UFPI – CCN – DC
Curso: Bacharelado em Ciência da Computação
Disciplina: Compiladores **Período: 2021-2**
Prof. Dr. Raimundo Moura

O projeto

Desenvolver um compilador de um subconjunto básico da linguagem Python.

Composição

O projeto está dividido em três partes e as entregas deverão seguir o calendário disponível no site da disciplina.

- Primeira parte: análise sintática (+léxica), com geração da AST – dia 13/04/2022;
- Segunda parte: análise sintática com análise semântica – dia 28/04/2022;
- Terceira parte: geração de código – dia 12/05/2022.

Linguagem utilizada

A linguagem adotada para a implementação do compilador fica a critério da equipe, podendo ser Java, Rubi, Python, C# ou C/C++. Outras opções de linguagem devem ser discutidas com o professor da disciplina.

Linguagem fonte

A linguagem fonte do projeto será Python Simplificado, que apresenta diversas limitações e possui somente fins acadêmicos.

Python Simplificado será especificada informalmente neste documento. O primeiro desafio do projeto é transformar esta descrição em uma especificação formal, através de uma gramática livre de contexto (GLC). O uso de colchetes nas descrições de sintaxe indica que o termo é opcional.

Linguagem alvo

O compilador deverá traduzir programas escritos em Python Simplificado para linguagem de máquina, usando como linguagem intermediária: código executável da máquina virtual Java (JVM);

Ferramentas

Para o desenvolvimento do projeto, ferramentas como o ANTLR, Flex/Bison e Lex/Yacc podem ser utilizadas para a geração automática de *scanners* e *parsers*.

Para a geração do código de máquina, deverá ser usado *JASMIN – tradutor de linguagem intermediária para bytecode java*.

Python Simplificado

Python Simplificado é sensível (*case-sensitive*) em relação a letras maiúsculas e minúsculas nos nomes das variáveis, funções e palavras reservadas. (i.e. “variable”, “VARIABLE” e “VaRiAbLe” são identificadores diferentes).

A linguagem não deve suportar declarações aninhadas de funções, mas pode conter várias chamadas recursivas. A linguagem possui funções *built-in*, utilizadas para realizar IO (***print*** e ***input***) de usuário através do console, que serão a única forma de se comunicar com o mundo exterior.

Nenhuma característica de outra linguagem deve ser reconhecida pelo compilador.

Tipos e operadores

Python Simplificado é fortemente tipado e dá suporte aos seguintes tipos: inteiros, reais strings e booleanos. Os únicos operadores disponíveis na linguagem são:

- ‘not’: negação, inverte o valor booleano ao qual foi aplicada – unário;
- ‘-’: menos unário, inverte o valor inteiro ou real ao qual foi aplicada – unário;
- ‘and’: AND lógico – binário;
- ‘or’: OR lógico – binário;
- ‘+’: soma – binário;
- ‘-‘: subtração – binário;
- ‘*’: multiplicação – binário;
- ‘/’: divisão inteira e real – binário;
- ‘==’: comparação, checa se os operadores são iguais – binário;
- ‘!=’: comparação, checa se os operadores são diferentes – binário;
- ‘>=’: maior ou igual que – binário;
- ‘<=’: menor ou igual que – binário;
- ‘>’: maior que – binário;
- ‘<’: menor que – binário;

Precedência de operadores

I. Quando dois operadores diferentes forem utilizados ao mesmo tempo, o operador de maior precedência será avaliado primeiro;
II. Operadores de mesma precedência serão avaliados da esquerda para a direita;
III. É possível forçar a precedência de uma operação, colocando-a entre parênteses.

A tabela a seguir apresenta a hierarquia de precedência dos operadores (decrescente de cima para baixo), os tipos dos operandos aos quais podem ser aplicados e o tipo do retorno:

Tabela 1. Precedência de operadores.

Operador	Aridade	Tipos aplicados	Tipo de retorno
not	1	Booleano	booleano
-	1	Inteiro e real	Inteiro e real
* e /	2	Inteiro e real	Inteiro e real
+ e -	2	Inteiro e real	Inteiro e real
== e !=	2	booleano, inteiro, real e string	booleano
>=, <=, > e <	2	booleano, inteiro, real e string	booleano
and	2	Booleano	booleano
or	2	Booleano	booleano

Identificadores

São os nomes utilizados para identificar variáveis e funções no programa. Identificadores podem ter qualquer tamanho maior que zero e devem obrigatoriamente começar com uma letra. Após isso, podem conter qualquer letra, digito ou o caractere *underscore*.

Dentro do mesmo escopo, variáveis não podem ter o mesmo nome de outras variáveis ou funções. Identificadores nunca podem ter o mesmo nome de uma palavra reservada da linguagem (***if***, ***for***, **while**, etc.).

Variáveis e Constantes

Variáveis são posições de memória que guardarão dados do programa. Em PYTHON Simplificado, toda variável tem o tipo definido na sua declaração, que não poderá ser alterado.

Para declarar uma variável, atribui-se um tipo a um identificador válido. O valor padrão da variável definida desta forma será 0 (zero) para inteiros e reais, *False* para as variáveis do tipo booleano e string vazia para strings.

Sintaxe de declaração:

<tipo> <lista_de_identificadores>|<lista_de_atribuições> ;

Ex:

```
int INT, FLOAT, inteiro1;
string string1="123", string2="456";
boolean flag1;
float x=0;
```

Como se pode observar, declarações sequenciais são permitidas, em uma mesma linha, porém apenas quando as variáveis forem de um mesmo tipo.

Os valores podem ser atribuídos na própria da declaração das variáveis. Para isso deve-se usar a atribuição '='. Strings devem estar entre aspas duplas, como pode ser visto nos exemplos abaixo.

Ex:

```
string inst="UFPI";
...
inst = "Teste";
```

Python Simplificado não possui constantes.

Escopo das variáveis

Variáveis podem ser:

1. **Globais**: conhecidas por todo o programa. Devem ser declaradas no início do programa.

Erros de compilação devem ser gerados caso uma variável seja declarada após o início do bloco ou após o início do programa.

Funções

Funções têm um nome, uma lista de parâmetros e um tipo de retorno. Nos casos em que não se deve retornar nenhum valor, usa-se o tipo ***void***.

Nenhuma função poderá ter o mesmo nome que outra função ou variável global, independente da sua assinatura.

Funções podem ser chamadas recursivamente. Elas possuem uma lista com zero ou mais parâmetros e todo parâmetro é passado por valor.

Para definir o valor a ser retornado, utiliza-se o comando ***return***.

Sintaxe de declaração (função):

```
def <tipo><nome_funcao> ( <lista_parametros> ) :
   ...
   return <valor_retorno>;
}
```

Para funções do tipo ***void*** o comando ***return*** não deve ser usado.

Funções nativas

O compilador deverá reconhecer previamente um conjunto de procedimentos nativos para implementação de IO do usuário.

PRINT

Envia saída para o console. A cada chamada também é enviada uma quebra de linha, exceto entre os itens de ***print (lista de expressões)***. A lista de expressões deve ser separada por vírgulas.

Sintaxe de chamada:

print <lista_expressao>;

Ex:

```
print "Hello", "World!";
print "1 + 1 = ", 1+1;
print soma; # imprime o valor da variável soma
```

Saída:

```
HelloWorld!
1 + 1 = 2
145 # considerar que soma = 145
```

INPUT

Requisita a entrada de números inteiros, reais e strings a partir do teclado. O valor fornecido é armazenado na variável usada na chamada do ***input***.

O comando de leitura interrompe a execução do programa até que o usuário digite um valor (o valor é confirmado após pressionar a tecla "<Enter>").

Sintaxe de chamada:

<variável> = ***input***();

Ex:

```
print “Digite um número”;
a = input();
print “Digite dois números”;
b = input();
c = input();
print “a = ”, a, “; b = ”, b, “; c = ”, c;
```

Saída:

```
Digite um número
?1
Digite dois números
?2 3
a = 1; b = 2; c = 3
```

Comandos de controle

O compilador deve reconhecer os seguintes comandos de controle de execução: if-else, for, while.

IF-ELSE

Sintaxe:

```
if <teste_logico>:
  ...

[} else :
  ...
 }]
```

FOR

Sintaxe:

```
for <variável> in range([start:]stop[step]):
  ...
  [break]
  ...
}
```

O comando ***break*** (opcional) interrompe a iteração do laço e transfere o controle para o final da construção do ***for***.

WHILE

Sintaxe:

```
while (<teste_logico>):
  …
  [break]
  …
}
```

Programa Principal

Um programa é composto por zero ou mais declarações de variáveis globais, zero ou mais declarações de funções e um bloco principal (função ***main***) com início logo depois do bloco de funções, sendo finalizado com o símbolo fecha chaves.

Todo programa deve ser escrito em um arquivo de texto sem formatação com a extensão ".py".

Ex:

```
int numero;

def int fatorial (int fat):
  if fat > 1:
    print fat;
    return fat * fatorial(fat - 1);
  } else:
    return 1;
  }
}

def void resultado (int valor):
  print "Resultado: ", valor;
}

main():
  print "Fatorial de N. Digite o número?";
  numero = input();
  resultado (fatorial (numero));
}
```