
    size_t i, cur_pos;
    bool reg[4]; // flag registers

    std::string prev_op1, prev_op2;
    std::map<std::string, External_Callback> callb;

    std::map<std::string, Code_Label> labels;
    std::map<std::string, std::string> obj;

    friend class Parser;
};

inline int IdentToValue(std::string s);

inline BackEnd *GetBend() { return BackEnd::back; }

}

#endif

```

```

#ifndef _MASC_STD_H_
#define _MASC_STD_H_

#include "masc.h"
#include<sstream>

namespace mascstd {

    static masc::BackEnd *back;

    int file_read(masc::Symbol_Table &t) {

        std::fstream fin;
        fin.open(t.Get("file.file_name").val.c_str(),std::ios::in);

        t["file.buffer"].val = "";
        while(fin.is_open() && !fin.eof()) {

            std::string line;
            std::getline(fin,line);
            t["file.buffer"].val += line + "\n";

        }

        fin.close();
        return 0;
    }

    int file_write(masc::Symbol_Table &t) {

        std::fstream *fout = new std::fstream;

        fout->open(t["file.file_name"].val.c_str(),std::ios::out);

        if(fout->is_open()) {
            *fout << t["file.buffer"].val;

            fout->close();
        }
        else
            std::cout << "**** Error file " << t["file.file_name"].val << " Could not be opened!\n";

        delete fout;
        return 0;
    }

    int file_bread(masc::Symbol_Table &t) {

        std::fstream fin;
        fin.open(t.Get("file.file_name").val.c_str(),std::ios::in);

        t["file.char"].val = "";
        while(fin.is_open() && !fin.eof()) {

            unsigned char ch = 0;
            fin.read(reinterpret_cast<char*>(&ch),sizeof(ch));

            t["file.char"].val += ch;
        }
        std::cout << t["file.char"].val << "\n";

        fin.close();
        return 0;
    }

    int console_clear(masc::Symbol_Table &t) {

#ifdef _WIN32
        system("CLS");
#else
        system("clear");
#endif
        return 0;
    }

    int console_pause(masc::Symbol_Table &t) {

#ifdef _WIN32
        system("PAUSE");
#else
        getchar();
#endif
    }
}

```

```

#endif
        return 0;
    }

    int console_input(masc::Symbol_Table &t) {

#ifdef MASC_APACHE
        std::getline(std::cin, t["console.buffer"].val);
#endif

        return 0;
    }

    int console_output(masc::Symbol_Table &t) {

        std::cout << t["console.buffer"].val;
        return 0;
    }

    int console_flush(masc::Symbol_Table &t) {
        t["console.buffer"].val = "";

        return 0;
    }

#ifdef _WIN32
#include<windows.h>
#include<memory.h>

HDC cdc = 0;

HBITMAP hbm = 0;

LRESULT CALLBACK WndProc(HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam) {

    switch(msg) {
        case WM_DESTROY:
            PostQuitMessage(0);

            break;
        case WM_KEYDOWN:
            {
                std::ostringstream s;
                s << int(wParam);

                back->symbols.Set(masc::VAR_INTEGER,s.str(),"window.key");

                back->CallProcedure("master.key_press");
            }
            break;
        case WM_MOUSEMOVE:

        case WM_LBUTTONDOWN:
        case WM_LBUTTONUP:
            {
                POINT p = { LOWORD(lParam), HIWORD(lParam) };

                std::ostringstream s,ss;
                s << int(p.x);

                back->symbols.Set(masc::VAR_INTEGER,s.str(),"window.mouse_x");

                ss << int(p.y);
                back->symbols.Set(masc::VAR_INTEGER,ss.str(),"window.mouse_y");

                switch(msg)
                {
                    case WM_MOUSEMOVE:
                        back->CallProcedure("master.mousemove");

                        break;
                    case WM_LBUTTONDOWN:
                        back->CallProcedure("master.mousedown");

                        break;
                    case WM_LBUTTONUP:
                        back->CallProcedure("master.mouseup");

                        break;
                }
            }
    }
}

```



```

        }
        break;
    case WM_PAINT:
    {
        PAINTSTRUCT ps;

        HDC dc = BeginPaint(hwnd,&ps);
        RECT rc;
        GetClientRect(hwnd,&rc);

        cdc = CreateCompatibleDC(dc);
        hbm = CreateCompatibleBitmap(dc,rc.right,rc.bottom);

        SelectObject(cdc,hbm);
        back->CallProcedure("master.paint");
        BitBlt(dc,0,0,rc.right,rc.bottom,cdc,0,0,SRCCOPY);

        EndPaint(hwnd,&ps);
        DeleteDC(cdc);
        DeleteObject(hbm);
    }

    break;
default:
    return DefWindowProc(hwnd,msg,wParam,lParam);
}

return 0;
}
static HWND hwnd;

int window_create(masc::Symbol_Table &t) {

    WNDCLASSEX wc;
    wc.cbSize = sizeof(WNDCLASSEX);
    wc.cbClsExtra = 0;

    wc.cbWndExtra = 0;
    wc.hbrBackground = (HBRUSH) (COLOR_3DFACE+1);

    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);

    wc.hIconSm = LoadIcon(NULL,IDI_APPLICATION);
    wc.hInstance = GetModuleHandle(0);

    wc.lpfnWndProc = (WNDPROC) WndProc;
    wc.lpszClassName = t["window.cls_name"].val.c_str();

    wc.lpszMenuName = 0;
    wc.style = CS_HREDRAW | CS_VREDRAW;

    RegisterClassEx(&wc);
    hwnd = CreateWindow(t["window.cls_name"].val.c_str(),t["window.win_title"].val.c_str(),

        WS_OVERLAPPED | WS_SYSMENU | WS_MINIMIZEBOX ,atoi(t["window.win_x"].val.c_str()),

        atoi(t["window.win_y"].val.c_str()),atoi(t["window.win_w"].val.c_str()),atoi(t["window.win_h"].

        0,0,GetModuleHandle(0),0);
    ShowWindow(hwnd,SW_SHOW);

    UpdateWindow(hwnd);
    return 0;
}

static HANDLE thandle;

DWORD thread_id = 0;

BOOL CALLBACK MsgLoop(void *ptr) {

    MSG msg;
    while(GetMessage(&msg,0,0,0))
    {

        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return FALSE;
}

```

```

static MSG msg;
int window_loop(masc::Symbol_Table &t) {

    if(GetMessage(&msg,0,0,0)) {
        TranslateMessage(&msg);

        DispatchMessage(&msg);
        t["master.app_over"].val = "1";
    }

    else
        t["master.app_over"].val = "0";
    return 0;
}

int window_msgbox(masc::Symbol_Table &t) {
    std::string title = t.Pop().val,message = t.Pop().val;

    MessageBox(GetForegroundWindow(),message.c_str(),title.c_str(),MB_OK | MB_ICONINFORMATION);

    return 0;
}

int window_trans(masc::Symbol_Table &t) {

    return 0;
}

int window_disp(masc::Symbol_Table &t) {

    return 0;
}

int window_waitmsg(masc::Symbol_Table &t) {

    WaitMessage();
    return 0;
}

int window_drawtext(masc::Symbol_Table &t) {

    std::string text = t.Pop().val;
    HDC dc = GetDC(hwnd);

    TextOut(dc,atoi(t.Pop().val.c_str()),atoi(t.Pop().val.c_str()),text.c_str(),int(text.length()));

    DeleteDC(dc);
    return 0;
}

int window_redraw(masc::Symbol_Table &t) {

    InvalidateRect(hwnd,NULL,FALSE);
    return 0;
}

#endif

void SetStdFunc(masc::BackEnd &bend) {
    back = &bend;

    bend.AddExternal("file.read", file_read);
    bend.AddExternal("file.write", file_write);

    bend.AddExternal("file.bread", file_bread);
    bend.AddExternal("console.clear", console_clear);

    bend.AddExternal("console.pause", console_pause);
    bend.AddExternal("console.input", console_input);

    bend.AddExternal("console.output", console_output);
    bend.AddExternal("console.flush", console_flush);

#ifdef _WIN32
    bend.AddExternal("window.create", window_create);
    bend.AddExternal("window.loop", window_loop);

    bend.AddExternal("window.msgbox", window_msgbox);
    bend.AddExternal("window.waitmsg", window_waitmsg);

```

```
    bend.AddExternal("window.trans", window_trans);
    bend.AddExternal("window.disp", window_disp);

    bend.AddExternal("window.drawtext", window_drawtext);
    bend.AddExternal("window.redraw", window_redraw);

#endif

    }

}

#endif
```

```
/* NO its NOT RANDOM the way I format my CODE */
```

```
#include "masc.h"  
#include<sstream>
```

```
#include<stdarg.h>
```

```
namespace masc {  
  
    using std::cout;  
  
    using std::endl;  
  
    char *mysprintf(char *src, const char *sptr, ...) {  
  
        va_list l;  
        char *p, *sval = src, *ptr;  
  
        va_start(l, sptr);  
        for(p = const_cast<char*>(sptr); *p; p++) {  
  
            if(*p != '%') {  
                *sval = *p;  
                sval++;  
  
                continue;  
            }  
            else {  
                switch(*++p) {  
                    case 'd': {  
  
                        int ival = va_arg(l, int);  
                        std::ostringstream s;  
  
                        s << ival;  
                        std::string d = s.str();  
  
                        for(size_t i = 0; i < d.length(); i++)  
                            *sval = d[i], sval++;  
                    }  
  
                    break;  
                    case 's':  
                        for(ptr = va_arg(l, char*); *ptr; ptr++)  
                            *sval = *ptr, sval++;  
                        break;  
                    case 'f':  
                        {  
  
                            double dval = va_arg(l, double);  
  
                            std::ostringstream s;  
                            s << dval;  
                            std::string d = s.str();  
  
                            for(size_t i = 0; i < d.length(); i++)  
                                *sval = d[i], sval++;  
                        }  
  
                        break;  
                    default:  
                        *sval = *p;  
                        sval++;  
  
                        break;  
                }  
            }  
        }  
  
        *sval = 0;  
        return src;  
    }  
  
    int myprintf(const char *src, ...) {  
        va_list l;  
  
        char *p, *ptr;  
        va_start(l, src);
```

```

        for(p = const_cast<char*>(src); *p; p++) {
            if(*p != '%') {
                putchar(*p);
                continue;
            }
            else {
                switch(++p) {
                    case 'd': {
                        int ival = va_arg(l,int);
                        std::cout << ival;

                        break;
                    }
                    case 's':
                        for(ptr = va_arg(l, char*); *ptr; ptr++)
                            putchar(*ptr);
                        break;
                    case 'f':
                        {
                            double dval = va_arg(l,double);
                            std::cout << dval;
                        }
                        break;
                    default:
                        putchar(*p);
                        break;
                }
            }
        }
        return 0;
    }

void Symbol_Table::Set(int type, std::string val, std::string name) {
    vars[name].name = name;
    vars[name].val = val;

    vars[name].type = type;
}

symb &Symbol_Table::Get(std::string name) {
    return vars[name];
}

void Symbol_Table::DebugPrint() const {
    std::map<std::string,symb>::const_iterator i;
    for(i = vars.begin(); i != vars.end(); i++)
        std::cout << "Variable name: " << i->first << " Variable value: " << i->second.val << '\n';
}

symb &Symbol_Table::operator() ( size_t pos ) {
    std::list<symb>::iterator i = st.begin();

    for(size_t z = 0; z < pos; z++)
        i++;
    return *i;
}

symb Symbol_Table::Pop() {
    if(!st.empty()) {
        symb p = st.back();

        st.pop_back();
        return p;
    }
    return symb();
}

```

```

void Symbol_Table::operator+=(const symb &s) {
    st.push_back(s);
}

symb Symbol_Table::operator--() {
    return Pop();
}

void Symbol_Table::PopAll() {
    while(!st.empty())
        st.pop_back();
}

void Symbol_Table::Push(symb &s) {
    st.push_back(s);
}

Parser::Parser(std::string s) : MFile(s) , err(0) , current_line(0) {

    if(opened == true) {
        GetLine();
        Begin();
    }

    else {
        current_line = 0;
        ErrorMessage("Couldnt open file (%s)\n", s.c_str());

        return;
    }

}

void Parser::StartParse(std::string str) {

    opened = false;
    Open(str);
    if(opened == true) {

        GetLine();
        Begin();
    }
    else {
        current_line = 0;

        ErrorMessage("Couldnt open file (%s)\n", str.c_str());
        return;
    }

}

void Parser::Begin() {

    Parse(cur_line);
    if(!GetLine())
        Begin();
}

void Parser::Parse(std::string str) {
    std::string::iterator p;

    std::string cur_tok = "";
    cur.line_num = curline;

    cur.source_file = file_name;
    cur.token = "";

    str += "\n";
    if(str.find("insert") != -1 && str.find('\n') != -1 && str.find(";") != -1) {

        std::string insert_file;
        insert_file = str.substr(str.find('\n')+1, str.length());

        if(insert_file.find('\n') != -1)
            insert_file = insert_file.substr(0, insert_file.find('\n'));

        insert_vec.push_back(insert_file);
    }
}

```

```

for(p = str.begin(); p != str.end(); p++) {
    if(*p == '/')
        return;

    if(isalpha(*p)) {
        cur_tok = "";
        while(*p != '\n' && (isalpha(*p) || isdigit(*p) || *p == '_' || *p == '$' || *p == '&'))
            cur_tok += *p;
        p++;
    }
    if(*p != ':')
        cur.type = IDENTIFIER;
    else
        cur.type = CODELABEL;

    cur.token = cur_tok;
    tok_vec.push_back(cur);

    cur_tok = "";
}
else if(isdigit(*p)) {
    cur_tok = "";

    while((isdigit(*p) || *p == '.') && p != str.end()) {
        cur_tok += *p;
        p++;
    }
    cur.token = cur_tok;

    cur.type = DIGIT;
    tok_vec.push_back(cur);
}

else if(*p == '\\') {
    p++;
    while(*p != '\\') && p != str.end()) {
        if(*p != '\\') {
            cur_tok += *p;
            p++;
        }
        else
            if(*p == '\\')
            {
                p++;

                if(*p == '\\')
                    cur_tok += *p, p++;

                else if(*p == 'n')
                    cur_tok += '\\n', p++;
            }
    }

    cur.token = cur_tok;
    cur.type = STRING;

    tok_vec.push_back(cur);
    cur_tok = "";
}

switch(*p) {
    case '{':
    case '}':

    case ';':
    case ',':
    case '=':

    case '<':
    case '~':
    case '\\':

```

```

        case '@':
        case '!':
        case '#':

        case '$':
        case '%':
        case '^':

        case '&':
        case '*':
        case '(':

        case ')':
        case '_':
        case '+':

        case '/':
        case '\\':
        case '.':

        case '>':
        case ':':
            if(*(p+1) == '<') {

                p++;
            }
            cur.token = *p;
            cur.type = *p;
            tok_vec.push_back(cur);

            cur_tok = "";
            break;
    }
}

void Parser::DebugTokens() {
    for(unsigned int i = 0; i < tok_vec.size(); i++)
        cout << "token (type:" << int(tok_vec[i].type) << ") " << tok_vec[i].token << endl;
}

void Parser::ErrorMsg(const char *str, ...) {
    va_list l;

    char error_msg[5000];
    va_start(l,str);
    vsprintf(const_cast<char*>(error_msg),str,l);

    va_end(l);
    cout << "**** error#: " << (++err) << " " << error_msg << " \n\t->line(" << current_line << ").file(" <
}

bool Parser::LexAnalyze() {
    int masteronLucy = 0;

    std::string obj_name = "";

    for(unsigned int z = 0; z < tok_vec.size(); z++) {
        current_line = tok_vec[z].line_num;
        current_file = tok_vec[z].source_file;

        int vid = IdentToValue(tok_vec[z].token);

        if(vid == TOK_USES) {
            std::map<std::string,masc::External_Callback>::iterator itz;

            for(itz = GetBend()->callb.begin(); itz != GetBend()->callb.end(); itz++) {
                if(itz->second.imp == true && itz->second.isdef == false) {

                    std::map<std::string,std::string>::iterator it;
                    bool found_it = false;

                    for(it = GetBend()->obj.begin(); it != GetBend()->obj.end(); it++) {
                        if(GetBend()->Find(it->second + "." + itz->second.other_name) !

```



```

        found_it = true;
        break;
    }

    }
    if(found_it == false) {
        ErrorMsg(" (fatal) procedure (%s) not implemented yet is define
        return false;
    }
}

}

else if(tok_vec[z].type == '{' && pBounds(z-4)) {
    int zid = IdentToValue(tok_vec[z-1].token), zzid = IdentToValue(tok_vec[z-2].token);
    if(zid == -1 && zzid != TOK_PROC && zzid != TOK_EXTERN && tok_vec[z-4].type != '}' && G
        ErrorMsg("Invalid object/uses/procedure name on opening block you put (%s)", to
        z++;
        continue;
    }
}
else if(pBounds(z-1) && (vid == TOK_USES || vid == TOK_BEGIN || vid == TOK_END || vid == TOK_PR
    int rid = IdentToValue(tok_vec[z-1].token);
    if(rid == -1 && tok_vec[z-1].type != ';' && tok_vec[z-1].type != '}') {
        ErrorMsg("Error bad value before keyword (%s) i found (%s)", tok_array[vid], to
        z++;
        continue;
    }
    else {
        if((vid == TOK_MAS || vid == TOK_MAIN) && pBounds(z+1) && tok_vec[z+1].type ==
            obj_name = tok_vec[z+1].token;
            continue;
        }
    }
}

else if(vid == TOK_IMPLEMENT) {
    if(pBounds(++z) && (vid = IdentToValue(tok_vec[z].token)) == TOK_PROC && pBounds(++z) &
        if(pBounds(z-1) && GetBend()->callb[obj_name + "." + tok_vec[z-1].token].imp ==
            ErrorMsg("implemented procedure is missing from object (%s) ", "");
            continue;
        }
    }
    else {
        ErrorMsg(" there is a problem with the implement keyword probaly you forgot a s
    }
    z++;
}

}
current_line = 0;
int ID = 0;
for(unsigned int i = 0; i < tok_vec.size(); i++) {
    current_line = tok_vec[i].line_num;
    current_file = tok_vec[i].source_file;
    switch(tok_vec[i].type) {
        case IDENTIFIER:

```

```

{
    int id = IdentToValue(tok_vec[i].token);

    switch(id)
    {
    case TOK_USES:
        {
            if(pBounds(i+3) && (tok_vec[i+1].type == '{')) {

                i+=2;
                while(pBounds(i) && tok_vec[i].type != '}') {

                    if(pBounds(i) && tok_vec[i].type != ';')

                        ErrorMsg("Error uses block is t
                                i++;

                                continue;

                                }
                                i++;
                            }
                        }
                    }
                break;
            case TOK_INSERT:
                {
                    i++;
                    if(pBounds(i)) {
                        std::fstream f;

                        f.open(tok_vec[i].token.c_str(), std::ios::in);

                        if(f.is_open())
                        { f.close(); continue; }
                        else

                            f.close(), ErrorMsg("insert not includi

                            continue;

                        }
                    }
                break;
            case TOK_MAS:
            case TOK_MAIN:
                {

                    if(id == TOK_MAIN) {
                        masteronLucy++;
                        if(masteronLucy >= 2) {

                            ErrorMsg("(fatal) more than one main ob
                            return false;

                            }
                        }
                    if(pBounds(++i)) {

                        int tid = IdentToValue(tok_vec[i].token);
                        if(tid >= TOK_INSERT && tid <= VAR_FLOAT) {

                            ErrorMsg("mas(ter) keyword found anothe
                            continue;

                            }

                            else if(tok_vec[i].type == DIGIT || tok_vec[i].

                                ErrorMsg("mas(ter) keyword found a digi
                                continue;

                                }

                                }

                                ID++;

                                }
                                break;
                            case TOK_VAR:
                                {
                                    ID++;

                                    i++;
                                    if(pBounds(i) && pBounds(i+1)) {

```

```

        if(tok_vec[i].type != '{') {
            ErrorMsg("(fatal) variable code block m
                return false;
        }
    }
}

while(pBounds(++i) && tok_vec[i].type != '}') {
    current_line = tok_vec[i].line_num;
    current_file = tok_vec[i].source_file;

    int zid = IdentToValue(tok_vec[i].token);
    if(zid >= TOK_USES && zid <= TOK_EXTERN) {
        ErrorMsg("(fatal) found keyword (%s) in
            return false;
    }

    if(zid >= VAR_INTEGER && zid <= VAR_FLOAT) {
        if(pBounds(++i)) {
            if(tok_vec[i].type != IDENTIFIE
                ErrorMsg(" found a stri
                    continue;
            }

            else if(pBounds(++i) && tok_vec
                ErrorMsg(" found %s ins
                    continue;
            }

            else if(pBounds(i) && tok_vec[i]
                ErrorMsg(" looking for
                    continue;
            }

            else if(pBounds(i) && (zid == V
                ErrorMsg(" numeric vari
                    continue;
            }

            else if(pBounds(i) && (zid == V
                ErrorMsg(" string varia
                    continue;
            }
        }
    }

    else if(GetBend()->symbols.vars[tok_vec[i].toke
        ErrorMsg("Error invalid variable decler
            continue;
    }

}

if(tok_vec[i].type == '{')
    continue;

else {
    ErrorMsg("(fatal) variable code block missing c
        return false;
}

}

break;
case TOK_EXTERN:
case TOK_BEGIN:
case TOK_END:
case TOK_PROC:
{
    ID++;

```

```

if(id == TOK_EXTERN && pBounds(i+2) && tok_vec[i+1].typ
    ErrorMsg("external function (c++) call dosent h
    i+=2;
    continue;
}
if(id == TOK_BEGIN && pBounds(i+1) && tok_vec[i+1].type
    ErrorMsg("looking for { on begin statement inst
    continue;
}
if(id == TOK_END && pBounds(i+1) && tok_vec[i+1].type !
    ErrorMsg("looking for { on end statement instea
    continue;
}
if(id == TOK_PROC && pBounds(i+2) && (tok_vec[i+2].type
    ErrorMsg("looking for (normal) { or (extern) ;
    continue;
}
if(id == TOK_PROC && pBounds(++i) && tok_vec[i].type !=
    ErrorMsg(" procedure has string or digit or sym
    continue;
}
i ++;
while(pBounds(++i) && (tok_vec[i].type != '}')) {
    int prev_i = i;
    current_line = tok_vec[i].line_num;
    current_file = tok_vec[i].source_file;
    int qid = IdentToValue(tok_vec[i].token);
    if(qid == -1 && id != TOK_EXTERN && tok_vec[i].
        ErrorMsg("Invalid instruction (%s)", to
        continue;
    }
    if(qid >= TOK_MOV && qid <= TOK_PRINT) {
        if(pBounds(i-1) && tok_vec[i-1].type !=
            ErrorMsg(" Error should have fo
            continue;
        }
        switch(qid) {
        case TOK_MOV:
        case TOK_CMP:
        case TOK_ADD:
        case TOK_SUB:
        case TOK_MUL:
        case TOK_DIV:
        case TOK_OR:
        case TOK_XOR:
        case TOK_AND:
        {
            if(pBounds(++i) && tok_
                ErrorMsg(" look

```

```

        continue;
    }
    else if(pBounds(i) && t
        errorMsg(" look
        continue;
    }
    else if(pBounds(i+2) &&
        errorMsg(" expe
        continue;
    }
    else if(pBounds(++i) &&
        errorMsg(" look
        continue;
    }
    else if(pBounds(++i) &&
        errorMsg(" look
        continue;
    }
    }
    break;
case TOK_INC:
case TOK_DEC:

case TOK_NOT:
case TOK_JMP:
case TOK_JNE:

case TOK_JE:
case TOK_JL:
case TOK_JG:

case TOK_JGE:
case TOK_JLE:
case TOK_POP:
    {
        if(!(qid >= TOK_JMP &&
            errorMsg(" look
            continue;
        }
        else if(!(qid >= TOK_J
            errorMsg(" look
            continue;
        }
        else if(qid >= TOK_JMP
            errorMsg(" look
            continue;
        }
        else if(qid >= TOK_JMP
            errorMsg(" coul
            continue;
        }
    }
    break;
case TOK_CALL:
    {
        if(pBounds(++i) && tok_
            errorMsg("call
            continue;
    }

```

```

    }
}
break;
case TOK_PRINT:
case TOK_SCAT:
{
    if(qid == TOK_PRINT &&
        if(pBounds(i+1)
            ErrorMs
            continu
        }
        while(pBounds(+
            if(tok_

        }
        else if

    }
}
}
else if(qid == TOK_PRIN
    ErrorMsg("print
    continue;
}
else if(qid == TOK_SCAT
    ErrorMsg(" on s
    continue;
}
else if(qid == TOK_SCAT
    ErrorMsg(" on s
    continue;
}
else if(qid == TOK_SCAT
    while(pBounds(+
        if(tok_

    }
}
}
break;
}
    if(pBounds(i+1) && tok_vec[i+1].type ==
}
else if(id != TOK_EXTERN && qid == TOK_PROC ||
    ErrorMsg("nested procedures / blocks no
    continue;
}
}
if(tok_vec[i].type != '}') {

```

```

        ErrorMsg(" code block is missing closing brace
        continue;
    }

    }
    break;
    default:
    {
        }
    }
}
break;
}
}
if(err == 0 && ID >= 4) // make sure its a valid program before execution..
    //this can be disabled for speed but its not advised

return true;
return false;
}

void BackEnd::Convert() {
    for(size_t z = 0; z < p->insert_vec.size(); z++)
    {
        p->StartParse(p->insert_vec[z]);
    }

    if(!p->insert_vec.empty()) p->insert_vec.clear();

    std::string current_proc = "";
    for(size_t i = 0; i < p->tok_vec.size(); i++) {
        switch(p->tok_vec[i].type) {
            case IDENTIFIER:
            {
                int id = IdentToValue(p->tok_vec[i].token);

                if(id != -1) {
                    switch(id)
                    {
                        case VAR_INTEGER:

                            i++;
                            if(Bounds(i)) {
                                if(Bounds(i+1)) {

                                    if(p->tok_vec[i].token.find '[' != -1
                                        std::string num = p->tok_vec[i]
                                        std::string name = p->tok_vec[i]
                                        for(size_t p = 0; p < size_t(at
                                            std::ostringstream s;
                                            s << current_obj + "."
                                            symbols.Set(VAR_INTEGER
                                        }
                                    }
                                }
                                else
                                    if(p->tok_vec[i+1].type == '=')
                                        if(Bounds(i+2))
                                            symbols.Set(VAR
                                        }
                                    else {
                                        symbols.Set(VAR_INTEGER
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
}

break;
case VAR_STRING:
    i++;
    if(Bounds(i)) {

        if(Bounds(i+1)) {
            if(p->tok_vec[i].token.find('[') != -1

                std::string num = p->tok_vec[i]

                std::string name = p->tok_vec[i]

                for(size_t p = 0; p < size_t(at

                    std::ostringstream s;
                    s << current_obj + "."

                    symbols.Set(VAR_STRING,

                }

            } else

                if(p->tok_vec[i+1].type == '=')

                    if(Bounds(i+2))
                        symbols.Set(VAR

                }

                else {
                    symbols.Set(VAR_STRING,

                }

            }

        }

    }

break;
case VAR_FLOAT:
    i++;
    if(Bounds(i)) {

        if(Bounds(i+1)) {
            if(p->tok_vec[i].token.find('[') != -1

                std::string num = p->tok_vec[i]

                std::string name = p->tok_vec[i]

                for(size_t p = 0; p < size_t(at

                    std::ostringstream s;
                    s << current_obj + "."

                    symbols.Set(VAR_FLOAT, "

                }

            } else if(p->tok_vec[i+1].type == '=') {

                if(Bounds(i+2))
                    symbols.Set(VAR_FLOAT, p

                }

                else {
                    symbols.Set(VAR_FLOAT, "0.0", cu

                }

            }

        }

    }

break;
case TOK_MAIN:
    {
        i++;
        if(Bounds(i)) {

            prog_name = p->tok_vec[i].token;
            current_obj = "master";

            obj[p->tok_vec[i].token] = p->tok_vec[

        }

    }
}

```



```

        break;
    case TOK_USES:
    {
        i += 2;

        if(Bounds(i))
            while(p->tok_vec[i].type != ';' && i <
                uses.push_back(current_obj + ".

                i++;
                if(Bounds(i) && p->tok_vec[i].t

                i++;
            }
        }
        break;
    case TOK_MAS:
        i++;

        if(Bounds(i)) {
            current_obj = p->tok_vec[i].token;
            obj[current_obj] = current_obj;
        }
        break;
    case TOK_BEGIN:
        i ++;
        current_proc = current_obj + "." + "begin";

        if(Bounds(i))
            AddIncBlock(current_obj + "." + "begin", i);

        break;
    case TOK_END:
        i ++;
        current_proc = current_obj + "." + "end";

        if(Bounds(i))
            AddIncBlock(current_obj + "." + "end", i);

        break;
    case TOK_EXTERN:
        i += 2;

        if(Bounds(i)) {
            callb[current_obj + "." + p->tok_vec[i].token].
        }

        break;
    case TOK_PROC:
        i++;
        current_proc = current_obj + "." + p->tok_vec[i].token;

        if(Bounds(i+2)) {
            AddIncBlock(current_obj + "." + p->tok_vec[i].t
        }

        break;
    case TOK_IMPLEMENT:
    {
        i+=2;
        if(Bounds(i)) {

            callb[current_obj + "." + p->tok_vec[i]

            callb[current_obj + "." + p->tok_vec[i]

            callb[current_obj + "." + p->tok_vec[i]

        }

    }

    break;
}

}

}
break;
case STRING:

```

```

        break;

    case DIGIT:
        break;
    case CODELABEL:
        labels[p->tok_vec[i].token].label_name = p->tok_vec[i].token;

        labels[p->tok_vec[i].token].proc = current_proc;

        break;
    }
}

void BackEnd::AddIncBlock(std::string block_name, size_t start) {

    std::vector<Instruct> v;
    v.push_back ( Instruct(0xFF,block_name, "") );

    size_t pos;
    for(start, pos = 0; p->tok_vec[start].type != '}' && start < p->tok_vec.size(); start++) {

        int id = IdentToValue(p->tok_vec[start].token);

        if(id >= TOK_MOV && id <= TOK_PRINT) {

            Instruct i;
            i.op_code = id;
            i.op1 = p->tok_vec[start+1].token;

            if(Bounds(start+3) && p->tok_vec[start+2].type == ',')

                i.op2 = p->tok_vec[start+3].token;

            else if(Bounds(start+3) && p->tok_vec[start+1].type == '<') {

                std::ostringstream string_s;
                start++;
                while(Bounds(start) && p->tok_vec[start].type != ';') {

                    if(p->tok_vec[start].type != '<')

                        string_s << p->tok_vec[start].token << char(0xFF);

                    start++;

                }

                i.op1 = string_s.str();

                i.op2 = "";

            }
            else if(Bounds(start+3) && p->tok_vec[start+2].type == '<') {

                std::ostringstream string_s;
                i.op1 = p->tok_vec[start+1].token;

                start+=2;
                while(Bounds(start) && p->tok_vec[start].type != ';') {

                    if(p->tok_vec[start].type != '<')

                        string_s << p->tok_vec[start].token << char(0xFF);

                    start++;

                }
                i.op2 = string_s.str();

            }
            else

                i.op2 = "";

            v.push_back(i);

            pos++;

        }
        if(p->tok_vec[start].type == 4) {

            labels[p->tok_vec[start].token].pos = pos++;

        }

    }

    ins.push_back(v);
}

```

```

}

void BackEnd::DebugInc() {
    size_t i;
    for(i = 0; i < ins.size(); i++)
        for(size_t z = 0; z < ins[i].size(); z++) {
            if(ins[i][z].op_code == 0xFF)
                cout << "Code Block Name: " << ins[i][z].opl << endl;

            else
                cout << "Opcode: " << tok_array[ins[i][z].op_code] << " Operand 1: " << ins[i]
        }

    for(i = 0; i < uses.size(); i++)
        cout << "Uses clause: " << uses[i] << endl;

    std::map<std::string, Code_Label>::iterator l = labels.begin();
    for(l; l != labels.end(); l++)
        cout << "Code Label: " << l->second.label_name << " In Procedure: " << l->second.proc <

}

void BackEnd::CallProcedure(std::string proc_name) {
    size_t p = Find(proc_name);

    if(p != 0) {
        code.push_back(std::make_pair(cur_pos, i+1));

        cur_pos = p, i = 0;
        return;
    }

    else
        std::cout << "**** Procedure " << proc_name << " Not Found!\n";

}

void BackEnd::Execute(bool analyze) {
    if(analyze && !p->LexAnalyze()) {
        cout << "**** Errors in script cannot continue...\n";

#ifdef _WIN32
        system("PAUSE");
#endif

        return;
    }

    if(p->err != 0) {
        cout << "**** Errors in script cannot continue...\n";

#ifdef _WIN32
        system("PAUSE");
#endif

        return;
    }

    if(!p->tok_vec.empty())
        p->tok_vec.clear();
    if(p->file)
    {
        p->file->close();
    }

    size_t begin_pos = Find("master.begin"), end_pos = Find("master.end");

    size_t u, b, e;
    cur_pos = begin_pos;

    i = 0;

    if(end_pos != 0)

```

```

        code.push_back( std::pair<size_t,size_t>(end_pos,0) );

    if(uses.size()!=0) {
    for(u = 0; u < uses.size(); u++) {

        std::string str = uses[u].substr(uses[u].find(".")+1,uses[u].length());

        e = Find(str + ".end");
        if(e != 0)
            code.push_back(std::make_pair(e,0));
    }

    code.push_back(std::make_pair(begin_pos,0));
    for(u = 0; u < uses.size(); u++) {

        std::string str = uses[u].substr(uses[u].find(".")+1,uses[u].length());

        b = Find(str + ".begin");
        if(u != uses.size()-1)

            code.push_back(std::make_pair(b,0));
    }
    cur_pos = b;
    }

    else
        cur_pos = begin_pos;

    while(1) {

        while(i < ins[cur_pos].size()) {
            PROC_RETURN rt;

            if((rt = ProcInc(cur_pos, i)) == PROC_CONT)

                ins[cur_pos][i].op1 = prev_op1, ins[cur_pos][i].op2 = prev_op2, i++;

            else if(rt == PROC_JMP){
                ins[cur_pos][i].op1 = prev_op1, ins[cur_pos][i].op2 = prev_op2;

                size_t pos = Find(ins[cur_pos][i].op1);
                if(pos != 0) {

                    if(ins[pos][0].op_code == 0xFF) {
                        code.push_back(std::pair<size_t,size_t>(cur_pos,i+1));

                        code.push_back(std::pair<size_t,size_t>(pos,0));

                        break;
                    }
                }
                else
                {
                    if(callb[ins[cur_pos][i].op1].name == "" || callb[ins[cur_pos][i].op1].

                        cout << "**** invalid procedure call-> " << ins[cur_pos][i].op1

                    else

                        if(callb[ins[cur_pos][i].op1].f(symbols) != 0)

                            std::cerr << "Error external callback didnt return 0\n"

                                i++;

                            }
                        }
                    else if(rt == PROC_RET) {

                        ins[cur_pos][i].op1 = prev_op1, ins[cur_pos][i].op2 = prev_op2;

                        break;
                    }
                }

            if(code.empty())
                break;

            cur_pos = code.back().first;
            i = code.back().second;

            code.pop_back();
        }
    }
}

```

```

PROC_RETURN BackEnd::ProcInc(size_t block, size_t &inc) {

    prev_op1 = ins[block][inc].op1;
    prev_op2 = ins[block][inc].op2;

    if(ins[block][inc].op1.find('[') != -1 && ins[block][inc].op1.find(']') != -1) {

        std::ostringstream curs;
        std::string num = ins[block][inc].op1;

        std::vector<std::string> vec;
        while(num.find('[') != -1) {

            num = num.substr(num.find('[')+1,num.length());

            num = num.substr(0,num.rfind(']'));

            vec.push_back(num);

        }

        std::string prev_index = "";

        for(int array_pos = int(vec.size())-1; array_pos >= 0; array_pos--) {

            std::string the_name = "";
            if(vec[array_pos].find('[') != -1) {

                if(IsVar(vec[array_pos])) {
                    prev_index = symbols.vars[vec[array_pos]].val;
                }

                else {
                    the_name = vec[array_pos].substr(0,vec[array_pos].find('['));
                    prev_index = symbols.vars[the_name + "[" + prev_index + "].val;
                }

            }

            else
                prev_index = (IsVar(vec[array_pos]) == true ? symbols.vars[vec[array_pos]].val

        }

        std::string name = ins[block][inc].op1.substr(0,ins[block][inc].op1.find('['));

        curs << name << "[" << prev_index << "]";
        ins[block][inc].op1 = curs.str();
    }

    if(ins[block][inc].op2.find('[') != -1 && ins[block][inc].op2.find(']') != -1) {

        std::ostringstream curs;
        std::string num = ins[block][inc].op2;

        std::vector<std::string> vec;
        while(num.find('[') != -1) {

            num = num.substr(num.find('[')+1,num.length());

            num = num.substr(0,num.rfind(']'));

            vec.push_back(num);

        }

        std::string prev_index = "";

        for(int array_pos = int(vec.size())-1; array_pos >= 0; array_pos--) {

            std::string the_name;
            if(vec[array_pos].find('[') != -1) {

                if(IsVar(vec[array_pos])) {
                    prev_index = symbols.vars[vec[array_pos]].val;
                }

                else {
                    the_name = vec[array_pos].substr(0,vec[array_pos].find('['));
                    prev_index = symbols.vars[the_name + "[" + prev_index + "].val;
                }

            }

        }

    }

}

```

```

        else
            prev_index = (IsVar(vec[array_pos]) == true ? symbols.vars[vec[array_pos]].val
        }

        std::string name = ins[block][inc].op2.substr(0,ins[block][inc].op2.find(' '));

        curs << name << "[" << prev_index << "]";
        ins[block][inc].op2 = curs.str();
    }

    switch(ins[block][inc].op_code) {
        case TOK_MOV:

            if(IsVar(ins[block][inc].op1)) {
                if(IsVar(ins[block][inc].op2))

                    symbols.vars[ins[block][inc].op1].val = symbols.vars[ins[block][inc].op2].val;

            else
                symbols.vars[ins[block][inc].op1].val = ins[block][inc].op2;

            }

        else
            std::cout << "**** invalid variable name on mov..." << ins[block][inc].op1 <<

            break;
        case TOK_ADD:
        case TOK_SUB:
        case TOK_MUL:

        case TOK_DIV:
        case TOK_OR:
        case TOK_XOR:
        case TOK_AND:
        {
            if(IsVar(ins[block][inc].op1)) {

                if(IsVar(ins[block][inc].op2))
                {
                    std::ostringstream s;

                    switch(ins[block][inc].op_code)
                    {
                        case TOK_ADD:

                            s << (symbols.vars[ins[block][inc].op1].type == VAR_INT

                                break;
                            case TOK_SUB:
                                s << (symbols.vars[ins[block][inc].op1].type == VAR_INT

                                    break;
                                case TOK_MUL:
                                    s << (symbols.vars[ins[block][inc].op1].type == VAR_INT

                                        break;
                                    case TOK_DIV:
                                        if(std::atoi(symbols.vars[ins[block][inc].op1].val.c_str()) == 0)
                                            std::cout << "***** Error cannot divide by zero

                                                else
                                                    s << (symbols.vars[ins[block][inc].op1].type ==

                                                        break;
                                                        case TOK_OR:
                                                            s << (std::atoi(symbols.vars[ins[block][inc].op1].val.c_str()) == 0)

                                                                break;
                                                                case TOK_XOR:
                                                                    s << (std::atoi(symbols.vars[ins[block][inc].op1].val.c_str()) == 0)

                                                                        break;
                                                                        case TOK_AND:
                                                                            s << (std::atoi(symbols.vars[ins[block][inc].op1].val.c_str()) == 0)

                                                                                break;
                                                                                }
                                                                            symbols.vars[ins[block][inc].op1].val = s.str();
                                                                        }
                        }
                    }
                }
            }
        }
    }

```

```

else
{
    std::ostringstream s;
    switch(ins[block][inc].op_code)
    {
        case TOK_ADD:
            s << (symbols.vars[ins[block][inc].op1].type == VAR_INT
                break;
        case TOK_SUB:
            s << (symbols.vars[ins[block][inc].op1].type == VAR_INT
                break;
        case TOK_MUL:
            s << (symbols.vars[ins[block][inc].op1].type == VAR_INT
                break;
        case TOK_DIV:
            if(std::atoi(symbols.vars[ins[block][inc].op1].val.c_str())
                cout << "**** Error cannot divide by zero...\n
            else
                s << (symbols.vars[ins[block][inc].op1].type ==
                break;
        case TOK_OR:
            s << (std::atoi(symbols.vars[ins[block][inc].op1].val.c
                break;
        case TOK_XOR:
            s << (std::atoi(symbols.vars[ins[block][inc].op1].val.c
                break;
        case TOK_AND:
            s << (std::atoi(symbols.vars[ins[block][inc].op1].val.c
                break;
    }
    symbols.vars[ins[block][inc].op1].val = s.str();
}
}
else
    std::cout << "**** invalid variable name ..." << ins[block][inc].op1 <
}
break;
case TOK_NOT:
{
    if(IsVar(ins[block][inc].op1)) {
        std::ostringstream s;
        s << (!std::atoi(symbols.vars[ins[block][inc].op1].val.c_str()));
        symbols.vars[ins[block][inc].op1].val = s.str();
    }
    else
        std::cout << "**** invalid variable name ..." << ins[block][inc].op1 <
}
break;
case TOK_INC:
case TOK_DEC:
{
    if(IsVar(ins[block][inc].op1)) {
        std::ostringstream s;
        if(ins[block][inc].op_code == TOK_INC)
            s << (std::atoi(symbols.vars[ins[block][inc].op1].val.c_str())+
        else
            s << (std::atoi(symbols.vars[ins[block][inc].op1].val.c_str())-
        symbols.vars[ins[block][inc].op1].val = s.str();
    }
}
else
    std::cout << "**** invalid variable name ..." << ins[block][inc].op1 <

```

```

    }
    break;
case TOK_PUSH:
{
    static symb s;

    if(IsVar(ins[block][inc].op1)) {
        s.name = ins[block][inc].op1;

        s.val = symbols.vars[s.name].val;

        s.type = symbols.vars[s.name].type;

        symbols += s;
    }
    else
    {
        s.name = "constant";

        s.type = IDENTIFIER;
        s.val = ins[block][inc].op1;

        symbols += s;
    }
}
break;
case TOK_POP:
{
    if(IsVar(ins[block][inc].op1))
    {
        symb s = --symbols;

        symbols[ins[block][inc].op1].val = s.val;
    }
    else
        std::cout << "**** invalid variable name on pop..." << ins[block][inc]

}
break;
case TOK_PRINT:
    if(ins[block][inc].op1.find(char(0xFF)) == -1 && symbols.vars[ins[block][inc].op1].name

        std::cout << symbols.vars[ins[block][inc].op1].val;

    else if(ins[block][inc].op1.find(char(0xFF)) == -1) {

        std::cout<< ins[block][inc].op1;
    }
    else { // stream data

        std::ostringstream string_s;
        std::string left, right,temp = ins[block][inc].op1;

        while(temp.length() > 0 && temp.find(char(0xFF)) != -1) {

            left = temp.substr(0,temp.find(char(0xFF)));

            temp = temp.substr(temp.find(char(0xFF))+1,temp.length());

            if(IsVar(left))
                string_s << symbols.vars[left].val;

            else
                string_s << left;

        }

        cout << string_s.str();
    }
}
break;
case TOK_SCAT:
{
    std::ostringstream string_s;
    if(ins[block][inc].op2.find(char(0xFF)) == -1 && symbols.vars[ins[block][inc].o

        std::cout << "**** error no data to stream to variable...\n";
    }
    else { // stream data

```



```

        std::string left, right, temp = ins[block][inc].op2;
        while(temp.length() > 0 && temp.find(char(0xFF)) != -1) {
            left = temp.substr(0, temp.find(char(0xFF)));
            temp = temp.substr(temp.find(char(0xFF))+1, temp.length());
            if(IsVar(left))
                string_s << symbols.vars[left].val;
            else
                string_s << left;
        }
        symbols.vars[ins[block][inc].op1].val = string_s.str();
    }
}

break;
case TOK_CALL:
    return PROC_JMP;
break;

case TOK_JMP:
    if(ins[block][0].op1 == labels[ins[block][inc].op1].proc)
        inc = labels[ins[block][inc].op1].pos;
    else
        std::cout << "***** on jmp instruction wrong procedure or no label found.\n";
    break;
case TOK_JNE:
    if(ins[block][0].op1 == labels[ins[block][inc].op1].proc) {
        if(reg[0] == false) inc = labels[ins[block][inc].op1].pos;
    }
    else
        std::cout << "***** on jmp instruction wrong procedure or no label found.\n";
    break;

case TOK_JE:
    if(ins[block][0].op1 == labels[ins[block][inc].op1].proc) {
        if(reg[0] == true) inc = labels[ins[block][inc].op1].pos;
    }
    else
        std::cout << "***** on jmp instruction wrong procedure or no label found.\n";
    break;

case TOK_JL:
    if(ins[block][0].op1 == labels[ins[block][inc].op1].proc) {
        if(reg[1] != true) inc = labels[ins[block][inc].op1].pos;
    }
    else
        std::cout << "***** on jmp instruction wrong procedure or no label found.\n";
    break;

case TOK_JG:
    if(ins[block][0].op1 == labels[ins[block][inc].op1].proc) {
        if(reg[1] == true) inc = labels[ins[block][inc].op1].pos;
    }
    else
        std::cout << "***** on jmp instruction wrong procedure or no label found.\n";
case TOK_JLE:
    if(ins[block][0].op1 == labels[ins[block][inc].op1].proc) {
        if(reg[3] == true) inc = labels[ins[block][inc].op1].pos;
    }
}

```

```

        else
            std::cout << "***** on jmp instruction wrong procedure or no label found.\n";
            break;
    case TOK_JGE:
        if(ins[block][0].opl == labels[ins[block][inc].opl].proc) {
            if(reg[2] == true) inc = labels[ins[block][inc].opl].pos;
        }
        else
            std::cout << "***** on jmp instruction wrong procedure or no label found.\n";
            break;
    case TOK_CMP:
        {
            if(IsVar(ins[block][inc].opl)) {
                if(IsVar(ins[block][inc].op2)) {
                    reg[0] = (symbols.vars[ins[block][inc].opl].val == symbols.vars
                    reg[1] = (std::atof(symbols.vars[ins[block][inc].opl].val.c_str
                    reg[2] = (std::atof(symbols.vars[ins[block][inc].opl].val.c_str
                    reg[3] = (std::atof(symbols.vars[ins[block][inc].opl].val.c_str

                }
                else {
                    reg[0] = (symbols.vars[ins[block][inc].opl].val == ins[block][i
                    reg[1] = (std::atof(symbols.vars[ins[block][inc].opl].val.c_str
                    reg[2] = (std::atof(symbols.vars[ins[block][inc].opl].val.c_str
                    reg[3] = (std::atof(symbols.vars[ins[block][inc].opl].val.c_str

                }
            }
        }
        break;
    case TOK_RET:
        {
            return PROC_RET;
        }
        break;
    case TOK_EXIT:
        {
            std::exit(0);
        }
        break;
    }
    return PROC_CONT;
}

size_t BackEnd::Find(std::string what) {
    for(size_t i = 0; i < ins.size(); i++)
        if(strcmp(ins[i][0].opl.c_str(),what.c_str()) == 0)
            return i;
    return 0;
}

void BackEnd::AddExternal(std::string name, int (*f)(Symbol_Table &t)) {
    callb[name].f = f;
    callb[name].name = name;
}

bool BackEnd::IsVar(std::string &name) {
    if(symbols.vars[name].name == "")
        return false;

```

```

        return true;
    }

std::string Backend::byteToString(unsigned char type) {
    std::string s;
    switch(type) {
        case 1:

            case 2: return "Identifier";
            case 3: return "Digit";

            case 4: return "String";
            case 5: return "Code Label";

    }

    s += type;
    return s;
}

std::string Backend::tokenTypeToString(TOK_TYPE t) {
    std::string s;

    switch(t) {
        case 0: return "Constant";

        case VAR_INTEGER: return "Integer";
        case VAR_STRING: return "String";
        case VAR_FLOAT: return "Float";
    }

    s += int(t);
    return s;
}

void Backend::DebugHTML(std::string src, bool sout) {
    std::fstream file;
    file.open(src.c_str(), std::ios::out);

    if(file.is_open()) {
        size_t i = 0, z = 0;

        file << "<HTML><TITLE>Masc Debug Output File for " << prog_name << "</TITLE><BODY>";

        file << "<font face=\"FixedSys\" size=\"4\" color=\"black\"><TABLE border=\"4\" bgcolor=\"#FFFF";
        file << "<TR><font color=\"#BD0000\"> Tokenized Code </font> </TR><br><TH> File/Line </TH><TH>";

        for(i = 0; i < p->tok_vec.size(); i++)

            file << "<TR><TH><font color=\"blue\"> " << p->tok_vec[i].source_file << ":" << p->tok_v

        file << "</TABLE><TABLE border =\"4\" bgcolor=\"#FFFFFF\" BORDERCOLOR=\"#000000\" CELLPADDING=\"";
        for(i = 0; i < ins.size(); i++)

            for(z = 0; z < ins[i].size(); z++)

                file << "<TR><TH><font color=\"orange\"> " << (ins[i][z].op_code == 0xFF ? "Co

        file << "</TABLE><TABLE border =\"4\" bgcolor=\"#FFFFFF\" BORDERCOLOR=\"#000000\" CELLPADDING=\"";
        std::map<std::string,symb>::const_iterator x;
        for(x = symbols.vars.begin(); x != symbols.vars.end(); x++)

            file << "<TR><TH><font color=\"teal\"> " << x->first << "</font></TH> <TH><font color="

        std::map<std::string, masc::External_Callback>::iterator y;

        file << "</TABLE><TABLE border =\"4\" bgcolor=\"#FFFFFF\" BORDERCOLOR=\"#000000\" CELLPADDING=\"";
        for(y = callb.begin(); y != callb.end(); y++)
    }
}

```

```

        file << "<TR><TH><font color=\"purple\">" << y->first << "</font></TH> <TH><font color
file << "<font></TABLE></BODY></HTML>";
file.close();
}

// dump debug output to stdout
if(sout == true) {
    std::fstream ifile;

    cout << "Content-type: text/html\n\n";
    ifile.open(src.c_str(),std::ios::in);

    while(ifile.is_open() && !ifile.eof()) {
        std::string s;

        std::getline(ifile,s);
        cout << s << "\n";
    }

    ifile.close();
}

inline int IdentToValue(std::string s) {
    for(int i = 0; tok_array[i] != 0; i++)
        if(s == tok_array[i])
            return i;
    return -1;
}

BackEnd *BackEnd::back;
}

```

```

#include "mascstd.h"
// #define MASC_APACHE // comment out if not running under apache

int main(int argc, char *argv[]) {

    bool confirm = false;
    if(argc == 2 || argc == 3) {

        masc::BackEnd bend( new masc::Parser(argv[1]) );

        bend.Convert();
        mascstd::SetStdFunc(bend);
#ifdef MASC_APACHE
        cout << "Content-type: text/html\n\n";
#endif

        bool analyze = true;
        if(argc == 3) {

            if(strcmp(argv[2], "-d") == 0)
                bend.DebugHTML(bend.GetName() + "_debug.html", false);

            else if(strcmp(argv[2], "-dstd") == 0)
                bend.DebugHTML(bend.GetName() + "_debug.html", true);

            else if(strcmp(argv[2], "-dstdn") == 0)
                bend.Debug();

            else if(strcmp(argv[2], "-na") == 0)
                analyze = false;

            else if(strcmp(argv[2], "-con") == 0)
                confirm = true;

            else {
                std::cout << "Invalid parameter [" << argv[2] << "] specified\n";
            }
        }

#ifdef _WIN32
#ifdef MASC_APACHE
        system("PAUSE");
#endif
#endif

        return 0;
    }

    bend.Execute(analyze);

    if(confirm == true) {
        std::cout << " Script (" << argv[1] << ") succesfully executed... \n";
    }

#ifdef _WIN32
#ifdef MASC_APACHE
        system("PAUSE");
#endif
#endif

    }

    else {

        std::cout << "MasC++ {Master Assembly Script Container++ (written in C++) " << double(masc::version) <<

#ifdef _WIN32
#ifdef MASC_APACHE
        system("PAUSE");
#endif
#endif

    }

    return 0;
}

```

```

#ifndef GARBAGE_H__
#define GARBAGE_H__

class Garbage_Collector {
public:
    ~Garbage_Collector() {
        for(size_t i = 0; i < size; i++) {

            /*char data[256];
            std::sprintf(data, "Freed data: %s\n", mem_ptr[i]);
            OutputDebugString(data);*/
            free(mem_ptr[i]);
        }
    }

    Garbage_Collector() {
        mem_ptr = 0;

        size = 0;
    }

    void *alloc(size_t pos) {

        void *ptr = malloc(pos);
        mem_ptr = (void**)realloc(mem_ptr, sizeof(void**) * (++size));

        void **temp_ptr = mem_ptr;
        temp_ptr += (size-1);
        *temp_ptr = ptr;

        return ptr;
    }

    void *operator[](size_t pos) { return mem_ptr[pos]; }

private:
    void **mem_ptr;
    size_t size;

};

static Garbage_Collector mem;

#endif

```