

Санкт-Петербургский политехнический университет Петра Великого
Кафедра компьютерных систем и программных технологий

Отчёт по курсовому проекту на тему “Интерпретатор LOOP”

Дисциплина: Прикладное программирование

Выполнила студентка гр. 23531/3

(подпись) М.А. Сабо

Преподаватель

(подпись) Т.Л. Сидорина

“ _ ” _____ 2018 г.

Санкт-Петербург

2018

Оглавление

| | |
|---------------------------------------|----|
| 1. Техническое задание | 3 |
| 2. Метод решения | 4 |
| 3. Листинг программы..... | 6 |
| 4. Makefile..... | 7 |
| 5. Коды ошибок | 8 |
| 6. Программа и методы испытаний | 9 |
| 7. Приложения | 16 |

1. Техническое задание

Входные параметры программы:

- имя файла с текстом программы (формат .txt) (Таблица 1)
- имя файла (формат .txt), где переменные перечислены в виде: x1=3, x2=5, x10=6 и тд (Таблица 2)
- имя выходного файла с результатом работы программы (формат .txt) (Таблица 3)

Командная строка : iloop.exe program.txt parameters.txt output.txt COMLIMIT(not necessary)

Default: COMLIMIT = 100

Выходные данные программы:

Программа формирует текстовый файл с результатом синтаксического разбора и результатами работы программы.

В файле содержится следующая информация:

- переданная команда
- результат синтаксического разбора (Таблица 3)
- значения переменных после выполнения команды
- в последней строке значение x0

Таблица 1.

```
program.txt
x1 := x2 + 3;
x1 := 6;
x0 := x1 + 1;
LOOP x1
DO
x6 := x1 + 2;
x5 := x6 + 4;
END;
x6 = x5 + 1;
```

Таблица 2.

```
parameters.txt
x2 = 1, x3 = 8
```

Программа выводит результат синтаксического разбора на экран и в файл. Если результат отрицательный, то программа не начинает работать.

Ссылка на репозиторий в Github :

<https://github.com/maria-sabo/InterpreterLoop>

Таблица 3.

```
output.txt
x1:=x2 + 3;
x1:=6;
x0:=x1+1;
LOOP x1
DO
x6:=x1+2;
x5:=x6+4;
END;
x6=x5+1;
Нет синтаксических ошибок.
x1 := x2 + 3;      x1 = 4 x2 = 1 x3 = 8
x1 := 6;           x1 = 6 x2 = 1 x3 = 8
x0 := x1 + 1;      x0 = 7 x1 = 6 x2 = 1 x3 = 8
DO                x0 = 7 x1 = 6 x2 = 1 x3 = 8
x6 := x1 + 2;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x6 = 8
x5 := x6 + 4;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
x6 := x1 + 2;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
x5 := x6 + 4;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
x6 := x1 + 2;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
x5 := x6 + 4;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
x6 := x1 + 2;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
x5 := x6 + 4;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
x6 := x1 + 2;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
x5 := x6 + 4;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
x6 := x1 + 2;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
x5 := x6 + 4;      x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
END;              x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 8
x6 = x5 + 1;       x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 13
x0 = 7             x0 = 7 x1 = 6 x2 = 1 x3 = 8 x5 = 12 x6 = 13
```

2. Метод решения

Программа запускается командной строкой:

`iloop.exe <имя файла программы> <имя файла со значениями переменных> <имя выходного файла> <количество выполняемых команд(необязательно)>`

Если параметров передано меньше, программа выводит справку.

Если с одним параметром `help` – выводится более подробная справка.

При работе программы происходит загрузка текста программы из файла программы, загрузка текста из файла с переменными, синтаксический разбор этих файлов и запуск программы на выполнение. Результат выполнения программы записывается в выходной файл. Если выходной файл не открыть, то программа показывает результат выполнения только на экран.

Программа состоит из следующих модулей:

-iloop.c – главный модуль программы.

В модуле производится разбор параметров, открытие входных и выходного файлов, запуск загрузки текстов программы и параметров в списки, вызов процедуры синтаксического разбора, запуск программы на исполнение.

-iload.c – программа загрузки текстов файлов.

В модуле из файлов с текстом программы и инициализации переменных считывается информация с разбором на команды и помещается в строки списков.

-isyntax.c – программа синтаксического разбора текста.

В модуле из списков со строками команд создаются списки типа `Command`:

```
typedef struct _Command {
    int ns; // номер строки
    char * buffer; // указатель на строку с командой
    int command; // номер команды
    int error; // номер ошибки
    size_t p1; // индекс переменной слева или переменной в LOOP
    size_t p2; // индекс переменной справа или константа
    size_t p3; // константа
} Command;
```

В этой структуре содержится результат синтаксического разбора – номер синтаксической ошибки или параметры команды.

Номера команд:

- 1 – DO
- 2 – END
- 3 – LOOPx
- 4 – :=+ (присвоение с плюсом)
- 5 – :=- (присвоение с минусом)
- 6 – = (инициализация переменной)

-command.c – программа, выполняющая загруженную программу.

В модуле проходим по списку типа `Command` и последовательно выполняем команды.

Если встретился `LOOP`, помещаем в список `list_loop` элемент <номер строки

начала> <количество повторений>.

Если встретился END, уменьшаем в последнем элементе списка list_loop

<количество повторений> и перемещаем указатель прохода на <элемент строки начала>, если <количество повторений> стало меньше либо равно нулю, убираем последний элемент из списка.

-**command.h** – заголовочный файл для описания структур Command, LoopEl, VarEl

-**ilist.c** – модуль функций работы со списком

-**ilist.h** – заголовочный файл для описания структур Node, DbLinkedList

3. Листинг программы

iloop.c (Приложение 1)

iloop.h (Приложение 2)

iload.c (Приложение 3)

isyntax.c (Приложение 4)

command.c (Приложение 5)

command.h (Приложение 6)

ilist.c (Приложение 7)

ilist.h (Приложение 8)

4. Makefile

iloop.exe : iloop.o isyntax.o command.o ilist.o iload.o

gcc -o iloop iloop.o isyntax.o command.o ilist.o iload.o

iloop.o: iloop.c

gcc -std=c11 -pedantic -Wall -Wextra -c -o iloop.o iloop.c

isyntax.o: isyntax.c

gcc -std=c11 -pedantic -Wall -Wextra -c -o isyntax.o isyntax.c

command.o: command.c

gcc -std=c11 -pedantic -Wall -Wextra -c -o command.o command.c

ilist.o: ilist.c

gcc -std=c11 -pedantic -Wall -Wextra -c -o ilist.o ilist.c

iload.o: iload.c

gcc -std=c11 -pedantic -Wall -Wextra -c -o iload.o iload.c

5. Коды ошибок

| Код ошибки | Описание ошибки | Действие |
|------------|---|----------------------|
| 1 | Garbage characters in the use of DO | Завершение программы |
| 2 | Garbage characters in the use of END | Завершение программы |
| 3 | Garbage characters in the use of LOOP-X | Завершение программы |
| 4 | Garbage characters in the index of LOOP x | Завершение программы |
| 5 | Garbage characters in the left part of +- | Завершение программы |
| 6 | Garbage characters in the left part of := | Завершение программы |
| 7 | Garbage characters in the index of left part | Завершение программы |
| 8 | Plus or minus are not found | Завершение программы |
| 9 | There is unrecognised command | Завершение программы |
| 10 | Garbage characters in the use of x | Завершение программы |
| 11 | Garbage characters in the index of x | Завершение программы |
| 12 | Garbage characters in the right part of = | Завершение программы |
| 13 | There is unrecognised command | Завершение программы |
| 14 | Count of DO greater than LOOP | Завершение программы |
| 15 | Count of END greater than LOOP or DO | Завершение программы |
| 21 | The x is not available | Завершение программы |
| 22 | The x is not initialised | Завершение программы |
| 23 | Garbage characters in the right part of +- \0 | Завершение программы |
| 24 | Limit of index is exceeded | Завершение программы |

Ошибки загрузки программы:

| Описание ошибки | Действие |
|---------------------|----------------------|
| Command is too long | Завершение программы |
| String is too long | Завершение программы |

Ошибки при передаче параметров:

| Описание ошибки | Действие |
|---|-------------------|
| Input file name and output file name couldn't equal | Завершение работы |
| Wrong COMLIMIT paramrter | Завершение работы |
| Output file can not be create | Завершение работы |

6. Программа и методы испытаний

6.1 Запуск программы без аргументов или с параметром help.

```
C:\temp>iloop
Author: Sabo Maria
This program interpret programs written in the LOOP language.
For access the commands used, run the program with the parameter help.
For running the program you need to write paths to files.
Unnecessary COMLIMIT: positive integer. Default COMLIMIT = 100.
Example: iloop.exe program.txt var.txt out.txt COMLIMIT(not necessary)

C:\temp>iloop help
For access the commands used, run the program with the parameter help.
For running the program you need to write paths to files.
Unnecessary COMLIMIT  positive integer. Default COMLIMIT = 100.
Example: iloop.exe program.txt var.txt out.txt COMLIMIT(not necessary)

You can use commands such as this:
  xN := xM +- C;
  LOOP xi
  DO
  something...
  END;
You can input parametres such as this: x0 = C1, ... xN = CN
You can use only non-negative integer
Для продолжения нажмите любую клавишу . . .
```

6.2 Запуск программы с несуществующим входным файлом программы.

```
C:\temp>iloop.exe notexist.txt var.txt out.txt
File notexist.txt not found
Для продолжения нажмите любую клавишу . . .
Load program error
Для продолжения нажмите любую клавишу . . .

C:\temp>_
```

6.3 Запуск программы с несуществующим входным файлом переменных.

```
x1:=x0+5
x2:=x1+6
x3:=x1+1
LOOPx3
DO
LOOPx2
DO
LOOPx1
DO
x0:=x2+1
END
END
END

Для продолжения нажмите любую клавишу . . .
File notexist.txt not found
Для продолжения нажмите любую клавишу . . .
Load program error
Для продолжения нажмите любую клавишу . . .

C:\temp>iloop.exe in.txt notexist.txt out.txt
```


6.6 Ошибки выполнения программы

Используемая переменная не инициализирована.

```
1  x1=3
2  x5=1
3  x3=8
4  x0=2147483647
5  x123=2
1  x1:=x3+19
2  LOOPx2  The x is not available
The program is interrupted
x0 = 2147483647
Для продолжения нажмите любую клавишу . . .
```

Execution trace for the first error:

```
x1=3
x1=3  x5=1
x1=3  x5=1  x3=8
x1=3  x5=1  x3=8  x0=2147483647
x1=3  x5=1  x3=8  x0=2147483647  x123=2
x1=27  x5=1  x3=8  x0=2147483647  x123=2
x1=27  x5=1  x3=8  x0=2147483647  x123=2
```

```
1  x1=3
2  x5=1
3  x3=8
4  x0=2147483647
5  x123=2
1  x1:=x3-8
2  x4:=x2+15  The x is not initialised
The program is interrupted
x0 = 2147483647
Для продолжения нажмите любую клавишу . . .
```

Execution trace for the second error:

```
x1=3
x1=3  x5=1
x1=3  x5=1  x3=8
x1=3  x5=1  x3=8  x0=2147483647
x1=3  x5=1  x3=8  x0=2147483647  x123=2
x1=0  x5=1  x3=8  x0=2147483647  x123=2
x1=0  x5=1  x3=8  x0=2147483647  x123=2
```

6.7 Пример успешного выполнения программы

Входные файлы:

```
program5.txt — Блокнот
Файл  Правка  Формат  Вид  Справка
x1 := x0 + 0;
x2 := x1 + 1;
x3 := x1 + 1;
LOOP x3
DO
    x3 := x0 - 6;
    LOOP x2
    DO
        x0 := x1 + 1000;
    END;
END;
```

```
var5.txt — Блокнот
Файл  Правка  Формат  Вид  Справка
x1 = 1, x2 = 3, x0 = 0, x3 = 4
```

```

x1:=x0+0
x2:=x1+1
x3:=x1+1
LOOPx3
DO
x3:=x0-6
LOOPx2
DO
x0:=x1+1000
END
END

```

Для продолжения нажмите

```

x1=1
x2=3
x0=0
x3=4

```

Для продолжения нажмите

Результаты синтаксического разбора:

```

1  x1:=x0+0
2  x2:=x1+1
3  x3:=x1+1
4  LOOPx3
5  DO
6  x3:=x0-6
7  LOOPx2
8  DO
9  x0:=x1+1000
10 END
11 END

```

Programm: 0 syntax error found
Для продолжения нажмите любую клавишу . . .

```

1  x1=1
2  x2=3
3  x0=0
4  x3=4

```

Variable: 0 syntax error found
Для продолжения нажмите любую клавишу . . .

Результат выполнения программы:

| | | | | | | | | | |
|----|-------------|------|------|---------|-------|-----|-----|--|--|
| 1 | x1=1 | | | | | | | | |
| 2 | x2=3 | | | | | | | | |
| 3 | x0=0 | | | | | | | | |
| 4 | x3=4 | | | | | | | | |
| 1 | x1:=x0+0 | x1=1 | x2=3 | x0=0 | x3=4 | | | | |
| 2 | x2:=x1+1 | x1=0 | x2=3 | x0=0 | x3=4 | | | | |
| 3 | x3:=x1+1 | x1=0 | x2=1 | x0=0 | x3=4 | | | | |
| 4 | LOOPx3 | x1=0 | x2=1 | x0=0 | x3=1 | | | | |
| 5 | DO | x1=0 | x2=1 | x0=0 | x3=1 | 5,1 | | | |
| 6 | x3:=x0-6 | x1=0 | x2=1 | x0=0 | x3=-6 | 5,1 | | | |
| 7 | LOOPx2 | x1=0 | x2=1 | x0=0 | x3=-6 | 5,1 | 8,1 | | |
| 8 | DO | x1=0 | x2=1 | x0=0 | x3=-6 | 5,1 | 8,1 | | |
| 9 | x0:=x1+1000 | x1=0 | x2=1 | x0=1000 | x3=-6 | 5,1 | 8,1 | | |
| 10 | END | x1=0 | x2=1 | x0=1000 | x3=-6 | 5,1 | | | |
| 11 | END | x1=0 | x2=1 | x0=1000 | x3=-6 | | | | |

x0 = 1000
Для продолжения нажмите любую клавишу . . .

Полученный выходной файл:

```
out5.txt — Блокнот
Файл  Правка  Формат  Вид  Справка
1  x1:=x0+0
2  x2:=x1+1
3  x3:=x1+1
4  LOOPx3
5  DO
6  x3:=x0-6
7  LOOPx2
8  DO
9  x0:=x1+1000
10 END
11 END

Programm: 0 syntax error found
1  x1=1
2  x2=3
3  x0=0
4  x3=4

Variable: 0 syntax error found
1  x1=1
2  x2=3
3  x0=0
4  x3=4
1  x1:=x0+0
2  x2:=x1+1
3  x3:=x1+1
4  LOOPx3
5  DO
6  x3:=x0-6
7  LOOPx2
8  DO
9  x0:=x1+1000
10 END
11 END
x0 = 1000

x1=1
x1=1  x2=3
x1=1  x2=3  x0=0
x1=1  x2=3  x0=0  x3=4
x1=0  x2=3  x0=0  x3=4
x1=0  x2=1  x0=0  x3=4
x1=0  x2=1  x0=0  x3=1
x1=0  x2=1  x0=0  x3=1  5,1
x1=0  x2=1  x0=0  x3=1  5,1
x1=0  x2=1  x0=0  x3=-6  5,1
x1=0  x2=1  x0=0  x3=-6  5,1  8,1
x1=0  x2=1  x0=0  x3=-6  5,1  8,1
x1=0  x2=1  x0=1000  x3=-6  5,1  8,1
x1=0  x2=1  x0=1000  x3=-6  5,1
x1=0  x2=1  x0=1000  x3=-6
```

6.8 Пример работы программы, в которой происходит очень выполнение большого количества команд.

Результат синтаксического разбора.

```
1  x1:=x0+5
2  x2:=x1+6
3  x3:=x1+1
4  LOOPx3
5  DO
6  LOOPx2
7  DO
8  LOOPx1
9  DO
10 x0:=x2+1
11 END
12 END
13 END

Programm: 0 syntax error found
Для продолжения нажмите любую клавишу . . .
```

```
1  x1=6
2  x2=6
3  x0=4
4  x3=5

Variable: 0 syntax error found
Для продолжения нажмите любую клавишу
```

В результате программа будет запрашивать пользователя о дальнейших действиях.

```
10 x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,10  7,12  9,1
11 END           x1=9  x2=15  x0=16  x3=10  5,10  7,12  9,1
7 DO            x1=9  x2=15  x0=16  x3=10  5,10  7,12
8 LOOPx1        x1=9  x2=15  x0=16  x3=10  5,10  7,11
9 DO           x1=9  x2=15  x0=16  x3=10  5,10  7,11  9,9
10 x0:=x2+1     x1=9  x2=15  x0=16  x3=10  5,10  7,11  9,9
9 DO           x1=9  x2=15  x0=16  x3=10  5,10  7,11  9,9
10 x0:=x2+1     x1=9  x2=15  x0=16  x3=10  5,10  7,11  9,8
9 DO           x1=9  x2=15  x0=16  x3=10  5,10  7,11  9,8
10 x0:=x2+1     x1=9  x2=15  x0=16  x3=10  5,10  7,11  9,7
9 DO           x1=9  x2=15  x0=16  x3=10  5,10  7,11  9,7
10 x0:=x2+1     x1=9  x2=15  x0=16  x3=10  5,10  7,11  9,6
9 DO           x1=9  x2=15  x0=16  x3=10  5,10  7,11  9,6
10 x0:=x2+1     x1=9  x2=15  x0=16  x3=10  5,10  7,11  9,6

100 commands completed.
0 - continue
1 - exit
```

```
10 x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,5
9 DO            x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,5
10 x0:=x2+1     x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,4
9 DO           x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,4
10 x0:=x2+1     x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,3
9 DO           x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,3
10 x0:=x2+1     x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,2
9 DO           x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,2
10 x0:=x2+1     x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,2
9 DO           x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,1
10 x0:=x2+1     x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,1
11 END         x1=9  x2=15  x0=16  x3=10  5,1  7,1
12 END         x1=9  x2=15  x0=16  x3=10  5,1
13 END         x1=9  x2=15  x0=16  x3=10

x0 = 16
Для продолжения нажмите любую клавишу . . .
C:\temp>
C:\temp>
```

Также можно запустить программу с необязательным параметром.

```
C:\temp>iloop.exe program6.txt var6.txt out6.txt 1000
```

```

10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,7
9   DO           x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,7
10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,6
9   DO           x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,6
10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,5
9   DO           x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,5
10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,4
9   DO           x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,4
10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,3
9   DO           x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,3
10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,2
9   DO           x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,2
10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,1
9   DO           x1=9  x2=15  x0=16  x3=10  5,7  7,14  9,1

1000 commands completed.
0 - continue
1 - exit

```

Таким образом, ~ 4000 командах программа закончит свое выполнение.

```

10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,6
9   DO           x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,5
10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,5
9   DO           x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,4
10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,4
9   DO           x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,3
10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,3
9   DO           x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,2
10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,2
9   DO           x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,1
10  x0:=x2+1      x1=9  x2=15  x0=16  x3=10  5,1  7,1  9,1
11  END           x1=9  x2=15  x0=16  x3=10  5,1  7,1
12  END           x1=9  x2=15  x0=16  x3=10  5,1
13  END           x1=9  x2=15  x0=16  x3=10

x0 = 16
Для продолжения нажмите любую клавишу . . .

```

7. Приложения

Приложение 1.

Файл loop.c

```
#include "stdafx.h"
#include "ilist.h"
#include "command.h"
#include "iloop.h"
#define COMLIMIT 100 // максимальная длина команды

int main(int argc, char* argv[])
{
    DbllinkedList * list_prog; // загруженные строки программы
    DbllinkedList * list_var; // загруженные строки с переменными
    DbllinkedList * list_command; // список команд разобранный программы
    DbllinkedList * list_init; // список команд разобранных переменных

    FILE * file_out;
    FILE * file_prog;
    FILE * file_var;

    list_prog = createDbllinkedList();
    list_var = createDbllinkedList();

    list_command = createDbllinkedList();
    list_init = createDbllinkedList();
    // вывод справки
    if (argc == 2 && strcmp(argv[1], "help") == 0) {
        printf(
            "\tFor access the commands used, run the program with the key. 'help'
\n"
            "\tFor running the program you need to write paths to files. \n"
            "\tUnnecessary COMLIMIT - positive integer. Default COMLIMIT = 100.
\n"
            "\tExample: iloop.exe program.txt var.txt out.txt [COMLIMIT] (not
necessary)\n\n");
        printf("You can use commands such as this: \n\txN := xM +- C;\n\tLOOP
xi\n\tDO\n\t\tsomething...\n\tEND;\n"
            "You can input parametres such as this: x0 = C1, ... xN = CN \n"
            "You can use only non-negative integer \n");
        //system("pause");
        printf("Enter a key to continue...\n");
        getchar();
        return 0;
    }
    if (argc < 4){
        printf("Author: Sabo Maria \n"
            "\tThis program interpret programs written in the LOOP language.\n"
            "\tFor access the commands used, run the program with the key. 'help'
\n"
            "\tFor running the program you need to write paths to files. \n"
            "\tUnnecessary COMLIMIT - positive integer. Default COMLIMIT = 100.
\n"
            "\tExample: iloop.exe program.txt var.txt out.txt [COMLIMIT](not
necessary) \n");
        return 0;
    }
    if (strcmp(argv[1], argv[3]) == 0 || strcmp(argv[2], argv[3]) == 0) {
        printf("Input file name and output file name couldn't equal");
        return 0;
    }
    // необязательный пятый аргумент
```



```

int comlimit;
if (argc == 5) {
    comlimit = atoi(argv[4]);
    if (comlimit <= 0) {
        printf("Wrong COMLIMIT parameter \n");
        return 0;
    }
}
else
    comlimit = COMLIMIT;

file_out = fopen(argv[3], "w");
if (file_out == NULL) {
    printf("Output file %s can not be create \n", argv[3]);
    printf("Enter a key to continue...\n");
    getchar();
}
file_prog = fopen(argv[1], "r");
if (file_prog == NULL) {
    printf("File %s is not found \n", argv[1]);
    printf("Enter a key to continue...\n");
    getchar();
}
else {
    file_var = fopen(argv[2], "r");
    if (file_var == NULL) {
        printf("File %s is not found \n", argv[2]);
        printf("Enter a key to continue...\n");
        getchar();
    }
    else {
        // выполнение программы, если файлы загружены и синтаксический разбор
        не имеет ошибок
        if (load_prog(file_prog, list_prog) && load_var(file_var, list_var))
        {
            if (checkSyntax(file_out, list_prog, list_var, list_command,
list_init))
                doProg(file_out, list_command, list_init, comlimit);
        }
        else
            printf("Load program error\n");
        fclose(file_var);
    }
    fclose(file_prog);
}
if (file_out != NULL) fclose(file_out);

deleteDbllinkedList(&list_prog); // списки и так пустые - удаляем сразу
deleteDbllinkedList(&list_var);

Command *buf; // удаляем, освобождая память
buf = (Command*) popFront(list_init);
while (buf != NULL) {
    free(buf->buffer);
    free(buf);
    buf = (Command*)popFront(list_init);
}
deleteDbllinkedList(&list_init);

buf = (Command*) popFront(list_command);
while (buf != NULL) {
    free(buf->buffer);
    free(buf);
}

```

```
        buf = (Command*)popFront(list_command);
    }
    deleteDblLinkedList(&list_command);
    //system("pause");
    printf("Program is completed...\n");
    getchar();
}
```

Заголовочный файл loop.h

```
#include <stdio.h>
#include <stdlib.h>

int load_prog(FILE *, DbLinkedList *);
int load_var(FILE *, DbLinkedList *);
int checkSyntax(FILE*, DbLinkedList *, DbLinkedList *, DbLinkedList *,
DbLinkedList * );
void doProg(FILE *, DbLinkedList *, DbLinkedList *, int);
```

Файл load.c

```

#include "stdafx.h"
#include <stdlib.h>
#include <string.h>
#include "ilist.h"

int load_prog(FILE *, DbllinkedList * );
int load_var(FILE *, DbllinkedList * );

// разбиваем программу через ; так чтобы было через \n в list_prog
int load_prog(FILE *file_prog, DbllinkedList * list_prog) {
    int i = 0;
    char *buffer1; // строка до ;
    char *buffer2; // LOOPx хранится в буфере
    char *buffer3; // DO
    char *buffer4; // остаток строки -> в первый и опять укорачивание
    char *buf;
    char ch;
    char *yk;

    buffer1 = (char *)malloc(1 * sizeof(char));
    buffer2 = (char *)malloc(1 * sizeof(char));
    buffer3 = (char *)malloc(3 * sizeof(char));
    buffer4 = (char *)malloc(1 * sizeof(char));

    //int lencount = 0;

    while ((ch = fgetc(file_prog)) != EOF) {
        if (!(ch == ' ' || ch == '\n' || ch == '\t')) {
            if (ch == ';') {
                buffer1 = realloc(buffer1, sizeof(char) * (i + 1));
                if (!checkMemory(buffer1)) return 0;
                buffer1[i] = '\0'; // строка кончилась
                int ex = 1;
                // несколько LOOP DO подряд
                while (ex) {
                    yk = strstr(buffer1, "DO");
                    if (yk == NULL) {
                        pushBack(list_prog, (char *)buffer1);
                        buffer1 = (char *)malloc(1 *
sizeof(char));

                        ex = 0;
                    }
                    else {
                        buffer2 = realloc(buffer2, sizeof(char) *
(yk - buffer1 + 1));

                        if (buffer2 == NULL) {
                            printf("Memory allocation error
\n");

                            return 0;
                        }
                        strcpy(buffer2, buffer1, yk - buffer1);
                        buffer2[yk - buffer1] = '\0';
                        pushBack(list_prog, (char *)buffer2);
                        buffer2 = (char *)malloc(1 *
sizeof(char));

                        strcpy(buffer3, "DO\0");
                        pushBack(list_prog, (char *)buffer3);
                        buffer3 = (char *)malloc(3 *
sizeof(char));

```

```

        (strlen(yk + 2) + 1));

        \n");

    *)buffer4);
    sizeof(char));

    sizeof(char) *(strlen(buffer4) + 1));

    error \n");

        buffer4 = realloc(buffer4, sizeof(char) *
        if (buffer4 == NULL) {
            printf("Memory allocation error
                return 0;
        }
        strcpy(buffer4, yk + 2);
        buf = strstr(buffer4, "D0");
        if (buf == NULL) {
            pushBack(list_prog, (char
                buffer4 = (char *)malloc(1 *
                    ex = 0;
                }
            else {
                buffer1 = realloc(buffer1,
                    if (buffer1 == NULL) {
                        printf("Memory allocation
                            return 0;
                    }
                    strcpy(buffer1, buffer4);
                }
            }
        }
        i = 0;
    }
    else {
        buffer1[i] = ch;
        i++;
        buffer1 = realloc(buffer1, sizeof(char) * (i + 1));
        if (buffer1 == NULL) {
            printf("Memory allocation error \n");
            return 0;
        }
    }
}

}
int exit = 1;
buffer1[i] = '\0';
free(buffer1);
free(buffer4);
buffer1 = NULL;
buffer4 = NULL;
buffer2 = NULL;
buffer3 = NULL;

//system("cls");
// печатаем список команд
if (exit)
    printDbllinkedList(list_prog);
//system("pause");
printf("Enter a key to continue...\n");
getchar();
return exit;
//system("pause");
printf("Enter a key to continue...\n");
getchar();
return 1;
}

```

```

// разбиваем программу через ; и , так чтобы было \n в list_var
int load_var(FILE *file_var, DbLinkedList * list_var) {
    //FILE *file_var;
    //file_var = fopen(varName, "r");
    int i = 0;
    char *buffer1;
    //int lencount = 0;
    char ch;
    buffer1 = (char *)malloc(1 * sizeof(char));
    while ((ch = fgetc(file_var)) != EOF) {
        if (!(ch == ' ' || ch == '\n' || ch == '\t')) {
            if (ch == ';' || ch == ',') {
                buffer1 = realloc(buffer1, sizeof(char) * (i + 1));
                if (buffer1 == NULL) {
                    printf("Memory allocation error \n");
                    return 0;
                }
                buffer1[i] = '\0';
                pushBack(list_var, (char *)buffer1);
                buffer1 = (char *)malloc(1 * sizeof(char));
                i = 0;
            }
            else {
                buffer1[i] = ch;
                i++;
                buffer1 = realloc(buffer1, sizeof(char) * (i + 1));
                if (buffer1 == NULL) {
                    printf("Memory allocation error \n");
                    return 0;
                }
            }
        }
    }
    buffer1[i] = '\0';

    //system("cls");
    // печатаем список переменных
    printDbLinkedList(list_var);
    //system("pause");
    printf("Enter a key to continue...\n");
    getchar();
    return 1;
}

//system("pause");
printf("Enter a key to continue...\n");
getchar();
return 1;
}

```

Файл isyntax.c

```

#include "stdafx.h"
#include "ilist.h"
#include "command.h"

#define INDLIMIT 2147483646 // определяем так максимальный индекс переменной

int checkpline(char*, int, DbLinkedList *, int*, int*, int*);
int checkvline(char*, int, DbLinkedList *);

int checkSyntax(FILE* file_out, DbLinkedList * list_prog, DbLinkedList * list_var,
    DbLinkedList * list_command, DbLinkedList * list_init) {

    // проверка синтаксиса программы
    int k_end, k_do, k_loop;
    k_end = 0;
    k_do = 0;
    k_loop = 0;

    int okp = 0;
    int ns = 0;
    void *buf;
    buf = popFront(list_prog);
    while (buf != NULL) {
        ns++;
        okp = okp + checkpline((char *)buf, ns, list_command, &k_loop, &k_do,
&k_end);
        buf = popFront(list_prog);
    }

    //system("cls");
    printCommandList(list_command, file_out);

    printf("Programm: %d syntax error found\n", okp);
    if (file_out != NULL)
        fprintf(file_out, "Programm: %d syntax error found\n", okp);
    if (k_end != k_loop || k_do != k_loop || k_end != k_do) {
        okp++;
        printf("Programm: %d LOOP - %d DO - %d END error found\n", k_loop, k_do,
k_end);
        if (file_out != NULL)
            fprintf(file_out, "Programm: %d LOOP - %d DO - %d END error found\n",
k_loop, k_do, k_end);
    }

    //system("pause");
    printf("Enter a key to continue...\n");
    getchar();

    // проверка синтаксиса переменных
    ns = 0;
    int okv = 0;
    buf = popFront(list_var);
    while (buf != NULL) {
        ns++;
        okv = okv + checkvline((char *)buf, ns, list_init);
        buf = popFront(list_var);
    }
    //system("cls");
    printCommandList(list_init, file_out);
    printf("Variable: %d syntax error found\n", okv);
}

```

```

    if (file_out != NULL)
        fprintf(file_out, "Variable: %d syntax error found\n", okv);
    //system("pause");
    printf("Enter a key to continue...\n");
    getchar();

    if (okp + okv == 0)
        return 1;
    else
        return 0;
}

// проверка строки программы
int checkpline(char* b, int ns, DbllinkedList * list_command, int * c_loop, int * c_do,
int * c_end) {
    char * yk;
    //char * e;
    Command * com;
    //extern int k_do, k_end, k_loop;

    int co, er, p1 = 0, p2 = 0, p3 = 0;
    int ok1 = -1;
    int ok2 = -1;
    int ok3 = -1;
    int ok4 = -1;
    int of;
    yk = strstr(b, "DO"); // DO входит в строку
    if (yk != NULL) {
        *c_do = *c_do + 1;
        if (strlen(b) == 2) {
            if (*c_do > *c_loop) {
                co = 1; er = 14; // количество DO некорректное
                ok1 = 1;
            }
            else {
                co = 1;      er = 0;
                ok1 = 1;
            }
        }
        else {
            co = 1;      er = 1; // неправильное написание команды DO
            ok1 = 0;
        }
    }

    if (ok1 == -1) {
        yk = strstr(b, "END"); // END входит в строку
        if (yk != NULL) {
            *c_end = *c_end + 1;
            if (strlen(b) == 3) {
                if (*c_end > *c_loop || *c_end > *c_do) {
                    co = 2; er = 15; // количество END некорректное
                    ok2 = 0;
                }
                else {
                    co = 2;      er = 0;
                    ok2 = 1;
                }
            }
            else {
                co = 2;      er = 2; // неправильное написание команды END
                ok2 = 0;
            }
        }
    }
}

```



```

    }

    if (ok2 == -1 && ok1 == -1) {
        yk = strstr(b, "LOOPx"); // LOOPx в строке
        if (yk != NULL) {
            *c_loop = * c_loop + 1;
            if (yk - b != 0 || strlen(b) == 5) {
                co = 3;          er = 3; // перед LOOPx некорректная запись
                ok3 = 0;
            }
            else {
                of = 1;
                for (size_t i = 5; i < strlen(b); i++) {
                    if ((int)b[i] < 48 || (int)b[i] > 57) { // после
x стоит число, состоящее не из цифр (проверка по ASCII)
                        of = 0;
                    }
                    if (!of) {
x после LOOP
                        co = 3;          er = 4; // ошибка в индексе
                        ok3 = 0;
                    }
                    else {
индекс x
                        p1 = atoi(yk + 5); // в p1 будет храниться

                        if (p1 <= INDLIMIT && p1 >= 0) {
                            co = 3;          er = 0;
                            ok3 = 1;
                        }
                        else {
индексов
                            co = 3; er = 24; // превышен лимит
                            ok3 = 0;
                        }
                    }
                }
            }
        }
    }

    if (ok3 == -1 && ok2 == -1 && ok1 == -1) {
        yk = strstr(b, ":="); // := в строке
        if (yk != NULL) {
            int bsc = strlen(b);
            char *left = malloc(bsc * sizeof(char)); // левая часть
выражения xN:=xM+C, т.е xN
            char *right = malloc(bsc * sizeof(char)); // правая
часть выражения xN:=xM+C, т.е xM+C
            char *rleft = malloc(bsc * sizeof(char)); // xM
            char *rright = malloc(bsc * sizeof(char)); // C
            strncpy(left, b, yk - b);
            left[yk - b] = '\0';
            strcpy(right, yk + 2);
            yk = strstr(left, "x"); // x в левой части строки
            if (yk == NULL || yk - left != 0 || strlen(yk) == 1) {
команды левой части перед :=
                co = 4;          er = 6; // неправильное написание
                ok4 = 0;
            }
            else {
                of = 1;
                for (size_t i = 1; i < strlen(left); i++) {
                    if ((int)left[i] < 48 || (int)left[i] >
57) { // после x не стоит число, состоящее из цифр (проверка по ASCII)

```

```

        of = 0;
    }
}
if (!of) {
    co = 4;      er = 7; // ошибка в индексе
    ok4 = 0;
}
else {
    p1 = atoi(left + 1); // в p1 будет
    if (p1 > INDLIMIT || p1 < 0) {
        co = 4; er = 24;
        ok4 = 0;
    }
}
}
if(ok4 != 0) {
    yk = strstr(right, "+"); // + в правой части
    int plus = 1;
    if (yk == NULL) {
        yk = strstr(right, "-"); // - в правой
        plus = 0;
    }
    if (yk == NULL) {
        co = 4;      er = 8; // не найден +/-
        ok4 = 0;
    }
    else {
        strncpy(rleft, right, yk - right);
        rleft[yk - right] = '\0';
        strcpy(rright, yk + 1);
        yk = strstr(rleft, "x");
        if (yk == NULL || yk - rleft != 0 ||
            strlen(yk) == 1) {
            co = 4;      er = 5; //
            ok4 = 0;
        }
        else {
            of = 1;
            for (size_t i = 1; i <
                strlen(rleft); i++) {
                if ((int)rleft[i] < 48 ||
                    (int)rleft[i] > 57) // после x не стоит число, состоящее из цифр (проверка по ASCII)
                    of = 0;
                if (!of) {
                    co = 4;      er = 5; //
                    ok4 = 0;
                }
            }
            else {
                p2 = atoi(rleft + 1); // в
                if (p2 > INDLIMIT || p2 < 0)
                    co = 4; er = 24;
                    ok4 = 0;
                }
            }
            else {
                of = 1;
            }
        }
    }
}

```

x перед :=

храниться индекс x

строки

части строки

неправильное написание команды левой части перед +-

неправильное написание команды левой части перед +-

p2 будет храниться индекс xM

{


```

char *right = malloc(bsc * sizeof(char));
strncpy(left, b, yk - b); // xN, где xN=C
left[yk - b] = '\0';
strcpy(right, yk + 1); // C, где xN=C
yk = strstr(left, "x");
if (yk == NULL || yk - left != 0 || strlen(left) == 1) {
    co = 6;      er = 10; // неправильное написание x в левой части
}
else {
    of = 1;
    for (size_t i = 1; i < strlen(left); i++) {
        if ((int)left[i] < 48 || (int)left[i] > 57) {
            of = 0;
        }
    }
    if (!of) {
        co = 6;      er = 11; // после x не цифровые символы
    }
    else {
        p1 = atoi(left + 1); // в p1 будет храниться индекс x в левой
части

        if (p1 > INDLIMIT || p1 < 0) {
            co = 6; er = 24; // индекс превышен
        }
        else {
            of = 1;
            for (size_t i = 1; i < strlen(right); i++)
                if ((int)right[i] < 48 || (int)right[i] > 57)
//!!!!
                    of = 0;
            if (!of) {
                co = 6;      er = 12; // некорректная запись в
правой части

            }
            else {
                p2 = atoi(right); // в p2 будет константа правой
части

                co = 6; er = 0;
            }
        }
    }
}
free(left);
free(right);
}
else {
    co = 7; er = 13;
}
com = createCommand(ns, b, co, er, p1, p2, p3);
pushBack(list_init, com);

if (er == 0)
    return 0;
else
    return 1;
}

```

Файл command.c

```

#include "stdafx.h"
#include "ilist.h"
#include "command.h"

void doProg(FILE *f, DbllinkedList * list_command, DbllinkedList * list_init, int
comlimit) {

    DbllinkedList * list_varvalue = createDbllinkedList();
    DbllinkedList * list_loop = createDbllinkedList();

    Node *tmp = list_init->head; // встаем в начало списка команд для инициализации
переменных
    int ok = 1;
    while (tmp && ok) {
        ok = doCommand((Command *)tmp->value, list_loop, list_varvalue, &tmp); //
выполняем поочередно команды инициализации переменных
        printCommand(tmp->value, f); // печать команды
        printVarLoop(f, list_loop, list_varvalue); // печать списка list_varvalue и
list_loop
        tmp = tmp->next;
    }
    if (!ok) printf("Variable initialisation error\n");

    tmp = list_command->head; // встаем в начало списка команд
    ok = 1;
    int cnt_temp = 0;
    int cnt_all = 0;
    int decision = 0; // решение продолжить ли работу программы
    while ((tmp && ok) && (decision == 0)) {
        ok = doCommand((Command *)tmp->value, list_loop, list_varvalue, &tmp);
        if (ok != -1) {
            printCommand(tmp->value, f);
            printVarLoop(f, list_loop, list_varvalue);
        }
        tmp = tmp->next;
        cnt_temp++;
        if (cnt_temp == comlimit)
        {
            cnt_all = cnt_all + cnt_temp;
            cnt_temp = 0;
            printf("\n%d commands completed. \n0 - continue \n1 - exit \n",
cnt_all);

            scanf("%d", &decision);
            if (decision == 1)
                printf("The program is interrupted by user \n");
        }
    }
    if (!ok) printf("The program is interrupted\n");

    if (findvar(0, list_varvalue)) {
        printf("x0 = %d\n", getvar(0, list_varvalue));
        if (f != NULL) fprintf(f, "x0 = %d\n", getvar(0, list_varvalue));
    }
    else {
        printf("x0 is not initialised\n");
        if (f != NULL) fprintf(f, "x0 is not initialised\n");
    }

    VarEl *buf1; // удаляем, освобождая память
    buf1 = (VarEl *) popFront(list_varvalue);

```

```

while (buf1 != NULL) {
    free(buf1);
    buf1 = (VarEl *) popFront(list_varvalue);
}
deleteDbllinkedList(&list_varvalue);
LoopEl *buf2;
buf2 = (LoopEl *) popFront(list_loop);
while (buf2 != NULL) {
    free(buf2);
    buf2 = (LoopEl *) popFront(list_loop);
}
deleteDbllinkedList(&list_loop);
}

int doCommand(Command * com, DbllinkedList * list_loop, DbllinkedList * list_varvalue,
Node** tp) {
    if (!checkloop(list_loop) && com->command != 2) // выход, если последний loop
    нулевой и не END
        return -1;
    int ok = 1;
    int val;
    switch (com->command) {
    case 1: // DO
        break;
    case 2: // END
        *tp = getloop(list_loop, tp);
        break;
    case 3: // LOOPx
        if (!findvar(com->p1, list_varvalue)) {
            com->error = 22;
            ok = 0;
        }
        else
            if (!putloop(com->p1, *tp, list_loop, list_varvalue)) {
                com->error = 21;
                ok = 0;
            }
        break;
    case 4: // :=+
        if (!findvar(com->p2, list_varvalue)) {
            com->error = 22;
            ok = 0;
        }
        else {
            val = getvar(com->p2, list_varvalue);
            putvar(com->p1, val + com->p3, list_varvalue);
        }
        break;
    case 5: // :=-
        if (!findvar(com->p2, list_varvalue)) {
            com->error = 22;
            ok = 0;
        }
        else {
            val = getvar(com->p2, list_varvalue);
            putvar(com->p1, val - com->p3, list_varvalue);
        }
        break;
    case 6: // =
        putvar(com->p1, com->p2, list_varvalue);
        break;
    }
    return ok;
}

```

```

}
// печать команды в соответствии с разбором ошибок
void printCommand(void * value, FILE *f) {
    char * er = "";
    Command * com;
    com = (Command *)value;
    er = "\0";
    switch (com->error) {
    case 0:
        er = "\0";
        break;
    case 1:
        er = " er: 1 Garbage characters in the use of D0";
        break;
    case 2:
        er = " er: 2 Garbage characters in the use of END";
        break;
    case 3:
        er = " er: 3 Garbage characters in the use of LOOP-X";
        break;
    case 4:
        er = " er: 4 Garbage characters in the index of LOOP x";
        break;
    case 5:
        er = " er: 5 Garbage characters in the left part of +-";
        break;
    case 6:
        er = " er: 6 Garbage characters in the left part of := ";
        break;
    case 7:
        er = " er: 7 Garbage characters in the index of left part";
        break;
    case 8:
        er = " er: 8 Plus or minus are not found";
        break;
    case 9:
        er = " er: 9 There is unrecognised command";
        break;
    case 10:
        er = " er: 10 Garbage characters in the use of x";
        break;
    case 11:
        er = " er: 11 Garbage characters in the index of x";
        break;
    case 12:
        er = " er: 12 Garbage characters in the right part of = ";
        break;
    case 13:
        er = " er: 13 There is unrecognised command";
        break;
    case 14:
        er = " er: 14 Count of D0 greater than LOOP";
        break;
    case 15:
        er = " er: 15 Count of END greater than LOOP or D0";
        break;
    case 21:
        er = " er: 21 The x is not available";
        break;
    case 22:
        er = " er: 22 The x is not initialised";
        break;
    case 23:
        er = " er: 23 Garbage characters in the right part of +- \0";

```

```

        break;
    case 24:
        er = "  er: 24 Limit of index is exceeded \0";
        break;
    }
    printf("%3d  %s %s  \n", com->ns, com->buffer, er);
    if (f != NULL)
        fprintf(f, "%3d  %s %s  \n", com->ns, com->buffer, er);
}

Command * createCommand(int i, char *b, int c, int e, size_t p1, size_t p2, size_t p3) {
    Command * com = (Command *)malloc(sizeof(Command));
    if (com != NULL) {
        com->ns = i;
        com->buffer = b;
        com->error = e;
        com->command = c;
        com->p1 = p1;
        com->p2 = p2;
        com->p3 = p3;
    }
    return com;
}

Node * getloop(DbllinkedList * list_loop, Node ** tp) {
    LoopEl *le = (LoopEl *)popBack(list_loop);
    if (le != NULL) {
        le->count--;
        Node *yk = le->yk;
        if (le->count < 1) {
            free(le);
            return *tp; // указатель на текущий узел
        }
        else
            pushBack(list_loop, le);
        return yk; // указатель на узел, с которого надо повторять
    }
    else
        return *tp; // указатель на текущий узел
}

int checkloop(DbllinkedList * list_loop) {
    Node * t = list_loop->tail;
    if (t == NULL)
        return 1;
    LoopEl *le = (LoopEl *)t->value;
    if (le->count < 1)
        return 0;
    else
        return 1;
}

int putloop(int p1, Node *tmp, DbllinkedList * list_loop, DbllinkedList * list_varvalue)
{
    if (findvar(p1, list_varvalue)) {
        int val = getvar(p1, list_varvalue);
        LoopEl *le = createLoopEl(val, tmp->next);
        pushBack(list_loop, le);
        return 1;
    }
    else
        return 0;
}

```



```

void putvar(int p1, int value, DbllinkedList * list_varvalue) {
    Node * t = list_varvalue->head;
    VarEl * ve;
    while (t) {
        ve = (VarEl *)t->value;
        if (ve->index == p1) {
            ve->value = value;
            break;
        }
        t = t->next;
    }
    if (t == NULL) {
        ve = createVarEl(p1, value);
        pushBack(list_varvalue, (void *)ve);
    }
}

int getvar(int p1, DbllinkedList * list_varvalue) {
    int value = 0;
    Node * t = list_varvalue->head;
    VarEl * ve;
    while (t) {
        ve = (VarEl *)t->value;
        if (ve->index == p1) {
            value = ve->value;
            break;
        }
        t = t->next;
    }
    return value;
}

int findvar(int p1, DbllinkedList * list_varvalue) {
    int value = 0;
    Node * t = list_varvalue->head;
    VarEl * ve;
    while (t) {
        ve = (VarEl *)t->value;
        if (ve->index == p1) {
            value = 1;
            break;
        }
        t = t->next;
    }
    return value;
}

LoopEl * createLoopEl(int i, Node * tmp) {
    LoopEl *le = (LoopEl *)malloc(sizeof(LoopEl));
    if (le != NULL) {
        le->count = i;
        le->yk = tmp;
    }
    return le;
}

void printVarLoop(FILE * f, DbllinkedList * list_loop, DbllinkedList * list_varvalue) {
    Node *tmp;
    tmp = list_varvalue->head;
    printf(" ");
    if (f != NULL) fprintf(f, " ");
    while (tmp) {

```

```

        VarEl * ve = (VarEl *)tmp->value;
        printf("x%d=%d ", (int)ve->index, (int)ve->value);
        if (f != NULL) fprintf(f, "x%d=%d ", (int)ve->index, (int)ve-
>value);
        tmp = tmp->next;
    }
    tmp = list_loop->head;
    while (tmp) {
        if (tmp->value != NULL) {
            LoopEl * le = (LoopEl *)tmp->value;
            Node * no = (Node *)le->yk;
            Command * co = (Command *)no->value;
            printf("%d,%d ", (int)co->ns, (int)le->count);
            if (f != NULL) fprintf(f, "%d,%d ", (int)co->ns, (int)le->count);
        }
        tmp = tmp->next;
    }
    printf("\n");
    if (f != NULL) fprintf(f, "\n");
}
void printCommandList (DbllinkedList * list, FILE * f) {
    Node *tmp;
    tmp = list->head;
    int i = 0;
    while (tmp) {
        if (tmp->value != NULL) {
            printCommand(tmp->value, f);
        }
        tmp = tmp->next;
        i++;
    }
    printf("\n");
    if (f != NULL) fprintf(f, "\n");
}
VarEl * createVarEl(int index, int value) {
    VarEl *ve = (VarEl *)malloc(sizeof(VarEl));
    if (ve != NULL) {
        ve->index = index;
        ve->value = value;
    }
    return ve;
}

```

Заголовочный файл command.h

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#pragma once
#ifndef COMMAND_H
#define COMMAND_H

typedef struct _Command {
    int ns; // номер строки
    char * buffer; // указатель на строку с командой
    int command; // номер команды
    int error; // номер ошибки
    size_t p1; // индекс переменной слева или переменной в LOOP
    size_t p2; // индекс переменной справа или константа
    size_t p3; // константа
} Command;

typedef struct _LoopEl {
    int count;
    Node *yk; // указатель на узел, который указывает на элемент типа Command, в
    котором LOOP, с которого надо повторять
} LoopEl;

typedef struct _VarEl {
    int index;
    int value;
} VarEl;

// #endif COMMAND_H

Command * createCommand(int, char *, int, int, size_t, size_t, size_t);
void printCommand(void *, FILE *);
int doCommand(Command *, DbllinkedList *, DbllinkedList *, Node **);
LoopEl * createLoopEl(int, Node *);
void printVarLoop(FILE *, DbllinkedList *, DbllinkedList *);
void printCommandList(DbllinkedList *, FILE *);

Node * getloop(DbllinkedList *, Node **);
int putloop(int p1, Node *tmp, DbllinkedList *, DbllinkedList *);
void putvar(int p1, int value, DbllinkedList *);
int getvar(int p1, DbllinkedList *);
int findvar(int p1, DbllinkedList *);
VarEl * createVarEl(int, int);
int checkloop(DbllinkedList *);
#endif COMMAND_H

```

Файл ilist.c

```

#include "stdafx.h"
#include "ilist.h"

// реализация двусвязного списка взята из интернета

// создание списка
DbLinkedList* createDbLinkedList() {
    DbLinkedList *tmp;
    tmp = (DbLinkedList*)malloc(sizeof(DbLinkedList));
    tmp->size = 0;
    tmp->head = tmp->tail = NULL;
    return tmp;
}

// удаление списка
void deleteDbLinkedList(DbLinkedList **list) {
    Node *tmp = (*list)->head;
    Node *next = NULL;
    while (tmp) {
        next = tmp->next;
        free(tmp);
        tmp = next;
    }
    free(*list);
    (*list) = NULL;
}

// вставка вперед
void pushFront(DbLinkedList *list, void *data) {
    Node *tmp = (Node*)malloc(sizeof(Node));
    if (tmp == NULL) {
        return;
    }
    tmp->value = data;
    tmp->next = list->head;
    tmp->prev = NULL;
    if (list->head) {
        list->head->prev = tmp;
    }
    list->head = tmp;

    if (list->tail == NULL) {
        list->tail = tmp;
    }
    list->size++;
}

// удаление первого
void* popFront(DbLinkedList *list) {
    Node *prev;
    void *tmp;
    if (list->head == NULL) {
        return NULL;
    }

    prev = list->head;
    list->head = list->head->next;
    if (list->head) {
        list->head->prev = NULL;
    }
    if (prev == list->tail) {

```

```

        list->tail = NULL;
    }
    tmp = prev->value;
    free(prev);

    list->size--;
    return tmp;
}

// вставка в конец
void pushBack(DbllinkedList *list, void *value) {
    Node *tmp = (Node*)malloc(sizeof(Node));
    if (tmp == NULL) {
        return;
    }
    tmp->value = value;
    tmp->next = NULL;
    tmp->prev = list->tail;
    if (list->tail) {
        list->tail->next = tmp;
    }
    list->tail = tmp;

    if (list->head == NULL) {
        list->head = tmp;
    }
    list->size++;
}

// удаление из конца
void* popBack(DbllinkedList *list) {
    Node *next;
    void *tmp;
    if (list->tail == NULL) {
        return NULL;
    }

    next = list->tail;
    list->tail = list->tail->prev;
    if (list->tail) {
        list->tail->next = NULL;
    }
    if (next == list->head) {
        list->head = NULL;
    }
    tmp = next->value;
    free(next);

    list->size--;
    return tmp;
}

// указатель на i элемент
Node* getNth(DbllinkedList *list, size_t index) {
    Node *tmp = list->head;
    size_t i = 0;

    while (tmp && i < index) {
        tmp = tmp->next;
        i++;
    }

    return tmp;
}

```

```

}

// печать списка
void printDbllinkedList(DbllinkedList *list) {
    Node *tmp;
    tmp = list->head;
    int i = 0;
    while (tmp) {
        if (tmp->value != NULL) {
            printf("%s \n", (char *)tmp->value);
        }
        tmp = tmp->next;
        i++;
    }
    printf("\n");
}

int checkMemory(void *buffer){
    if (buffer == NULL) {
        printf("Memory allocation error \n");
        return 0;
    }
    else return 1;
}

```

Заголовочный файл ilist.h

```

#include <stdio.h>
#include <malloc.h>
#pragma once
#ifndef ILIST_H
#define ILIST_H
// узел
typedef struct _Node {
    void *value;
    struct _Node *next;
    struct _Node *prev;
} Node;

// список
typedef struct _DblLinkedList {
    size_t size;
    Node *head;
    Node *tail;
} DblLinkedList;
#endif ILIST_H

DblLinkedList* createDblLinkedList();
void deleteDblLinkedList(DblLinkedList **list);
void pushFront(DblLinkedList *list, void *data);
void* popFront(DblLinkedList *list);
void pushBack(DblLinkedList *list, void *value);
void* popBack(DblLinkedList *list);
Node* getNth(DblLinkedList *list, size_t index);
void printDblLinkedList(DblLinkedList *list);
int checkMemory(void *buffer);

```