

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

КУРСОВАЯ РАБОТА

Разработка транслятора на C++ с языка Basic
по дисциплине «Технологии объектно-ориентированного программирования»

Выполнил

студент гр.3530903/00003

<подпись>

А.С. Усыченко

Руководитель

доцент, к.т.н.

<подпись>

К.А. Туральчук

«29» декабря 2021 г.

Санкт-Петербург

2021

Содержание

Введение.....	3
1. Теоретическая подготовка	
1.1. С чем будем иметь дело?.....	4
1.2. Описание программы.....	5
1.3. Главные проблемы.....	6
2. Разработка программы	
2.1. Реализация трансляции.....	7
2.2. Реализация взаимодействия с пользователем и запуска программы.....	9
2.3. Тестирование программы.....	10
3. Подведение итогов.....	10
4. Источники.....	11
5. Приложение.....	12

Введение

Транслятор с одного языка программирования на другой – крайне полезная программа. Довольно часто в написании программ, в сложных ситуациях, мы начинаем искать решение проблемы в интернете. В основном, решение действительно можно найти на подходящем языке программирования, но так происходит не всегда.

Приведу пример реальной ситуации – в интернете, ещё с 2006 года, есть код, который рисует летающий пончик в консоли C++ (Рис. 1), сам код не очень трудно понять, но вот математика, участвующая в построении – сложнее и интереснее. Однако, найти объяснение формул и законов – оказалось трудной задачей. После долгих поисков, достойная статья [1] в интернете была найдена, но все примеры в ней на языке программирования “Python”. Получается, что чтобы сделать что-то подобное – нужно выучить другой язык программирования, хоть и простой. Вот и необходимость в трансляторе.

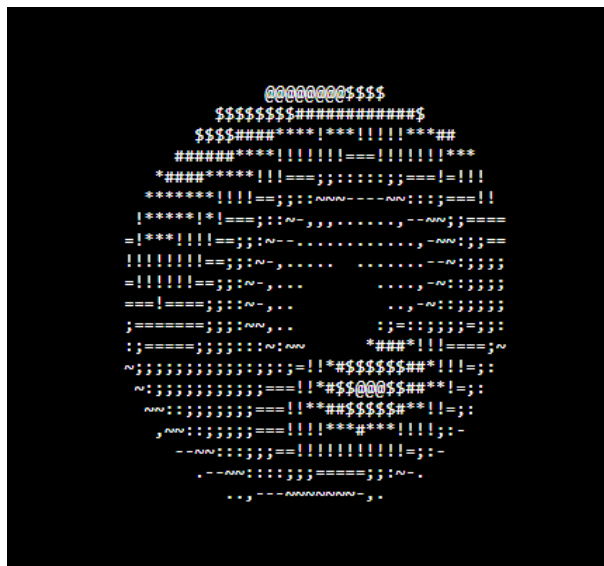


Рисунок 1

Выше описана лишь одна ситуация, когда может понадобиться транслятор, однако, такие ситуации встречаются часто.

Соответственно, создание транслятора с Basic на C++ - идея хорошая, полезная, хоть и не новая. К тому же, сразу становится ясно, что во время написания придется решать много проблем, что в итоге закрепляет знания в программировании.

Целью моей курсовой работы будет написание понятного для пользователя и верно работающего транслятора с языка программирования Visual Basic на C++

Программа должна уметь транслировать:

- 1) Взаимодействие с консолью (Ввод вывод)
- 2) Операторы с условием
- 3) Циклы
- 4) Операции с переменными
- 5) Комментарии

В случае выполнения всех вышеперечисленных задач получится примитивный транслятор.

Начнем!

1. Теоретическая подготовка

1.1 С чем будем иметь дело?

Программа выполнена в IDE “Visual Studio 2019”. В ходе написания программы придется использовать работу с файлами, несколько классов, организовать модули взаимодействия с пользователем и нужно придумать алгоритм транслирования.

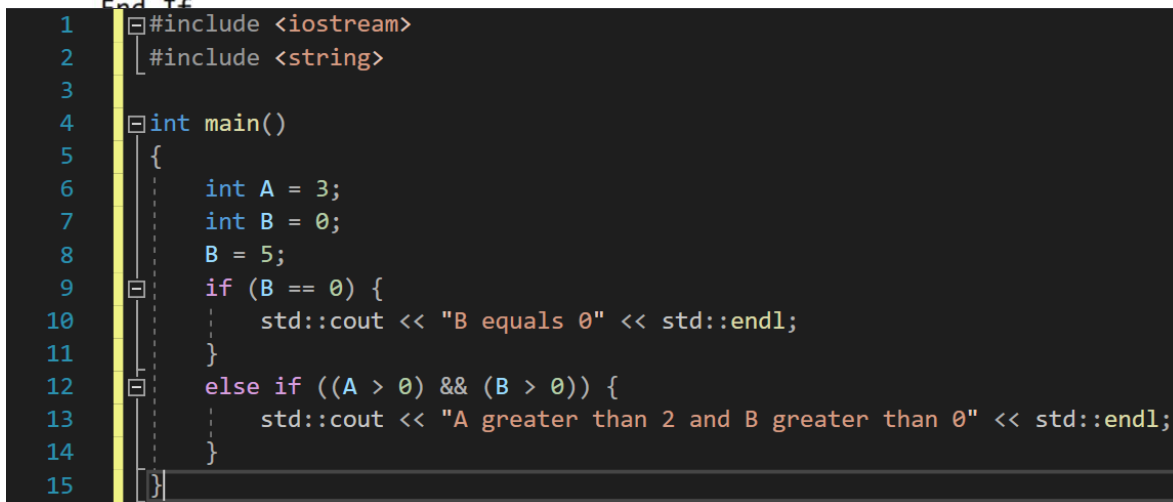
Разберемся с последним, остальное – в практической части.

После рассмотрения нескольких идей для скелета – я выбрал наиболее практичную и безотказную модель – переводить текст по словам с одного языка на другой.

Программа будет получать файл или же команды будут вводиться с консоли, далее все строчки будут преобразовываться в вектор, элементами которого как раз будут эти строчки. Далее отдельно каждая строчка будет разделяться по пробелам на слова, и они будут записываться в другой вектор и далее в цикле будет происходить работа с каждым словом, и оно будет переводиться с одного языка на другой.

Рассмотрим пример, с которым должна справляться программа и такой же код переведенный руками:

```
Dim A as Integer = 3
Dim B as Integer = 0
B = 5
If B = 0 Then
    Console.WriteLine("B equals 0")
ElseIf A > 2 AND B > 0 Then
    Console.WriteLine("A greater than 2 and B greater than 0")
End If
```



```
1 #include <iostream>
2 #include <string>
3
4 int main()
5 {
6     int A = 3;
7     int B = 0;
8     B = 5;
9     if (B == 0) {
10         std::cout << "B equals 0" << std::endl;
11     }
12     else if ((A > 0) && (B > 0)) {
13         std::cout << "A greater than 2 and B greater than 0" << std::endl;
14     }
15 }
```

Как видно – серьёзные различия в коде есть лишь в некоторых местах, а именно в объявлении переменных и в условных операторах. Следовательно, теория о последовательном переводе, слово за словом, имеет право на жизнь. Тем более, заранее известно после каких слов начинаются серьёзные различия, значит это можно предусмотреть.

Разберемся с тем, какой конкретно функционал будет присутствовать в программе и какие, основные, ограничения она будет иметь.

Функционал:

- 1) Реализация вывода на консоль, как уже готовых текстовых сообщений, так и возможность производить математические операции в поле вывода
- 2) Реализация ввода данных, а именно возможность пользователю ввести значение переменной
- 3) Поддержка циклов for и while
- 4) Поддержка условного оператора if и конструкции else if
- 5) Возможность определения переменных разных типов
- 6) Поддержка комментариев

Ограничения:

- 1) Использование только поддерживаемых команд
- 2) Не все типы поддерживаются

Так же, в разных версиях Visual Basic синтаксис отличается, за основу будут браться примеры с официального сайта Microsoft [2].

1.2 Описание программы

Я решил разделить весь функционал программы на 3 класса, первый будет отвечать непосредственно за трансляцию, второй – получать начальные данные и запускать итоговый результат, третий – визуализация, вывод данных на консоль.

Необходимости в разработке визуализации – нет, так как в данной программе главное – это практическая часть, да и всё что дает визуализация – это возможность изменения кода в реальном времени и более приятные кнопки. Казалось бы, возможность изменения кода – очень важная вещь, но в моей реализации она является неосуществимой. Подробнее об этом в следующем пункте.

Алгоритм:

Получение начального кода, из файла или с консоли → перевод от слова к слову, при наличии “Плохих мест”, обработка разом всей строки → запись в файл .cpp, находящийся в той же директории, что и программа → запуск файла.

1.3 Главные проблемы

Давайте углубимся в реализацию программы, а именно в то, как в ней запускается переведённый файл. В C++ есть несколько способов запуска других файлов, наиболее подходящими для меня были – `system(path)` и подключение файла к проекту.

Изначально, я пытался реализовать идею с `system`, однако, всё что мне удавалось – это открыть созданный .cpp файл в проекте. Далее мной была придумана простая, но работающая идея. Сохранять переведённый код всегда в один и тот же файл, поместить этот файл в директорию программы и подключить его как библиотеку к программе. Тогда весь переведенный код можно обернуть в функцию и вызвать её когда будет необходимо. “На бумаге” идея звучит хорошо, да и на практике всё работает, за исключением одной проблемы – со второго раза. Да, чтобы запустить переведенный код, нужно сначала запустить программу и перевести всё, что хочется, а потом запустить программу ещё раз и запустить переведенный код. Эту проблему мне так и не удалось обойти, так как файл подключается к программе на стадии компиляции, а на этой стадии файл ещё не содержит в себе актуального перевода. Я пытался обойти это попыткой подключением файла не в начале файла или запуском программы самой из себя, но это не помогло. В итоге пришлось просто прописать это в интерфейсе.

Другая проблема связана так же с подключенным файлом – он не должен содержать ошибок, таких как использование неинициализированной переменной, неверный синтаксис и всё подобное, что способно вызвать ошибку на стадии компиляции. Данные ошибки невозможно исправить без вмешательства. Чтобы их исправить, нужно запустить программу, а программа не запускается из-за ошибок компиляции, замкнутый круг. Решение этой проблемы довольно простое, но не рациональное – отключить файл руками, скомпилировать программу, закрыть программу, подключить файл обратно.

2. Разработка программы

2.1 Реализация трансляции

Как уже говорилось ранее, программа работает с каждым словом отдельно, но в некоторых случаях идет обработка целой строки. С таким раскладом так и просится switch case, однако, в C++ нет поддержки string в switch. Чтобы обойти это ограничение, был создан словарь map <string, int> со словами в роле ключа и их порядковым номером для использования в switch.

Теперь рассмотрим отдельные случаи.

For

Всего я выделил 3 классификации для цикла for, “For indexA = 1 To 3”, “For indexB = 20 To 1 Step -3”, “For number As Double = 2 To 0 Step -0.25”, основные различия в них – в первых двух вариациях нет определения типа переменной, поэтому я по умолчанию решил считать её int, так же они отличаются тем, что в каких-то случаях используется “Step”, а в каких-то нет, поэтому я проверяю есть ли на 7 или 9 месте слово “Step” и, в зависимости от результата, прибавляю число идущее после или же прибавляю 1. Стоит отметить, что в случае наличия – нам не важен знак числа, так как $+(-x) = -x$. После пред усмотрения всех вышеперечисленных пунктов и понимая, что ключевые элементы всегда стоят на одних и тех же местах, было принято решение обрабатываться всю строчку целиком. Итоговый код для трансляции for (Рис. 2).

```
case 12: {
    string nameOfVar;
    bool check = false;
    ostrm << "for(";
    k += 2;
    if (line[k] == "=") {
        k--;
        nameOfVar = line[k];
        ostrm << "int ";
        k += 2;
    }
    else {
        k++;
        ostrm << Translator::GetType(line[k]);
        k -= 2;
        nameOfVar = line[k];
        k += 4;
    }
    int firstbord = stoi(line[k]);
    k += 2;
    int secondbord = stoi(line[k]);
    ostrm << nameOfVar << " = ";
    if (firstbord < secondbord) {
        ostrm << firstbord << "; " << nameOfVar << " <= " << secondbord << "; " << nameOfVar;
        check = true;
    }
    else {
        ostrm << firstbord << "; " << nameOfVar << " >= " << secondbord << "; " << nameOfVar;
    }
    k++;
    if ((line.size() > 8) || ((line.size() > 6) && (line[k] == "Step"))) {
        k++;
        ostrm << " = " << nameOfVar << " + (" << line[k] << ") {\n";
    }
    else {
        if (check) ostrm << "++;" {\n";
        else ostrm << "--;" {\n";
    }
    k = line.size();
    break;
}
```

Рисунок 2

Cout && cin

В Visual Basic имеется так же 3 вариации взаимодействия с консолью – это “Console.Write()”, “Console.WriteLine()” и “Console.ReadLine()”. Как видим, различаются они частью слова, идущей после “.” и до “(”, поэтому было решено проверять все строчки и пытаться вычленить кусок от точки и до скобки. Поиск символа в строке осуществлялся с помощью метода find(“*”) [3], который возвращает индекс первого встреченного элемента в строке. К тому же, я заключил этот процесс в try{ } catch { }, так как тут мы работаем с индексами и можем случайно получить ошибку обращения к несуществующему индексу.

В случае благоприятного исхода и выделения куска от точки до скобки, этот кусок записывается во временную переменную и проверяется через if else, в зависимости от результата условного оператора – первое слово в строке заменялось на более простое и различное для всех 3 случаев, уже с этой заменой строка отправлялась в switch. Код поиска куска – Рис. 3.

```
for (int k = 0; k < line.size(); k++) {  
    try {  
        std::size_t found = line[k].find(".");  
        std::size_t found2 = line[k].find("(");  
        string console = line[k].substr(found + 1, found2 - found - 1);  
        if (console == "Write") {  
            copyline = translate.StringsInFile[i];  
            line[k] = "Wri";  
        }  
        else if (console == "WriteLine") {  
            copyline = translate.StringsInFile[i];  
            line[k] = "Con";  
        }  
    }  
}
```

Рисунок 3

If && if else

Последний исключительный случай связан с условным оператором, а точнее с записью условий, в Visual Basic не используются скобки в данном операторе, а значит нужно добавлять их самостоятельно. Было решено, заключать все выражения внутри if в скобки. С лишними скобками программа работает, а вот без них – нет. По поводу if else,

на самом деле там нет никаких отличий от if, поэтому программа просто выводит дополнительное слово else, а дальше переходит в case для if. Код реализации if – Рис. 4

Используется case тот же, что и для всей программы, только в него добавлены соответствующие ключи.

```
case 3: {
    ostrm << " if( ";
    for (int l = k+1; l < line.size() - 1; l++) {
        switch (dictionary[line[l]])
        {
            case 9:
                ostrm << " == ";
                break;
            case 4: ostrm << ") && (";
                break;
            case 5: ostrm << ") || (";
                break;
            default:
                ostrm << line[l];
                break;
        }
    }
    k = line.size() - 2;
```

Рисунок 4

2.2 Реализация взаимодействия с пользователем и запуска программы

Взаимодействие с пользователем было реализовано непосредственно в функции main, вывод начального кода и полученного реализован через перегрузку оператора вывода и через построчное считывание из полученного файла и вывод строчек на консоль. Меню организовано так же через switch и ввод пользователем того, что он хочет.

Для вывода создан дополнительный класс, чтобы разгрузить основной класс-переводчик. Он является наследником основного. В таком формате делать визуализацию проще, так как нет возможности сломать работающий алгоритм.

Для запуска программы написан еще один класс-стартер, который не имеет связи с классом-переводчиком. Вся задача стартера – в зависимости от желания пользователя получить код для перевода из разных источников, а затем запустить файл, в который записывался переведенный код. Файл, в который выполняется перевод, подключается как библиотека именно к этому классу и запустить переведённый код так же можно только из этого класса.

Так как никакой визуализации, кроме консоли, нет, то в визуализации не используется сложных конструкций, всё делается через обычную консольную графику.

2.3 Тестирование программы

Во время написания каждого блока трансляции проводилось неоднократное тестирование с разными входными данными. После комплексного тестирования готовой программы были выявлены проблемы, описанные в пункте 1.3 данной курсовой работы. К сожалению, решить их так и не удалось. Все вариации пользовательского ввода так же были протестированы, поэтому единственные ошибки, которые могут привести к неверной работе программы – это изначально неверный код для перевода. Так же реализована “Защита от дураков”, которая контролирует то, что пользователь вводит в консоль именно то, что необходимо.

3. Подведение итогов

В заключении своей работы могу сказать, что мне удалось выполнить все задачи и цели, которые были поставлены изначально. Я закрепил знания по работе с файлами и придумал способ трансляции, который считаю самым оптимальным.

Если доработать программу, добавить больше элементов для перевода, прикрутить визуализацию не через консоль, то может получиться очень полезное приложение, которое поможет начинающим программистам. Так как в большом количестве Российских школ изучают Pascal/Basic и с помощью такого транслятора детям будет куда проще перейти на C++ с Basic.

4. Источники

1. <https://ichi.pro/ru/donut> (24.12.21)
2. <https://docs.microsoft.com/ru-ru/dotnet/visual-basic/> (25.12.21)
3. <https://wwwcplusplus.com/reference/algorithm/find/> (25.12.21)

Бумажные источники, к которым я обращался во время всей работы

1. “Язык программирования C++; Лекции и упражнения” 6 издание, Стивен Прата.

5. Приложение

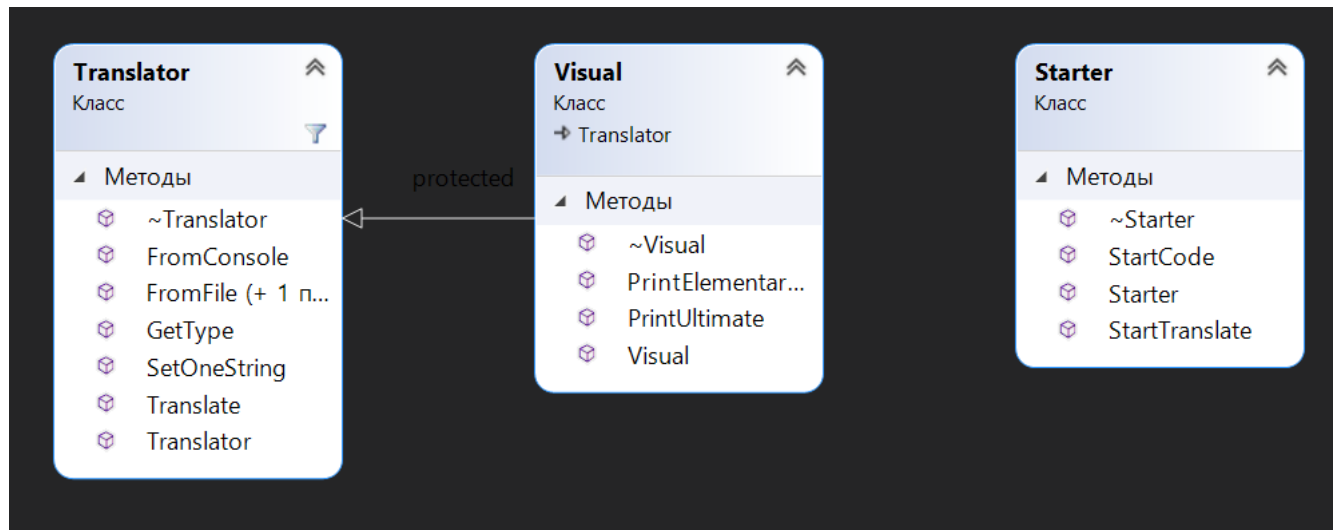


Диаграмма классов