Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №5

По теме «Интерпретатор»

Выполнил:
студент гр. 953501
Кореневский С. А.
Проверил:
ст. преподаватель Шиманский В. В.

Минск 2022

Содержание

1. Постановка задачи

В данной работе ставится задача полной реализации интерпретатора языка C++ на объектно-ориентированном языке программирования Python. Основной целью работы является реализация виртуальной машины, способной обрабатывать команды языка C++ и выдавать результат при исполнении, то есть быть интерпретатором. Это является последним шагом на пути в создании программы, способной выполнять программу самостоятельно.

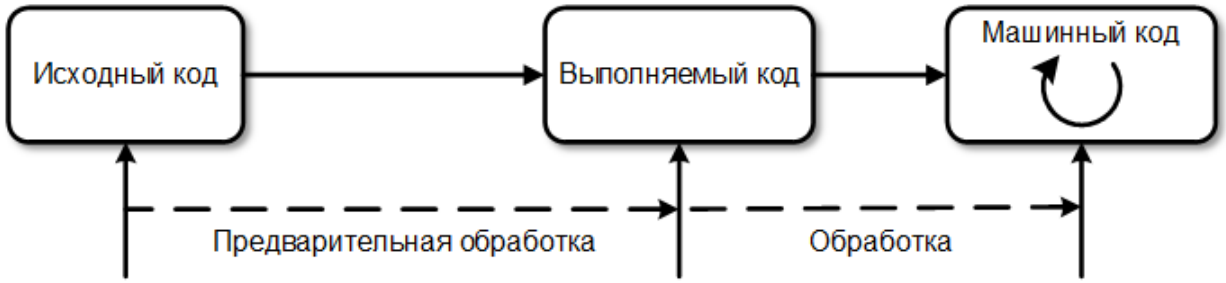
2. Теория

Компилятор и интерпретатор

Как Компилятор так и Интерпретатор имеют одно предназначение — конвертировать инструкции языка высокого уровня (как C или Java) в бинарную форму, понятную компьютеру. Это программное обеспечение, используемое для запуска высокоуровневых программ и кодов выполняемых различные задачи. Для разных высокоуровневых языков разработаны специфичные компиляторы/интерпретаторы. Не смотря на то что как компилятор так и интерпретатор преследуют одну и ту же цель, они отличаются способом выполнения своей задачи, то есть конвертирования высокоуровневого языка в машинные инструкции. В этой статье мы поговорим о базовой работе обоих и выделим главные отличия между компилятором и интерпретатором.

Компилятор

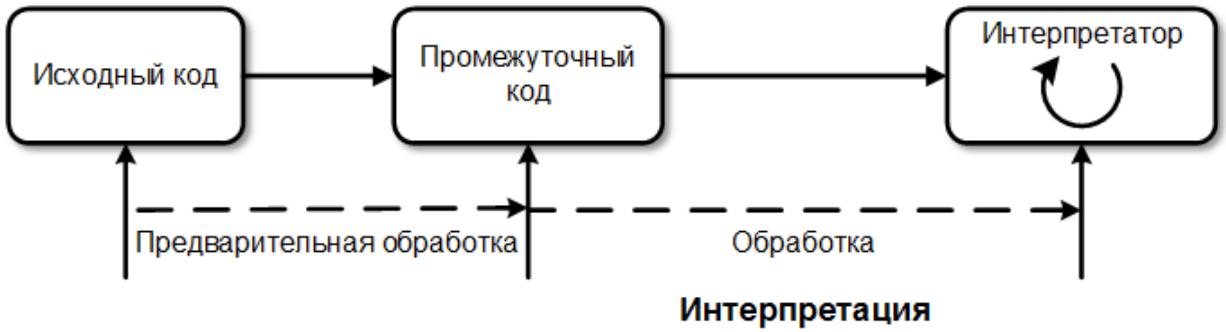
Компилятор транслирует высокоуровневый язык в машинный. Когда пользователь пишет код на языке высокого уровня, таком как Java, и хочет его выполнить, то прежде чем это может быть сделано, будет использован специальный компилятор разработанный для Java. Компилятор сначала сканирует всю программу, а потом транслирует ее в машинный код, который будет выполнен компьютерным процессором, после чего будут выполнены соответствующие задачи.



На картинке показано базовое очертание процесса компиляции. Тут программа написанная на языке высокого уровня показана как «Исходный код», а сконвертированный называется «Исполняемый код».

Интерпретатор

Интерпретаторы не очень сильно отличаются от компиляторов. Они также конвертируют высокоуровневые языки в читаемые машиной бинарные эквиваленты. Каждый раз когда интерпретатор получает на выполнение код языка высокого уровня, то прежде чем сконвертировать его в машинный код, он конвертирует этот код в промежуточный язык. Каждая часть кода интерпретируется и выполняется отдельно и последовательно, и если в какой-то части будет найдена ошибка, она остановит интерпретацию кода без трансляции следующей части кода.



Очертание процесса интерпретации на картинке выше показывает, что сначала исходный код конвертируется в промежуточную форму, а затем выполняется интерпретатором.

Ниже перечислены главные отличия между компилятором и интерпретатором:

- Интерпретатор берет одну инструкцию, транслирует и выполняет ее, а затем берет следующую инструкцию. Компилятор же транслирует всю программу сразу, а потом выполняет ее.
- Компилятор генерирует отчет об ошибках после трансляции всего, в то время как интерпретатор прекратит трансляцию после первой найденной ошибки.
- Компилятор по сравнению с интерпретатором требует больше времени для анализа и обработки языка высокого уровня.
- Помимо времени на обработку и анализ, общее время выполнения кода компилятора быстрее в сравнении с интерпретатором.

Транслятор

Трансля́тор — программа или техническое средство, выполняющее трансляцию программы.

Трансля́ция програ́ммы — преобразование программы, представленной на одном из языков программирования, в программу на другом языке. Транслятор обычно выполняет

также диагностику ошибок, формирует словари идентификаторов, выдаёт для печати текст программы и т. д.

Язык, на котором представлена входная программа, называется исходным языком, а сама программа — исходным кодом. Выходной язык называется целевым языком.

В общем случае понятие трансляции относится не только к языкам программирования, но и к другим языкам — как формальным компьютерным (вроде языков разметки типа HTML), так и естественным (русскому, английскому и т. п.).

Виды трансляторов

Существует несколько видов трансляторов.

Диалоговый транслятор — транслятор, обеспечивающий использование языка программирования в режиме разделения времени.

Синтаксически-ориентированный (синтаксически-управляемый) транслятор — транслятор, получающий на вход описание синтаксиса и семантики языка, текст на описанном языке и выполняющий трансляцию в соответствии с заданным описанием.

Однопроходной транслятор — транслятор, преобразующий исходный код при его однократном последовательном чтении (за один проход).

Многопроходной транслятор — транслятор, преобразующий исходный код после его нескольких чтений (за несколько проходов).

Оптимизирующий транслятор — транслятор, выполняющий оптимизацию создаваемого кода.

Тестовый транслятор — транслятор, получающий на вход исходный код и выдающий на выходе изменённый исходный код. Запускается перед основным транслятором для добавления в исходный код отладочных процедур. Например, транслятор с языка ассемблера может выполнять замену макрокоманд на код.

Обратный транслятор — транслятор, выполняющий преобразование машинного кода в текст на каком-либо языке программирования.

3. Результат работы интерпретатора

2.1 Пример 1

```
void SortAlgo::mergeSort(int data[], int lenD)
{
  if (lenD>1){
    int middle = lenD/2;
    int rem = lenD-middle;
    int *L = new int [middle];
    int *R = new int [rem];
    for (int i=0;i<lenD;i++){
      if (i<middle){
        L[i] = data[i];
      }
      elsee {
        R[i-middle] = data[i];
      }
    }
    mergeSort(L,middle);
    mergeSort(R,rem);
    merge(data, lenD, L, middle, R, rem);
  }
}
```

```
voi SortAlgo::merge(int merged[], int lenD, int L[], int lenL, int R[], int lenR){
  inttt i = 0;
  int j = 0;
  while123(i<lenL||j<lenR){
    iff (i<lenL & j<lenR){
      if (L[i]<=R[j]){
        merged[i+j] === L[i];
        i++;
      }
      else {
        merged[[i+j] = R[j];
        j++;
      }
    }
    else if (i<lenL){
      merged[i+j] = L[i];
      i++;
    }
    elses if (j<lenR){
      merged[i+j] = R[j];
      j++;
    }
  }
}
```

```
Enter number of lab : (2,3,4,5, 0-to exit)
5
result of example 1 arr = [78, 41, 4, 27, 3, 27, 8, 39, 19, 34, 6, 41, 13, 52, 16]
[3, 4, 6, 8, 13, 16, 19, 27, 27, 34, 39, 41, 41, 52, 78]
Enter number of lab : (2,3,4,5, 0-to exit)
```

Рис 1 Результат выполнения с данными 1

```
Enter number of lab : (2,3,4,5, 0-to exit)
5
result of example 2 arr = [-9, -98, 234]
[-98, -9, 234]
Enter number of lab : (2,3,4,5, 0-to exit)
```

Рис 2 Результат выполнения с данными 2

2.2 Пример 2

```
#include <iostream>
using namespace std;
long double fact(int N)
{
 if(N < 0)
return 0;
if (N == 0)
return 1;
else
return N * fact(N - 1);
}

int main()
 {
 int N; setlocale(0,"");
cout << "";
cin >> N;
cout << "" << N << " = " << fact(N) << endl << endl;
 return 0;
 }
```

```
Enter number of lab : (2,3,4,5, 0-to exit)
5
Enter : 5
120
```

Рис 1 Результат выполнения с данными 1

```
Enter number of lab : (2,3,4,5, 0-to exit)
5
Enter : 3
6
```

Рис 2 Результат выполнения с данными 2

Выводы

Я изучил основы методов трансляции и разработал собственный интерпретатор, работающий на 3 основных фазах: лексический, синтаксический и семантический анализаторы. Язык программирования Python оказался очень достойным языком для изучения данного раздела и C++ был отличным инструментом, для изучения.

Сложность заключалась в реализации конечного автомата, способного самостоятельно, используя анализаторы, реализовать конечную задумку и выдать результат обработки кода на C++.

Язык разбора Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода, в то же время стандартная библиотека включает большой объём полезных функций. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Код в Python организовывается в функции и классы, которые могут объединяться в модули (они в свою очередь могут быть объединены в пакеты).

Приложение А. - Исходный код анализируемой программы

```
void SortAlgo::mergeSort(int data[], int lenD)\n
{\n
    str mess = "HI" \n
  mess = mess + j \n
  iff (lenD>1){\n
    int middle = lenD/2;\n
    int rem = lenD-middle;\n
    int as = mess * j\n
    int *L = new int [middle];\n
    int *R = new int [rem];\n
    for (int i=0;i<lenD;i++){\n
      if (i<middle){\n
        L[i] = data[i];\n
      }\n
      elsee {\n
        R[i-middle] = data[i];\n
      }\n
    }\n
    strcat ( middle , rem )\n
    mergeSort@(L,middle);\n
    float res = mess * lenD
    mergeSort(R,rem);\n
    merge(data, lenD, L, middle, R, rem);\n
    mess ++ ; \n
  }\n
}\n

voi SortAlgo::merge(int merged[], int lenD, int L[], int lenL, int R[], int
lenR){\n
  inttt i = 0;\n
  int j = 0;\n
  while123 (i<lenL||j<lenR){\n
    iif (i<lenL & j<lenR){\n
      if (L[i]<=R[j]){\n
        merged[i+j] === L[i];\n
        i ++;\n
      }\n
      else {\n
        merged[[i+j] = R[j];\n
        j ++;\n
      }\n
    }\n
    else if (i<lenL){\n
      merged[i+j] =!= L[i];\n
      i++;\n
    }\n
    elses if (j<lenR){\n
      merged [[[ i+j ] = R[ j ];\n
      j ++;\n
    }\n
  }\n
}\n
"""
```

Приложение Б. - Исходный код интерпретатора

```
import os, sys
try:
    from setuptools import setup
    from setuptools.command.install import install as _install
    from setuptools.command.sdist import sdist as _sdist
except ImportError:
    from distutils.core import setup
    from distutils.command.install import install as _install
    from distutils.command.sdist import sdist as _sdist


def _run_build_tables(dir):
    from subprocess import check_call
    # This is run inside the install staging directory (that had no .pyc
files)
    # We don't want to generate any.
    # https://github.com/eliben/pycparser/pull/135
    check_call([sys.executable, '-B', '_build_tables.py'],
               cwd=os.path.join(dir, 'pycparser'))


class install(_install):
    def run(self):
        _install.run(self)
        self.execute(_run_build_tables, (self.install_lib,),
                     msg="Build the lexing/parsing tables")


class sdist(_sdist):
    def make_release_tree(self, basedir, files):
        _sdist.make_release_tree(self, basedir, files)
        self.execute(_run_build_tables, (basedir,),
                     msg="Build the lexing/parsing tables")


setup(
    # metadata
    name='pycparser',
    description='C parser in Python',
    long_description="""
        pycparser is a complete parser of the C language, written in
        pure Python using the PLY parsing library.
        It parses C code into an AST and can serve as a front-end for
        C compilers or analysis tools.
    """,
    license='BSD',
    version='2.21',
    author='Eli Bendersky',
    maintainer='Eli Bendersky',
    author_email='eliben@gmail.com',
    url='https://github.com/eliben/pycparser',
    platforms='Cross Platform',
    classifiers = [
        'Development Status :: 5 - Production/Stable',
        'License :: OSI Approved :: BSD License',
        'Programming Language :: Python :: 2',
        'Programming Language :: Python :: 2.7',
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.4',
        'Programming Language :: Python :: 3.5',
```

```
        'Programming Language :: Python :: 3.6',
        'Programming Language :: Python :: 3.7',
        'Programming Language :: Python :: 3.8',
        'Programming Language :: Python :: 3.9',
        'Programming Language :: Python :: 3.10',
    ],
    python_requires=">=2.7, !=3.0.*, !=3.1.*, !=3.2.*, !=3.3.*",
    packages=['pycparser', 'pycparser.ply'],
    package_data={'pycparser': ['*.cfg']},
    cmdclass={'install': install, 'sdist': sdist},
)

from data import data
import re
def second():
    pattern = ["void", "int", "while", "if", "else"]
    common_mistake = ["==","++","--"]
    i = 0
    j = 1

    n = len(data.split("\n"))
    for token in data.split("\n"):
        # print(token,"\n", token.split(),"\n")
        i+=1
        for el in token.split():

            if (len(el) >= 2 and el[0] == "i" and el[1] == "f") or (len(el)
>= 3 and el[0] == "i" and el[1] == "i" and el[0] == "f"):
                if len(el) != 2:
                    print(f"\n\n---------ERROR:UNEXPECTED TOKEN invalid lexic
if-statement {el} on line {i} of {n}  pos {1}")
                    print(token)

                    j += 1
                    break

            elif el[0] == "v" and el[1] == "o":
                if len(el) != 4 or el[2] != "i" or el[3] != "d":
                    print(f"\n\n---------ERROR:UNEXPECTED TOKEN invalid lexic
void-statement: {el} on line {i} of {n}  pos {1}")
                    print(token)
                    j += 1
                    break
            elif el[0] == "=":
                if len(el) > 2 or el[-1] != "=":
                    print(f"\n\n---------ERROR:UNEXPECTED TOKEN invalid lexic
===-statement: {el} on line {i} of {n}  pos {13}")
                    print("merged[i+j] === L[i];")

                    j += 1
            elif len(el)> 10 and el[9] == "@":
                if len(el) > 2 or el[-1] != "=":
                    print(
                        f"\n\n---------ERROR:UNEXPECTED TOKEN invalid lexic
non-existing statement: {el} on line {i} of {n}  pos {10}")
                    print(el)

                    j += 1

            elif len(el) > 2 and el[0] == "i" and el[1] == "n":
                if len(el) != 3 or el[-1] != "t":
                    print(f"\n\n---------ERROR:UNEXPECTED TOKEN invalid lexic
int-statement :{el} on line {i} of {n}  pos {0}")
                    j += 1
                    print(el)
```

```
                    print(token)

            elif el[0] == "[":
                if len(el) >= 2 and el[1] == "[":
                    print(f"\n\n---------ERROR:UNEXPECTED TOKEN invalid lexic
[]-statement :{el} on line {i} of {n}  pos {0}")
                    j += 1
                    print(token)
            elif el[0] == "]":
                if len(el) >= 2 and el[1] == "]":
                    print(f"\n\n---------ERROR:UNEXPECTED TOKEN invalid lexic
[]-statement :{el} on line {i} of {n}  pos {0}")
                    j += 1
                    print(token)
            elif el == '=!=':
                print(f"\n\n---------ERROR:UNEXPECTED TOKEN invalid lexic
non-existing statement :{el} on line {i} of {n}  pos {13}")
                j += 1
                print(token)
```