

Forum:[Hack Forums](#)

Thread: [All about compilers! - Make your own language](#)

magicparty - August 29, 2019, 9:34 pm

This is my first real contribution here on HF I hope you all enjoy! - Most of this information is just off the top of the head feel free to comment on any errors.

Compilers

(1) - Overview

You may wonder what exactly a compiler does and that fortunately has a simple answer; it converts instructions written in a specific syntax (hopefully with the goal of being more human readable) into for the most part completely unreadable lower level languages like assembler or machine code.

I'm open to have the discussion on how we handle compiling for multiple different CPUs with different machine code commands, but for now I will skip over that bit of knowledge.

Continuing on...You do not need to know machine code or assembler to build a compiler anymore! We have evolved over the years building compilers on top of older compilers using our more modern and powerful programming languages. It's almost like a chain of events depending on which language you are compiling (if it is a compiled language).

In this tutorial I will be using python to build a simple compiler-like program. It's not a true compiler but it acts similar and is enough to get you started on writing an actual compiler in a more popular language for the task, such as C.

(2) - Decide the syntax

First off I think it is worth your time to write a fairly complex program in your made up language and work backwards testing each piece of the compiler along the way. For tutorial purposes I will only write a simple program to keep this an easier read.

```
magicpartylang
Code # This is my first program ever in magicpartylang
message = "Hello World!"
x = 10
y = 10
z = x + y
w @ z
prnt message
prnt
prnt z
prnt
prnt w
# End of my program!
```

As you can see I've gone just a few steps further than the classic "Hello World" because I'm going to test a few features. Also I've added the line "w @ z" which makes no sense for what you may be

used to but this is our language so you can overlay your own features that may help you accomplish specific tasks easier! For this tutorial's purpose the command is probably useless but we will say that the @ statement takes in any data type and assigns it backwards so "w @ z" when z = 20 will result in "w = 2" or if we said "w @ message" the result would be "w = !dlroW olleH".

(3) - Lexical Analysis

The lexer is a tokenizer of sorts that strips any unwanted things out of the code i.e white spaces or comments and the rest of what you are left with is tokens for each statement of your program -- This is where the coding begins!

To me this is somewhat of a simple task using python considering we have the split() method provided for us in the development package, and a bunch of file operation tools.

A simple script to open a file of .mpl (magic-party-lang) extension and tokenize it stripping all un-necessary parts

script

```
Code def tokenize_code(code):
```

```
    statements = [line.split() for line in code.split('\n') if not '#' in line and len(line) > 0]
    lexical_analyzed = { }
```

```
    for idx in range(len(statements)):
```

```
        lexical_analyzed[idx] = statements[idx]
```

```
    return lexical_analyzed
```

```
def get_code_from_file(file_name):
```

```
    file = open(file_name, "r")
    return file.read()
```

```
if __name__ == '__main__':
```

```
    code = get_code_from_file("example.mpl")
    print(tokenize_code(code))
```

To test I've saved the script as hfcompiler.py which will be the file I'm working with for the remainder of this tutorial. As you may see the name of my made up programming language file is example.mpl

result of script

```
Code {
```

```
0:[
    'message',
    '=',
    '"Hello',
    'World!'"

```

```
],
1:[
    'x',
    '=',
    '10'

```

```

],
2:[
  'y',
  '=',
  '10'
],
3:[
  'z',
  '=',
  'x',
  '+',
  'y'
],
4:[
  'w',
  '@',
  'z'
],
5:[
  'prnt',
  'message'
],
6:[
  'prnt'
],
7:[
  'prnt',
  'z'
],
8:[
  'prnt'
],
9:[
  'prnt',
  'w'
]
]

```

(4) - Syntax Analyzer

Now it is time to break each line down and analyze the statements made in each line, for simplicity reasons this won't be the most wonderfully formatted code, but it gets the point across and the job done.

For my programming language like others there will be a level of precedence for each statement. For example '=' comes after a '+' in the hierarchy. Based upon this idea I will create simple if else statements to determine what to do with the line of code. At this stage most complex compilers pass the line of code through a tree like structure to form commands but since this language is so simple it won't be that robust.

Here is a script to pull out a few things like variable assignments, and print statements

```

script
Code def prnt_statement(line):

```

```
return 'print(' + ' '.join(line[1:]) + '')
```

```
def var_assignment(line):
```

```
    variable_name = line[line.index("=") - 1]
    value = line[line.index("=") + 1:]
    return variable_name + "=" + ' '.join(value)
```

```
if __name__ == '__main__':
```

```
    code = get_code_from_file("example.mpl")
    python_code = []
    errors = []
    lexer = tokenize_code(code)
```

```
    for linenumber in lexer.keys():
```

```
        if "=" in lexer[linenumber]:
```

```
            python_code.append(var_assignment(lexer[linenumber]))
```

```
        elif "prnt" in lexer[linenumber]:
```

```
            python_code.append(prnt_statement(lexer[linenumber]))
```

```
        else:
```

```
            errors.append("Syntax error on line " + str(linenumber) + ": " + ' '.join(lexer[linenumber]))
```

```
    print('\n'.join(python_code))
```

```
    print('\n'.join(errors))
```

This code may seem complex to a python beginner but it's really nothing more but some data structure manipulation.

result of script

```
Code message="Hello World!"
```

```
x=10
```

```
y=10
```

```
z=x + y
```

```
print(message)
```

```
print()
```

```
print(z)
```

```
print()
```

```
print(w)
```

```
Syntax error on line 4: w @ z
```

As you can see we have converted the target features to Python 3.x code! But what do we do with that syntax error? Well it seems that we haven't created a function to handle the @ statement as discussed earlier in section 2 to fix this I'll add a method to handle this situation along with the @ statement in the hierarchy.

@ method

```
Code def reverse_assignment(line):
```

```

variable_name = line[line.index("@") - 1]
value = line[line.index("@") + 1:]
return variable_name + " = ".join([str(" + ' ".join(value) + ")")[idx] for idx in range(len(str(" + '
'.join(value) + ")") - 1, -1, -1)])"
```

I'm not going to go over how to implement the statement into the hierarchy as that should be fairly self explanatory at this point.

result of script

```
Code message="Hello World!"
```

```
x=10
```

```
y=10
```

```
z=x + y
```

```
w = ".join([str(z)[idx] for idx in range(len(str(z)) - 1, -1, -1)])"
```

```
print(message)
```

```
print()
```

```
print(z)
```

```
print()
```

```
print(w)
```

```
(5) - Program Writer
```

Not much to go over in this section, you just need to develop some sort of file writing system that takes the generated code and streams it into a file with the proper file type.

In this case I'm compiling my language into Python 3.x so my file type will be a *.py file.

```
Code file = open("example.py", "w+")
```

```
file.write("\n".join(python_code))
```

```
file.close()
```

```
print("Done compiling!")
```

```
(6) - Finalizing the engineering of your system
```

I am using a Mac and obviously since the OS X system software is close to linux these next steps apply for both systems.

Windows users...I'm sorry...But you totally can upload your compiler to a Google App Engine / Heroku project and use the console there to compile your custom programming language!

To start off we need two programs; one to invoke the compiler and another to execute the program. Though not totally necessary as you can just run the compiled code with python here it will just keep everything in a nice package for our tutorial.

compiler method

```
Code def compile(file_name)
```

```
code = get_code_from_file(file_name)
```

```
python_code = []
```

```
errors = []
```

```
lexer = tokenize_code(code)
```

```
for linenumber in lexer.keys():
```

```
    if "=" in lexer[linenumber]:
```

```

python_code.append(var_assignment(lexer[linenumber]))

elif "prnt" in lexer[linenumber]:

    python_code.append(prnt_statement(lexer[linenumber]))

elif "@" in lexer[linenumber]:

    python_code.append(reverse_assignment(lexer[linenumber]))

else:

    errors.append("Syntax error on line " + str(linenumber) + ": " + ' '.join(lexer[linenumber]))

file = open(file_name[:-4] + ".py", "w+")
file.write('\n'.join(python_code))
file.close()
print("Done compiling with " + str(len(errors)) + " errors")
mplc.py compiler script
Code import hfcompiler
import sys

file_name = sys.argv[1]
hfcompiler.compile(file_name)
mpl.py runs mpl program
Code import sys
import os

mpl_file = sys.argv[1]
os.system("python3 " + mpl_file + ".py")
Now it's time to make system alias for each command we want to use.

Code alias mplc = 'python3 mplc.py'
alias mpl = 'python3 mpl.py'
You can add them to the ~/.bashrc file, profile.d, etc. to make it global if you wish as well.

```

(7) - Running our first magicpartylang program!

To compile:

Code mplc example.mplc
Should result in a "Done compiling!"

Code mpl example
Should result in the following output (unless you modified the original code)

Code Hello World!

20

02

If you take note here as expected our simple assignments worked perfectly, our variable w was

assigned the reverse string of z. z was assigned $x + y$ where both are equal to 10.

The code works.

(7) - Conclusion

The purpose of this tutorial is to show you the very high-level basics on how compilers work. Obviously this is not a true compiler compared to what you may find in a professional software development package.

Lindros - August 29, 2019, 10:30 pm

This actually looks to be a good contribution, thanks for this.

I don't know too much about writing compilers yet. I am just beginning in my software engineering courses.

Where did you learn?

Temp_reassign - August 29, 2019, 11:00 pm

Great effort. Considering you're new around and done something.

magicparty - August 29, 2019, 11:18 pm

(Aug 29, 2019 10:30 PM)ZietzSeV Wrote: [\[CITE\]](#) This actually looks to be a good contribution, thanks for this.

I don't know too much about writing compilers yet. I am just beginning in my software engineering courses.

Where did you learn?

I learned from several resources; myself when I set off to build an 8-Bit computer from pure logic chips, Paradigms in C++, and Architecture.

If you want me to clarify anything in regards to compilers or a computer science journey feel free to ask me at any time.

(Aug 29, 2019 11:00 PM)Temp_reassign Wrote: [\[CITE\]](#) Great effort. Considering you're new around and done something.

And more content will be coming soon in several different sections!

Lindros - August 30, 2019, 9:07 am

(Aug 29, 2019 11:18 PM)magicparty Wrote: [\[CITE\]](#) I learned from several resources; myself when I set off to build an 8-Bit computer from pure logic chips, Paradigms in C++, and Architecture.

If you want me to clarify anything in regards to compilers or a computer science journey feel free to ask me at any time.

And more content will be coming soon in several different sections!

Well, I hope to see you stick around. We could use some more active members in this section.

Morphine - August 31, 2019, 3:33 pm

Very good project i was thinking to do this months before

magicparty - August 31, 2019, 3:59 pm

(Aug 31, 2019 03:33 PM)Whoamii Wrote: [\[CITE\]](#) Very good project i was thinking to do this months before

Definitely a great learning experience.

tux1 - September 1, 2019, 4:59 am

Interesting thread, hope to see some more posts from you!