



MODUL TEKNIK KOMPILASI

BAHASA PEMOGRAMAN (RHS LANG)

Dosen Pengampu: Aji Purwinarko, S.Si., M.Cs.

Repository:

<https://github.com/fauzaaulia/Rhs-Lang>

Disusun oleh:

Fitri Amalia Langgundi (4611417009)

Nia Zulia Saputri (4611417020)

Ahmad Fauza Aulia (4611417038)

JURUSAN ILMU KOMPUTER

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS NEGERI SEMARANG

2019

KATA PENGANTAR

Puji syukur alhamdulillah kami panjatkan ke hadirat Tuhan Yang Maha Esa yang telah melimpahkan nikmat, taufik serta hidayah-Nya yang sangat besar sehingga kami pada akhirnya bisa menyelesaikan laporan Modul Teknik Kompilasi “RHS Lang” ini tepat pada waktunya.

Rasa terima kasih juga kami ucapkan kepada Dosen Pengampu mata kuliah Teknik Kompilasi yang selalu memberikan dukungan serta pengetahuannya sehingga Modul ini dapat disusun dengan baik.

Semoga Modul Teknik Kompilasi yang telah kami susun ini turut memperkaya khazanah Ilmu Komputer serta bisa menambah pengetahuan dan pengalaman para pembaca.

Selayaknya kalimat yang menyatakan bahwa tidak ada sesuatu yang sempurna. Kami juga menyadari bahwa Modul Teknik Kompilasi ini juga masih memiliki banyak kekurangan. Maka dari itu kami mengharapkan saran serta masukan dari para pembaca demi penyusunan Modul serupa yang lebih baik lagi

Semarang, 27 Juni 2019

Penyusun

DAFTAR ISI

Halaman Sampul	i
Kata Pengantar.....	ii
Daftar Isi	iii
BAB I PENDAHULUAN.....	1
1.1. Latar Belakang.....	1
1.2. Rumusan Masalah.....	1
1.3. Tujuan	1
BAB II PEMBAHASAN.....	3
2.1. Landasan Teori	3
2.2. Alat dan Bahan	5
2.3. Proses Instalasi	5
2.4. Dokumentasi Pembuatan Bahasa Pemograman Rhs-Lang	6
2.5. Cara Penggunaan Bahasa Rhs-Lang	12
2.6. Bahasa Perintah Rhs-Lang	15
BAB III PENUTUP	18
3.1. Akhir Kata	18
LAMPIRAN-LAMPIRAN	19
Lampiran 1. rhs_lexer.py.....	19
Lampiran 2 rhs_parser.py	21
Lampiran 3 rhs_interpreter.py	24
Lampiran 4 main.py	27

BAB I

PENDAHULUAN

1.1. LATAR BELAKANG

Bahasa pemrograman atau sering diistilahkan juga dengan bahasa komputer adalah teknik komando/instruksi untuk memerintahkan komputer. Bahasa pemrograman ini merupakan suatu himpunan dari aturan sintaks dan semantik yang dipakai untuk mendefinisikan program komputer. Bahasa ini memungkinkan seorang programmer dapat menentukan secara persis data mana yang akan diolah oleh komputer, bagaimana data ini akan disimpan/diteruskan, dan jenis langkah apa secara persis yang akan diambil dalam berbagai situasi. Semakin sederhana sintaksis dan cepat programnya maka bahasa pemrograman tersebut akan sangat efisien. Namun dalam proses pembuatan program tetap harus memahami pola dan nama sintaksis yang telah diterapkan pada setiap bahasa pemrograman.

Dalam dunia ilmu komputer kami ingin membuat sesuatu yang baru dengan pola dan nama sintaksis yang berbeda pada bahasa pemrograman sehingga dapat menarik minat yang lebih saat menulis program. Dan kami akan membuat bahasa pemrograman Rhs-Lang

1.2. RUMUSAN MASALAH

1. Bagaimana cara pembuatan bahasa pemrograman Rhs-Lang?
2. Bagaimana cara menggunakan bahasa pemrograman Rhs-Lang?
3. Bagaimana cara menjalankan bahasa pemrograman Rhs-Lang?

1.3. TUJUAN

1. Pemenuhan tugas akhir mata kuliah teknik kompilasi.
2. Meningkatkan pengetahuan dan mampu mengimplementasikan teknik kompilasi pada pembuatan aplikasi dalam suatu bahasa yang inovatif.
3. Menginformasikan cara penggunaan bahasa pemrograman Rhs-Lang.

4. Meningkatkan kerjasama antar praktikan dalam menyelesaikan praktikum teknik kompilasi
5. Sebagai referensi pembelajaran lanjutan

BAB II

PEMBAHASAN

2.1. LANDASAN TEORI

2.1.1. Python

Bahasa pemrograman python adalah bahasa pemrograman tinggi yang dapat melakukan eksekusi sejumlah instruksi multi guna secara langsung (interpretatif) dengan metode orientasi objek (Object Oriented Programming) serta menggunakan semantik dinamis untuk memberikan tingkat keterbacaan syntax. Sebagai bahasa pemrograman tinggi, python dapat dipelajari dengan mudah karena sudah dilengkapi dengan manajemen memori otomatis (pointer).

Python dapat digunakan secara bebas, bahkan untuk kepentingan komersial sekalipun. Banyak perusahaan yang mengembangkan bahasa pemrograman python secara komersial untuk memberikan layanan. Misalnya Anaconda Navigator, adalah salah satu aplikasi untuk pemrograman python yang dilengkapi dengan tool-tool pengembangan aplikasi.

Python diklaim mampu memberikan kecepatan dan kualitas untuk membangun aplikasi bertingkat (Rapid Application Development). Hal ini didukung oleh adanya library dengan modul-modul baik standar maupun tambahan misalnya NumPy, SciPy, dan lain-lain. Python juga mempunyai komunitas yang besar sebagai tempat tanya jawab.

Mesin pencari Google adalah contoh nyata dari penggunaan bahasa pemrograman python dalam kehidupan sehari-hari. Mesin pencari ini termasuk Rapid Application Development, ia tidak hanya berguna untuk mencari halaman website. Kolom pencarian Google juga dapat digunakan sebagai kalkulator, membuat grafik fungsi, memprediksi cuaca, memprediksi

harga saham, terjemahan, mencari dengan gambar, menanyakan hari, pemesanan tiket pesawat, dan lain-lain.

2.1.2. **Library SLY (Sly Lex Yacc)**

SLY adalah perpustakaan untuk menulis parser dan kompiler. Ini secara longgar didasarkan pada alat konstruksi tradisional compiler lex dan yacc dan mengimplementasikan algoritma parsing LALR⁽¹⁾ yang sama. Sebagian besar fitur yang tersedia di lex dan yacc juga tersedia di SLY. Perlu juga dicatat bahwa SLY tidak menyediakan banyak hal seperti bel dan peluit (mis., Konstruksi otomatis pohon sintaksis abstrak, traversal pohon, dll.).

SLY menyediakan dua kelas terpisah yaitu Lexer dan Parser. Kelas Lexer digunakan untuk memecah teks input menjadi kumpulan token yang ditentukan oleh kumpulan aturan ekspresi reguler. Kelas Parser digunakan untuk mengenali sintaks bahasa yang telah ditentukan dalam bentuk tata bahasa bebas konteks. Dua kelas biasanya digunakan bersama untuk membuat parser. Namun, ini bukan persyaratan ketat - ada banyak fleksibilitas yang diperbolehkan. Dua bagian selanjutnya menjelaskan dasar-dasarnya.

Berikut adalah beberapa fitur penting dalam SLY:

1. SLY menyediakan pelaporan kesalahan dan informasi diagnostik yang sangat luas untuk membantu dalam pembuatan parser. Implementasi asli dikembangkan untuk tujuan pengajaran. Akibatnya, sistem mencoba mengidentifikasi jenis kesalahan paling umum yang dibuat oleh pengguna pemula.
2. SLY memberikan dukungan penuh untuk produksi kosong, pemulihan kesalahan, penentu didahulukan, dan tata bahasa yang agak ambigu.
3. SLY menggunakan berbagai fitur pemrograman Python untuk menentukan lexers dan parser. Tidak ada file yang dibuat atau langkah-

langkah tambahan yang terlibat. Anda cukup menulis kode Python dan menjalankannya.

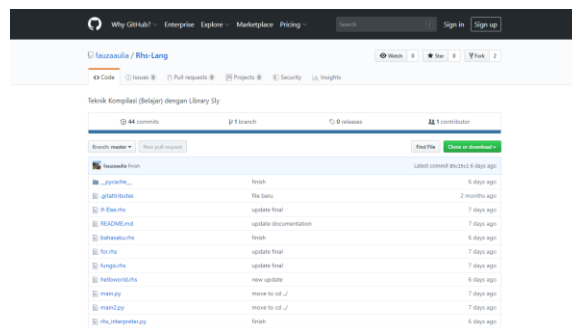
4. SLY dapat digunakan untuk membuat parser untuk bahasa pemrograman "nyata". Meskipun tidak terlalu cepat karena implementasi Python, SLY dapat digunakan untuk mengurai tata bahasa yang terdiri dari beberapa ratus aturan (seperti yang dapat ditemukan untuk bahasa seperti C).

2.2. ALAT DAN BAHAN

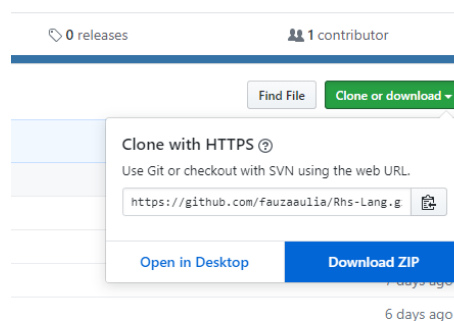
1. Personal Komputer (PC) / Laptop
2. Teks Editor (Visual Studio Code, Sublime, Notepad++, dll.)
3. Bahasa Python 3.7 atau yang terbaru
4. Command Prompt (CMD) / PowerShell Python

2.3. PROSES INSTALASI

1. Kunjungi repo github <https://github.com/fauzaaulia/Rhs-Lang>

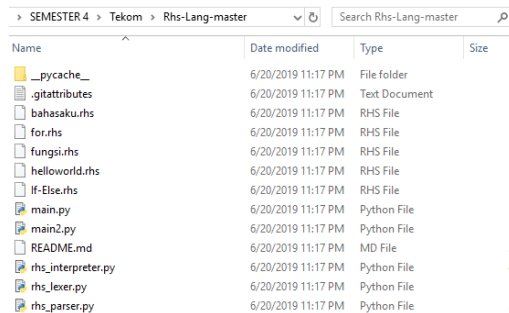


2. Lakukan download zip



3. Anda akan mendapatkan file **Rhs-Lang-master.zip**

4. Lakukan ekstrak file dan akan didapatkan folder **Rhs-Lang-master** yang berisi beberapa file (main.py , rhs_lexer.py, rhs_parser.py, rhs_interpreter.py, dll.)



Name	Date modified	Type	Size
__pycache__	6/20/2019 11:17 PM	File folder	
.gitattributes	6/20/2019 11:17 PM	Text Document	1
bahasaku.rhs	6/20/2019 11:17 PM	RHS File	1
for.rhs	6/20/2019 11:17 PM	RHS File	1
fungsi.rhs	6/20/2019 11:17 PM	RHS File	1
helloworld.rhs	6/20/2019 11:17 PM	RHS File	1
If-Else.rhs	6/20/2019 11:17 PM	RHS File	1
main.py	6/20/2019 11:17 PM	Python File	1
main2.py	6/20/2019 11:17 PM	Python File	1
README.md	6/20/2019 11:17 PM	MD File	3
rhs_interpreter.py	6/20/2019 11:17 PM	Python File	4
rhs_lexer.py	6/20/2019 11:17 PM	Python File	1
rhs_parser.py	6/20/2019 11:17 PM	Python File	3

5. Rhs-Lang berhasil di Install

2.4. DOKUMENTAS PEMBUATAN BAHASA PEMOGGRAMAN RHS-LANG

2.4.1. Pembuatan Lexer

Analisis Lexical (Lexer), adalah sebuah komponen yang berfungsi untuk mengkonversi atau menterjemahkan kumpulan-kumpulan karakter (yang biasa disebut dengan source code) ke dalam besaran - besaran token, sehingga dari token-token tersebut, kompilator bisa membuat keyword (kata kunci), operator, dan yang lainnya. Didalam bahasa Rhs-Lang penentuan karakter ke dalam token-token dapat dilihat dibawah ini :

```
class BasicLexer(Lexer):
    tokens = { NAME, NUMBER, STRING, IF, PRINT,
              THEN, ELSE, FOR, FUN, TO, ARROW, EQEQ }
    ignore = '\t '

    literals = { '=', '+', '-', '/', '*', '(', ')',
                ',', '; ' }

    # Define tokens
    IF = r'FI'
    PRINT = r'TNIRP'
    THEN = r'NEHT'
    ELSE = r'ESLE'
    FOR = r'ROF'
    FUN = r'NUF'
```

```

TO = r'OT'
ARROW = r'-'>'
NAME = r'[a-zA-Z_][a-zA-Z0-9_]*'
STRING = r'\".*?\"'

EQEQ = r'=='

```

2.4.2. Pembuatan Parser

Parser adalah komponen dari kompilator yang berfungsi untuk mengambil token-token yang telah dikonversi oleh lexer, sehingga parser dapat menciptakan sebuah pohon sintaks dari token-token tersebut. Penerjemahan dari token-token dalam lexer ditujukan pada sebuah perintah untuk token tersebut.

```

class BasicParser(Parser):
    tokens = rhs_lexer.BasicLexer.tokens

    precedence = (
        ('left', '+', '-'),
        ('left', '*', '/'),
        ('right', 'UMINUS'),
    )

    def __init__(self):
        self.env = { }

    @_( '')
    def statement(self, p):
        pass

    @_('FOR var_assign TO expr THEN statement')
    def statement(self, p):
        return ('for_loop', ('for_loop_setup',
                               p.var_assign, p.expr), p.statement)

    @_('IF condition THEN statement ELSE statement')
    def statement(self, p):
        return ('if_stmt', p.condition, ('branch',
                                           p.statement0, p.statement1))

    @_('FUN NAME "(" ")" ARROW statement')

```

```

def statement(self, p):
    return ('fun_def', p.NAME, p.statement)

@_('NAME "(" ")"')
def statement(self, p):
    return ('fun_call', p.NAME)

@_('expr EQEQ expr')
def condition(self, p):
    return ('condition_eqeq', p.expr0, p.expr1)

@_('var_assign')
def statement(self, p):
    return p.var_assign

@_('NAME "=" expr')
def var_assign(self, p):
    return ('var_assign', p.NAME, p.expr)

@_('NAME "=" STRING')
def var_assign(self, p):
    return ('var_assign', p.NAME, p.STRING)

@_('expr')
def statement(self, p):
    return (p.expr)

@_('expr "+" expr')
def expr(self, p):
    return ('add', p.expr0, p.expr1)

@_('expr "-" expr')
def expr(self, p):
    return ('sub', p.expr0, p.expr1)

@_('expr "*" expr')
def expr(self, p):
    return ('mul', p.expr0, p.expr1)

@_('expr "/" expr')
def expr(self, p):
    return ('div', p.expr0, p.expr1)

```

```

@_('"-" expr %prec UMINUS')
def expr(self, p):
    return p.expr

@_('NAME')
def expr(self, p):
    return ('var', p.NAME)

@_('NUMBER')
def expr(self, p):
    return ('num', p.NUMBER)

@_('PRINT expr')
def expr(self, p):
    return ('print', p.expr)

@_('PRINT STRING')
def statement(self, p):
    return ('print', p.STRING)

```

2.4.3. Pembuatan Interpreter

Interpreter adalah perangkat lunak yang mampu mengeksekusi code program lalu menterjemahkannya ke dalam bahasa mesin, sehingga mesin melakukan instruksi yang diminta oleh programmer. Perintah-perintah yang dibuat oleh programmer akan dieksekusi baris demi baris, sambil mengikuti logika yang terdapat di dalam kode tersebut. Proses ini sangat berbeda dengan compiler, dimana pada compiler, hasilnya sudah langsung berupa satu kesatuan perintah dalam bentuk bahasa mesin, dimana proses penterjemahan dilaksanakan sebelum program tersebut dieksekusi.

```

class BasicExecute:

    def __init__(self, tree, env):
        self.env = env
        result = self.walkTree(tree)
        if result is not None and isinstance(result,
            int):
            print(result)

```

```

        if isinstance(result, str) and result[0] ==
        '':
            print(result)

def walkTree(self, node):

    if isinstance(node, int):
        return node
    if isinstance(node, str):
        return node

    if node is None:
        return None

    if node[0] == 'program':
        if node[1] == None:
            self.walkTree(node[2])
        else:
            self.walkTree(node[1])
            self.walkTree(node[2])

    if node[0] == 'num':
        return node[1]

    if node[0] == 'str':
        return node[1]

    if node[0] == 'print':
        if node[1][0] == '':
            print(node[1][1:len(node[1])-1])
        else:
            return self.walkTree(node[1])

    if node[0] == 'if_stmt':
        result = self.walkTree(node[1])
        if result:
            return self.walkTree(node[2][1])
        return self.walkTree(node[2][2])

    if node[0] == 'condition_eqeq':
        return self.walkTree(node[1]) ==
        self.walkTree(node[2])

```

```

if node[0] == 'fun_def':
    self.env[node[1]] = node[2]

if node[0] == 'fun_call':
    try:
        return
        self.walkTree(self.env[node[1]])
    except LookupError:
        print("Undefined function '%s'" %
              node[1])
        return 0

if node[0] == 'add':
    return self.walkTree(node[1]) +
           self.walkTree(node[2])
elif node[0] == 'sub':
    return self.walkTree(node[1]) -
           self.walkTree(node[2])
elif node[0] == 'mul':
    return self.walkTree(node[1]) *
           self.walkTree(node[2])
elif node[0] == 'div':
    return int(self.walkTree(node[1]) /
              self.walkTree(node[2]))

if node[0] == 'var_assign':
    self.env[node[1]] =
    self.walkTree(node[2])
    return node[1]

if node[0] == 'var':
    try:
        return self.env[node[1]]
    except LookupError:
        print("Undefined variable
              '"+node[1]+"' found!")
        return 0

if node[0] == 'for_loop':
    if node[1][0] == 'for_loop_setup':
        loop_setup = self.walkTree(node[1])

        loop_count = self.env[loop_setup[0]]

```

```

        loop_limit = loop_setup[1]

        for i in range(loop_count+1,
loop_limit+1):
            res = self.walkTree(node[2])
            if res is not None:
                print(res)
                self.env[loop_setup[0]] = i
            del self.env[loop_setup[0]]

        if node[0] == 'for_loop_setup':
            return (self.walkTree(node[1]),
                self.walkTree(node[2]))

```

2.4.4. Pembuatan Main Utama

File main berfungsi sebagai jembatan dari file rhs-lang yang ber-ekstensi '.rhs' untuk kemudia di eksekusi oleh lexer, parser, dan interpreter.

```

#DENGAN MASUKAN BAHASAKU.RHS
lexer = rhs_lexer.BasicLexer()
parser = rhs_parser.BasicParser()
env = {}

file = open(argv[1])
text = file.readlines()
for line in text:
    tree = parser.parse(lexer.tokenize(line))
    rhs_interpreter.BasicExecute(tree, env)

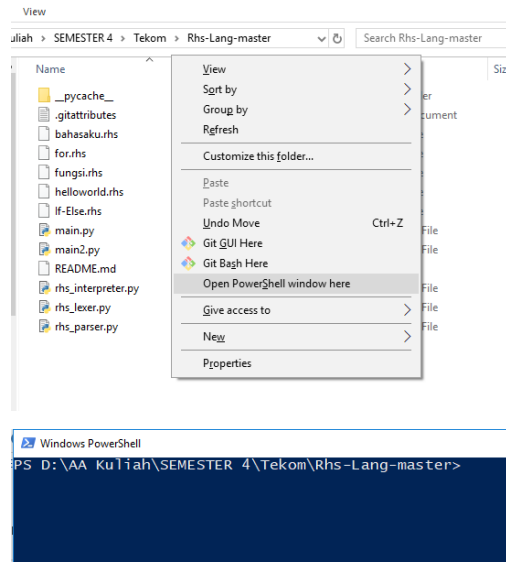
```

2.5. CARA PENGGUNAAN

2.5.1. Menggunakan PowerShell

Cara menggunakannya sebagai berikut:

1. Buka PowerShell dengan direktori ini direktori folder cd: ... / Rhs-
Lang-master / (Shift+klik kanan pada folder direktori) pilih Open
PowerShell window here



2. Dan kemudian jalankan file main.py di folder Rhs-Lang-master

Contoh:

Copy dan pastekan teks ini ke powershell anda

```
python .main.py .helloworld.rhs
```

```
Windows PowerShell
PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master> python .\main.py .\helloworld.rhs
WARNING: 4 shift/reduce conflicts
Hello World!!
"Siap 86!"
PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master>
```

2.5.2. Menggunakan CMD

Cara menggunakannya sebagai berikut:

1. Buka CMD atau TERMINAL dengan direktori ini direktori folder cd:

```
... / Rhs-Lang-master /
```

2. Dan kemudian jalankan file main.py di folder Rhs-Lang-master

Contoh:

Copy dan pastekan teks ini ke cmd atau terminal anda

```
python .main.py .helloworld.rhs
```

```
Command Prompt
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Asus>cd
D:\>cd AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master

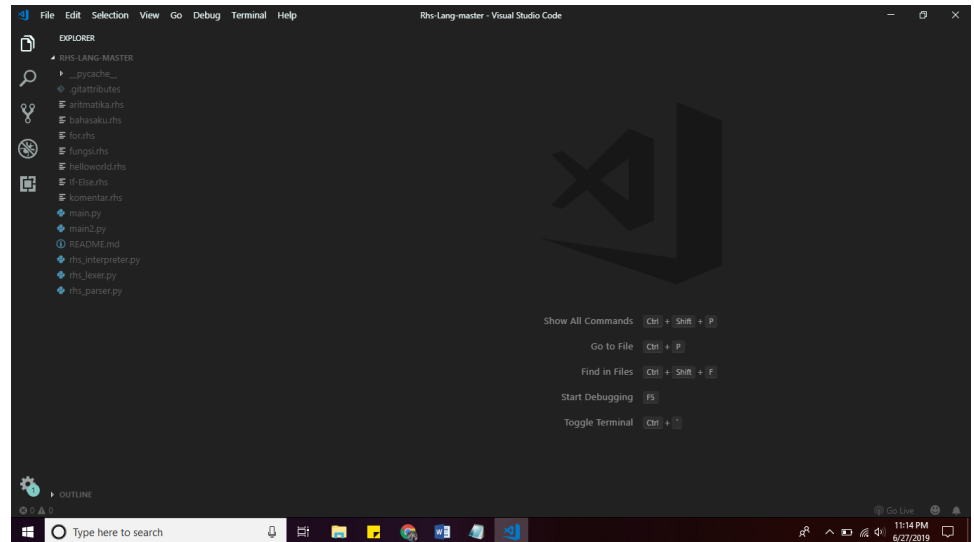
D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master>python main.py helloworld.rhs
WARNING: 4 shift/reduce conflicts
Hello World!!
"Siap 86!"

D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master>
```

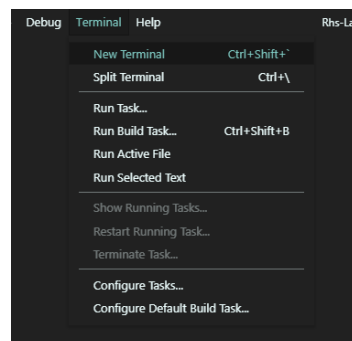

2.5.3. Menggunakan PowerShell di Visual Studio Code

Cara menggunakannya sebagai berikut:

1. Buka folder Rhs-Lang-master ke teks editor visual studio code.



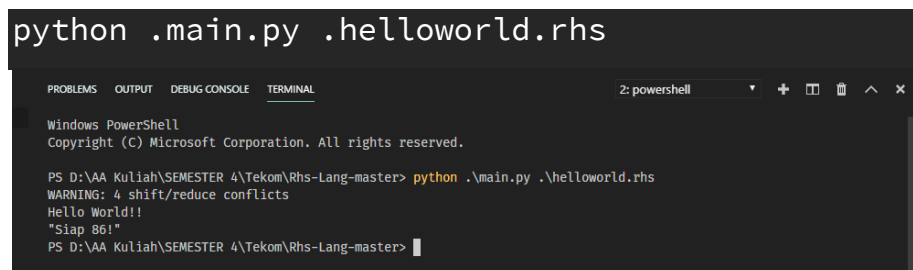
2. Buka terminal dengan cara klik tab **terminal-> New Terminal** atau dapat menggunakan shortcut **Ctrl+Shift+`**



3. Dan kemudian jalankan file main.py di folder Rhs-Lang-master

Contoh:

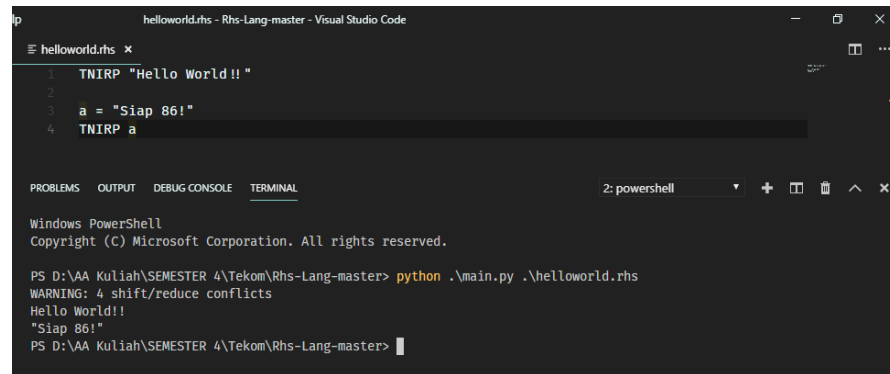
Copy dan pastekan teks ini ke cmd atau terminal anda



2.6. BAHASA PERINTAH RHS-LANG

2.6.1. Perintah “TNIRP”

Perintah TNIRP berartikan PRINT (cetak) atau perintah yang digunakan untuk mencetak sebuah variable atau nilai luaran. Dengan menggunakan perintah TNIRP maka hasil/result akan di tampilkan pada layar antarmuka pengguna.



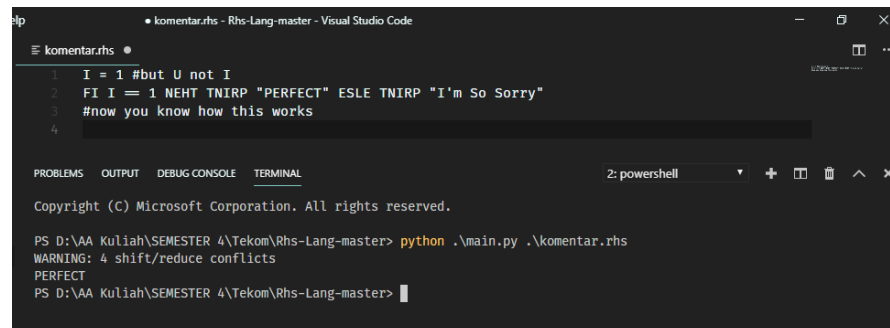
```
helloworld.rhs - Rhs-Lang-master - Visual Studio Code
1 TNIRP "Hello World!! "
2
3 a = "Siap 86!"
4 TNIRP a

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master> python .\main.py .\helloworld.rhs
WARNING: 4 shift/reduce conflicts
Hello World!!
"Siap 86!"
PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master>
```

2.6.2. Penggunaan Comment (Komentar)

Dalam bahasa pemrogramman Rhs-Lang penggunaan comment atau komentar hanya dengan menambahkan tanda ‘#’ (pagar/hastag) di awal kalimat.



```
komentar.rhs - Rhs-Lang-master - Visual Studio Code
1 I = 1 #but U not I
2 FI I = 1 NEHT TNIRP "PERFECT" ESLE TNIRP "I'm So Sorry"
3 #now you know how this works
4

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: powershell
Copyright (C) Microsoft Corporation. All rights reserved.

PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master> python .\main.py .\komentar.rhs
WARNING: 4 shift/reduce conflicts
PERFECT
PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master>
```

2.6.3. Penggunaan Operasi Aritmatika

Dalam bahasa pemrogramman Rhs-Lang penggunaan operasi perhitungan aritmatika (penjumlahan, pengurangan, perkalian, pembagian) dapat langsung menulis perintah tanpa harus ada penulisan token sebagai identifier atau dengan tambahan perintah TNIRP. Untuk operasi pembagian hanya dapat berupa data integer.

```

1  3+2
2  4-1
3  3*3
4  4/2
5
6  TNIRP "atau seperti ini"
7
8  a=6
9  b=2
10 TNIRP a+b
11 TNIRP a-b
12 TNIRP a+b
13 TNIRP a/b
14

```

```

PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master> python .\main.py .\aritmatika.rhs
WARNING: 4 shift/reduce conflicts
5
3
9
2
atau seperti ini
8
4
12
3
PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master>

```

2.6.4. Perintah “FI *expr* NEHT *stmt1* ESLE *stmt2*”

Pada perintah FI *expr* NEHT *stmt1* ESLE *stmt2* dibutuhkan *expression* dan 2 *statement*.

- FI adalah sintaksis atau perintah untuk menyatakan logika IF
- NEHT adalah sintaksis atau perintah untuk menyatakan logika THEN
- ESLE adalah sintaksis atau perintah untuk menyatakan logika ELSE

```

1  a=6
2  y="true"
3  n="false"
4  FI a=6 NEHT TNIRP y ESLE TNIRP n

```

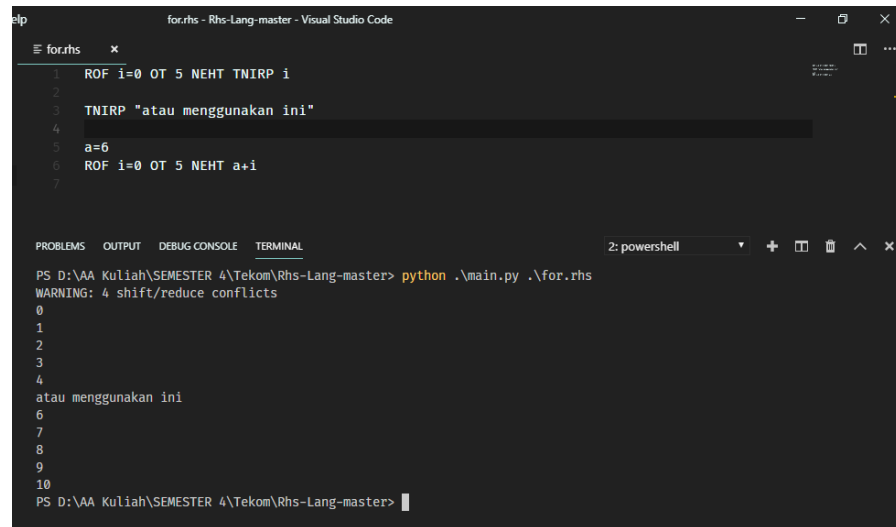
```

PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master> python .\main.py .\If-Else.rhs
WARNING: 4 shift/reduce conflicts
true
PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master>

```

2.6.5. Perintah “ROF *expr* OT *stmt1* NEHT *stmt2*”

- ROF adalah sintaksis atau perintah untuk menyatakan logika FOR
- OT adalah sintaksis atau perintah untuk menyatakan logika TO



```
for.rhs
1  ROF i=0 OT 5 NEHT TNIRP i
2
3  TNIRP "atau menggunakan ini"
4
5  a=6
6  ROF i=0 OT 5 NEHT a+i
7

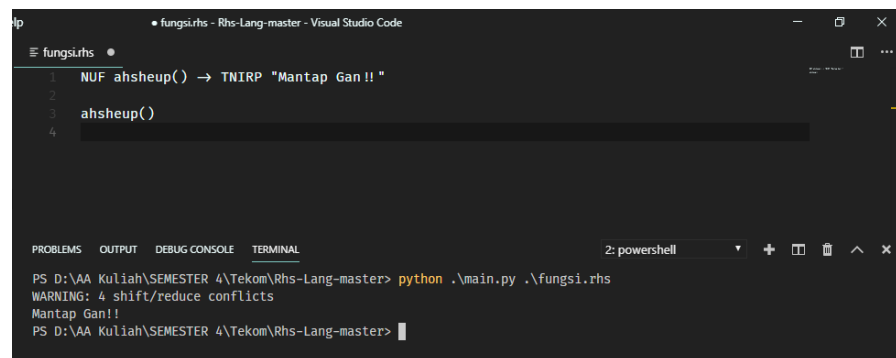
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: powershell
PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master> python .\main.py .\for.rhs
WARNING: 4 shift/reduce conflicts
0
1
2
3
4
atau menggunakan ini
6
7
8
9
10
PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master>
```

2.6.6. Perintah “NUF functionName() -> Kode kamu...”

Sintaksis NUF merupakan perintah untuk fungsi. “Fungsi” adalah sintaksis atau perintah untuk membuat suatu fungsi dengan parameter di dalamnya.

- NUF adalah sintaksis atau perintah untuk menyatakan logika FUN
- functionName() merupakan nama fungsi yang akan dibuat

Contoh penggunaannya : “NUF functionName() -> Kode kamu...”



```
fungsi.rhs
1  NUF ahsheup() -> TNIRP "Mantap Gan!! "
2
3  ahsheup()
4

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: powershell
PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master> python .\main.py .\fungsi.rhs
WARNING: 4 shift/reduce conflicts
Mantap Gan!!
PS D:\AA Kuliah\SEMESTER 4\Tekom\Rhs-Lang-master>
```

BAB III

PENUTUP

3.1. AKHIR KATA

Selama pembuatan bahasa pemrograman Rhs-Lang ini kami menyadari masih diperlukannya pengembangan serta bimbingan supaya penulisan dapat tepat guna dan mudah digunakan. Kami sangat menerima saran dan masukan demi berkembangnya bahasa pemrograman dan Modul Teknik Kompilasi ini. Supaya dapat diperbaiki dalam kesempatan berikutnya. Semoga Modul Teknik Kompilasi ini dapat berguna dan bermanfaat bagi pembaca.

LAMPIRAN-LAMPIRAN

Lampiran 1. *rhs_lexer.py*

```
from sly import Lexer

class BasicLexer(Lexer):
    tokens = { NAME, NUMBER, STRING, IF, PRINT, THEN, ELSE, FOR,
FUN, TO, ARROW, EQEQ }
    ignore = '\t '

    literals = { '=', '+', '-', '/', '*', '(', ')', ',', ';' }

    # Define tokens
    IF = r'FI'
    PRINT = r'TNIRP'
    THEN = r'NEHT'
    ELSE = r'ESLE'
    FOR = r'ROF'
    FUN = r'NUF'
    TO = r'OT'
    ARROW = r'->'
    NAME = r'[a-zA-Z_][a-zA-Z0-9_]*'
    STRING = r'\".*?\"'

    EQEQ = r'=='

    @_(r'\d+')
    def NUMBER(self, t):
        t.value = int(t.value)
        return t

    @_(r'#.*')
    def COMMENT(self, t):
        pass

    @_(r'\n+')
    def newline(self, t):
        self.lineno = t.value.count('\n')

if __name__ == '__main__':
    lexer = BasicLexer()
```

```
env = {}  
while True:  
    try:  
        text = input('rhs > ')  
    except EOFError:  
        break  
    if text:  
        lex = lexer.tokenize(text)  
        for token in lex:  
            print(token)
```

Lampiran 2. *rhs_parser.py*

```
from sly import Parser

import rhs_lexer

class BasicParser(Parser):
    tokens = rhs_lexer.BasicLexer.tokens

    precedence = (
        ('left', '+', '-'),
        ('left', '*', '/'),
        ('right', 'UMINUS'),
    )

    def __init__(self):
        self.env = { }

    @_( '')
    def statement(self, p):
        pass

    @_('FOR var_assign TO expr THEN statement')
    def statement(self, p):
        return ('for_loop', ('for_loop_setup', p.var_assign,
p.expr), p.statement)

    @_('IF condition THEN statement ELSE statement')
    def statement(self, p):
        return ('if_stmt', p.condition, ('branch', p.statement0,
p.statement1))

    @_('FUN NAME "(" ")" ARROW statement')
    def statement(self, p):
        return ('fun_def', p.NAME, p.statement)

    @_('NAME "(" ")"')
    def statement(self, p):
        return ('fun_call', p.NAME)

    @_('expr EQEQ expr')
    def condition(self, p):
        return ('condition_eqeq', p.expr0, p.expr1)
```



```

@_('var_assign')
def statement(self, p):
    return p.var_assign

@_('NAME "=" expr')
def var_assign(self, p):
    return ('var_assign', p.NAME, p.expr)

@_('NAME "=" STRING')
def var_assign(self, p):
    return ('var_assign', p.NAME, p.STRING)

@_('expr')
def statement(self, p):
    return (p.expr)

@_('expr "+" expr')
def expr(self, p):
    return ('add', p.expr0, p.expr1)

@_('expr "-" expr')
def expr(self, p):
    return ('sub', p.expr0, p.expr1)

@_('expr "*" expr')
def expr(self, p):
    return ('mul', p.expr0, p.expr1)

@_('expr "/" expr')
def expr(self, p):
    return ('div', p.expr0, p.expr1)

@_('"-" expr %prec UMINUS')
def expr(self, p):
    return p.expr

@_('NAME')
def expr(self, p):
    return ('var', p.NAME)

@_('NUMBER')
def expr(self, p):
    return ('num', p.NUMBER)

```

```

@_('PRINT expr')
def expr(self, p):
    return ('print', p.expr)

@_('PRINT STRING')
def statement(self, p):
    return ('print', p.STRING)

if __name__ == '__main__':
    lexer = rhs_lexer.BasicLexer()
    parser = BasicParser()
    env = {}
    while True:
        try:
            text = input('rhs > ')
        except EOFError:
            break
        if text:
            tree = parser.parse(lexer.tokenize(text))
            print(tree)

```

Lampiran 3. *rhs_interpreter.py*

```
import rhs_lexer
import rhs_parser

class BasicExecute:

    def __init__(self, tree, env):
        self.env = env
        result = self.walkTree(tree)
        if result is not None and isinstance(result, int):
            print(result)
        if isinstance(result, str) and result[0] == '':
            print(result)

    def walkTree(self, node):

        if isinstance(node, int):
            return node
        if isinstance(node, str):
            return node

        if node is None:
            return None

        if node[0] == 'program':
            if node[1] == None:
                self.walkTree(node[2])
            else:
                self.walkTree(node[1])
                self.walkTree(node[2])

        if node[0] == 'num':
            return node[1]

        if node[0] == 'str':
            return node[1]

        if node[0] == 'print':
            if node[1][0] == '':
                print(node[1][1:len(node[1])-1])
            else:
                return self.walkTree(node[1])
```

```

if node[0] == 'if_stmt':
    result = self.walkTree(node[1])
    if result:
        return self.walkTree(node[2][1])
    return self.walkTree(node[2][2])

if node[0] == 'condition_eqeq':
    return self.walkTree(node[1]) == self.walkTree(node[2])

if node[0] == 'fun_def':
    self.env[node[1]] = node[2]

if node[0] == 'fun_call':
    try:
        return self.walkTree(self.env[node[1]])
    except LookupError:
        print("Undefined function '%s'" % node[1])
        return 0

if node[0] == 'add':
    return self.walkTree(node[1]) + self.walkTree(node[2])
elif node[0] == 'sub':
    return self.walkTree(node[1]) - self.walkTree(node[2])
elif node[0] == 'mul':
    return self.walkTree(node[1]) * self.walkTree(node[2])
elif node[0] == 'div':
    return int(self.walkTree(node[1]) /
self.walkTree(node[2]))

if node[0] == 'var_assign':
    self.env[node[1]] = self.walkTree(node[2])
    return node[1]

if node[0] == 'var':
    try:
        return self.env[node[1]]
    except LookupError:
        print("Undefined variable '"+node[1]+' found!")
        return 0

if node[0] == 'for_loop':
    if node[1][0] == 'for_loop_setup':

```

```

        loop_setup = self.walkTree(node[1])

        loop_count = self.env[loop_setup[0]]
        loop_limit = loop_setup[1]

        for i in range(loop_count+1, loop_limit+1):
            res = self.walkTree(node[2])
            if res is not None:
                print(res)
                self.env[loop_setup[0]] = i
            del self.env[loop_setup[0]]

    if node[0] == 'for_loop_setup':
        return (self.walkTree(node[1]), self.walkTree(node[2]))

if __name__ == '__main__':
    lexer = rhs_lexer.BasicLexer()
    parser = rhs_parser.BasicParser()
    env = {}
    while True:
        try:
            text = input('rhs > ')
        except EOFError:
            break
    if text:
        tree = parser.parse(lexer.tokenize(text))
        BasicExecute(tree, env)

```

Lampiran 3. *main.py*

```
import rhs_lexer
import rhs_parser
import rhs_interpreter

from sys import *

#DENGAN MASUKAN BAHASAKU.RHS
lexer = rhs_lexer.BasicLexer()
parser = rhs_parser.BasicParser()
env = {}

file = open(argv[1])
text = file.readlines()
for line in text:
    tree = parser.parse(lexer.tokenize(line))
    rhs_interpreter.BasicExecute(tree, env)
```