НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра системного програмування і спеціалізованих комп’ютерних систем

**КУРСОВА РОБОТА**

з дисципліни «Системне програмування»

на тему: Розробка компілятора програм мовою Асемблера

Студентки ІІ курсу групи КВ-32

напряму підготовки

6.050102 «Комп’ютерна інженерія»

Ахмедова Д.Н.

(прізвище та ініціали)

Керівник Тесленко О.К.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Члени комісі\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали

Київ- 2015 рік

**Технічне завдання**

На виконання курсової роботи по курсу

**"Системне программування"**

1. **Тема курсової роботи** - "Розробка транслятора програм на мові Асемблера. Варіант 29."
2. **Загальні вимоги**.
   1. Вхідні дані транслятора - текстовий файл з довільною програмою на мові Асемблера, складеною в відповідності з обмеженнями, які задані в варіанті курсової роботи. Для підготовки програми на мові Асемблера використовується, наприклад, стандартний додаток OS Windows Блокнот.
   2. На всі синтаксичні конструкції (ідентифікатори, константи, директиви, машинні команди, режими адресаціїї і т.д.), які допускаються в TASM(MASM) і які виходять за рамки обмежень в варіанті курсової роботи повинно видаватись діагностичне повідомлення про синтаксичну помилку.
   3. В результаті роботи транслятора повинен бути створений текстовий файл лістінга (розширення .lst). Формат файла лістінга повинен співпадать з форматом файла лістінга MASMабо TASM. Діагностичні повідомлення формуються на українській мові. Таблиця символів в файлі лістінга може бути в довільному форматі.
   4. Транслятор повинен аналізувати командний рядок, в якому задаються імена початкового файла та файла лістінга. Всі діагностичні повідомлення, які формуються в файлі лістінга додатково повинні виводитись на екран монітора. Крім того, на екран виводиться загальна кількість помилок, виявлених в початковій програмі.
3. **Вимоги до мови програмування**.
   1. Рекомендується модулі транслятора створювати на мові Паскаль (Delphi) або C++, з обов’язковим і обґрунтованим використанням Асемблерних вставок..
4. **Вимоги до оформлення результатів курсової роботи**.
   1. Для допуску до приймання курсової роботи необхідно підготувати:
      1. Роздруковану та підписану автором Пояснювальну записку
      2. Текстовий файл, який містить пояснювальну записку
      3. Файли початкових модулів транслятора
      4. Виконавчий файл транслятора
      5. Файли тестових програм
   2. Пояснювальна записка повинна містить:
      1. Титульну частину з загальноприйнятим вмістом
      2. Технічне завдання, включаючи номер та зміст варіанту
      3. Календарний план виконання етапів робіт. ,
      4. Опис розроблених модулів та підпрограмм
      5. Додаток 1 з текcтами початкових модулів транслятора
      6. Додаток 2 з двома тестовими прикладами - один без помилок, другий з типовими помилками
      7. Додаток 3 з роздруківками файлів лістінга, сформованими розробленим транслятором із початкових файлів з Додатку 2
      8. Додаток 4 з роздруківками файлів лістінга, сформованими транслятором MASM ( абоTASM) із початкових файлів з Додатку 2
5. **Етапи виконання роботи**
   1. Створення на базі варіанту завдання та погодження з викладачемтестового файлу
   2. Розробка лексичного аналізатора
   3. Розробка програми 1-го перегляду (формування таблиці ідентифікаторів, визначення кількості байт, які будуть формуватись по кожній інструкції)
   4. Розробка програми 2-го перегляду (генерування команд та даних, формування файла лістінга)
6. **Література**
   1. Зубков С.В. Assembler. Язык неограниченных возможностей. Приложение 2. Команды Intel 80x86. М.2001
   2. Юров В. Assembler. Учебный курс. Урок 5. Синтаксис ассемблера.С-Пб. 2001.
   3. Квиттнер П. Задачи Программ Вычисления Результаты. Гл 5. Языки ассемблеров и ассемблеры. М. Мир. 1980.

Лебедев В.Н. Введение в системы программирования. (в части организации вычисления выражений). М. Статистика. 1975.

**Варіант 29.**

*Ідентифікатори*

Містять великі і малі букви латинского алфавіту та цифри. Починаються з букви. Великі та малі букви не відрізняються. Довжина ідентифікаторів не більше 8 символів

*Константи*

Шістнадцятерічні, десяткові, двійкові та текстові константи

*Директиви*

END,

SEGMENT - без операндів, ENDS, программа може мати тільки один сегмент кодів і тільки один сегмент даних

EQU

IF-ELSE-ENDIF

DB,DW,DD з одним операндом - константою (строкові константи тільки для DB)

*Розрядність даних та адрес*

16 - розрядні дані та зміщення в сегменті, 32-розрядні дані та зміщення не використовуються

*Адресація операндів пам'яті*

Базова адресація ([bx+5] і т.п.) з оператором визначення типу (ptr) при необхідності.

*Заміна сегментів*

Префікси заміни сегментів можуть задаватись тільки явно

*Машинні команди*

Cli

Inc **reg**

Dec **mem**

Or **reg,reg**

Cmp **reg,mem**

And **mem,reg**

Mov **reg,imm**

Add **mem,imm**

Jna

Де **reg –** 8 або 16-розрядні РЗП

**mem –** адреса операнда в пам’яті

**imm -** 8 або 16-розрядні безпосередні дані (константи)

**Опис загальної структури розробленої програми, окремих модулів і підпрограм та їх взаємодії**

Программа складається з 8 заголовочних файлів та 13 файлів коду .cpp

Файл **Main.h**:

Клас **AllTab** – головний клас в якому містяться все потрібні таблиці.

**Методи класу** **AllTab**:

AllTab() – конструктор за замовчуванням, виконує необхідну початкову ініціалізацію полів класу.

DynTab\* FindLex(int) – пошук лексеми за номером

Bool IsID(char\*) – перевірка ідентифікаторів на повторення в таблиці ідентифікаторів

Int WeightLabel(int) – визначає вагу мітки

~AllTab() – деструктор за замовчуванням

**Поля класу AllTab**:

Class DynTab \*LexTab – таблиця лексем

Int number\_str\_lex – кількість лексем

Class DynTab\_sint \*TabSentStruct – таблиця структури речень

Class Dyn\_Tab\_Segment \*SegmentTab – таблиця сегментів

Class DynTabIdentificator \*TabIdentificator – таблиця ідентифікаторів

**Функції поза класом**:

Int degree(int x, int n, int result) – функція, яка рахує степень числа

**Залежності**:

#include “error.h”

#include “func\_lex.h”

#include “sint\_func.h”

#include “segment”

#include “ident.h”

#include “code.h”

**Виконавчий файл** Main\_func.cpp

**Файл Func\_lex.h**

Включає клас tab\_lex – містить один рядок таблиці лексем.

**Методи класу** tab\_lex :

tab\_lex() – конструктор за замовчуванням, виконує необхідну початкову ініціалізацію полів класу.

void Set\_lex(char\*,int) – встановлює ім’я та довжину лексеми.

void Set\_type\_lex(char\*) - встановлює тип лексеми.

void check\_type\_lex(char\*) – перевіряє якого типу повинна бути лексема

void Set\_number(int) – установка номеру лексеми

int Get\_number() – повернення номеру лексеми

bool cmpLex(char\*) – порівнює лексему з строкою

char\* GetLex() – повернає ім’я лексеми

int Get\_len\_lex() – повертає довжину лексеми

char\* Get\_type\_lex() – повертає тип лексеми

bool type\_lexIs(char\*) – порівнює тип лексеми з строкою

static void SetEND(char\*, int) – установлює строку кінця сегмента

static void SetSTART(char\*, int) – установлює строку початку сегмента

static int GetEND(char\*) – повертає строку кінця сегмета

static int GetSTART(char\*) – повертає строку початку сегмента

static void SetIDEND(char\*, int) – установлює номер ідентифікатора кінця сегмента

static void SetIDSTART(char\*, int) – установлює номер ідентифікатора початку сегмента

static int GetIDEND(char\*) - повертає номер ідентифікатора кінця сегмента

static int GetIDSTART(char\*) – повертає номер ідентифікатора початку сегмента

void print() – друк у файл

~tab\_lex() – деструктор за замовчуванням

**Поля класу** tab\_lex:

int number - номер лексеми

char \*lex – ім’я лексеми

int len\_lex – довжина лексеми

char \*type\_lex – тип лексеми

static int DataEnd – номер строки кінця сегмента даних

static int IDdataEnd – номер ідентифікатора кінця сегмента даних

static int DataStart – номер строки початку сегмента даних

static int IDdataStart - номер ідентифікатора початку сегмента даних

static int CodeEnd - номер строки кінця сегмента коду

static int IDcodeEnd - номер ідентифікатора кінця сегмента коду

static int CodeStart - номер строки початку сегмента коду

static int IDcodeStart - номер ідентифікатора початку сегмента коду

-------------------------------------------------------------------------------------------------

Включає клас DynTab –двузв’язний список, який містить клас tab\_lex.

**Поля класу** DynTab:

tab\_lex Inf – об’єкт класу tab\_lex, який містить 1 строку таблиці лексем

class DynTab \*Next – вказівник на наступний елемент

class DynTab \*Prev – вказівник на попередній елемент

**Функції поза класом**:

int cuter(char\* str, tab\_lex \*tab, int i, int \*numb) –« розрізає» строку на лексеми

**Виконавчий файл**:func\_lex.cpp

**Файл sint\_func.h**

Включає стуктури:

static struct equ{

char \*name - ім’я ідентифікатора equ

int value – значення ідентифікатора equ

static int n – кількість ідентифікаторів equ

int str\_number – номер ідентифікатора equ

} equ;

static struct Dyn\_equ{

struct equ Inf - об’єкт структури equ, який містить дані про 1 ідентифікатор зарезервований директивою equ

struct Dyn\_equ\* Next - вказівник на наступний елемент struct Dyn\_equ\* Prev - вказівник на попередній елемент

} Dyn\_equ;

Включає клас tab\_sint – клас в якому міститься 1 строка таблиці структури речень.

**Методи класу** tab\_sint:

void Set\_number\_lex\_field\_label(int) – встановлює номер лексеми поля мітки

void Set\_number\_first\_lex\_field\_mnemocode(int) – встановлює номер 1 лексеми поля мнемокоду

void Set\_amount\_lex\_field\_mnemocode(int) – встановлює кількість лексем поля мнемокоду

void Set\_number\_first\_lex\_operand1(int) – встановлює номер 1лексеми 1операнда

void Set\_amount\_lex\_operand1(int) – встановлює кількість лексем 1 операнда

void Set\_number\_first\_lex\_operand2(int) - встановлює номер 1лексеми 2операнда

void Set\_amount\_lex\_oprerand2(int) - встановлює кількість лексем 2 операнда

void check\_lex(char\*, DynTab\*, int\*, int, tab\_sint\*,int) – встановлює в яке поле потрібно записати лексему

static int Get\_number\_sent() - повертає кількість речень

static void INC\_number\_sent() – інкрементація кількості речень

static void SetEQU(char\*,char\*,int\*,int) – встановлення ідентифікаторів директиви equ

static int CmpEQU(char\*,int) - перевіряє на повторення ідентифікаторів директиви equ

static void IncAmountEQU() - інкрементація кількості ідентифікаторів директиви equ

static int GetAmountEQU() - повертає кількість ідентифікаторів директиви equ

static struct Dyn\_equ\* GetEQU() – повертає вказівник на 1 ідентифікатор директиви equ

void print() – друкує в файл

int Get\_number\_lex\_field\_label()– повертає номер лексеми поля мітки

int Get\_number\_first\_lex\_field\_mnemocode()– повертає номер 1 лексеми поля мнемокоду

int Get\_amount\_lex\_field\_mnemocode()– повертає кількість лексем поля мнемокоду

int Get\_number\_first\_lex\_operand1()– повертає номер 1 лексеми 1 операнда

int Get\_amount\_lex\_operand1() – повертає кількість лексем 1 операнда

int Get\_number\_first\_lex\_operand2()– повертає номер 1 лексеми 2 операнда

int Get\_amount\_lex\_operand2()– повертає кількість лексем 2 операнда

void AddWeight(int) – додає значення до ваги речення

int GetWeight() - повертає вагу речення

void SetCode(int,int) – встановлює машинний код

int GetCode(int) - повертає машинний код

void CreateCode(int) – виділяє пам’ять під машинний код

void SetCodeData(int) – встановлює машинний код в сегменті даних

void SetCodeData(int\*) – встановлює машинний код

void SetCodeTXT(int, int) – встановлює машинний код для строкової змінної

int Get\_nCode() –повертає кількість елементів масиву з машинним кодом

**Поля класу** tab\_sint:

int flag – багатозмінний прапорець

bool flagstr –прапорець

int number\_lex\_field\_label – номер лексеми поля мітки

int number\_first\_lex\_field\_mnemocode – номер 1 лексеми поля мнемокоду

int amount\_lex\_field\_mnemocode –кількість лексем поля мнемокоду

int number\_first\_lex\_operand1 – номер 1 лексеми 1операнда

int amount\_lex\_operand1 – кількість лексем 1 операнда

int number\_first\_lex\_operand2 – номер 1 лексеми 2 операнда

int amount\_lex\_operand2 – кількість лексем 2 операнда

int weight – вага речення

int \*Code – машинний код

int nCode – довжина машинного коду

static int number\_sent – кількість речень

static struct Dyn\_equ \*EQU - таблиця ідентифікаторів дерективи equ

static int amount\_EQU – кількість ідентифікаторів дерективи equ

----------------------------------------------------------------------------------------------------

Включає клас DynTab\_sint –двузв’язний список, який містить клас tab\_sint.

**Поля класу** DynTab\_sint:

tab\_sint Inf – об’єкт класу tab\_lex, який містить 1 строку таблиці структур речень

class DynTab\_sint \*Next – вказівник на наступний елемент

class DynTab\_sint \*Prev – вказівник на попередній елемент

**Функції поза класом**:

int cuter\_sint(char\* str, DynTab \*tab\_lex, tab\_sint \*tab, int \*numb, int number\_str, DynTab\_sint\*,int) – «розрізає» строку на лексеми

void directEQU(char\* str, int\* numb,int str\_number) – обробляє дерективу equ

int directIF(char \*str, DynTab\_sint \*tab, int \*numb) –обробляє дерективу if

void amountWord(char \*str, int\* numb) – рахує кількість слів

**Залежності**:

#include "error.h"

#include "func\_lex.h"

**Виконавчий файл**:sint\_func.cpp, sint.cpp

**Файл segment.h**

Включає клас tab\_segment – містить 1 рядок таблиці сегментів.

**Методи класу** tab\_segment:

tab\_segment() - конструктор

void SetName(char\*) – встановлює ім’я сегменту

char\* GetName() – повертає ім’я сегменту

void SetLength(int) – встановлює вагу сегменту

int GetLength() - повертає вагу сегменту

**Поля класу** tab\_segment:

char \*Name - ім’я сегменту

int Length - вагу сегменту

----------------------------------------------------------------------------------------------------

Включає клас dyn\_tab\_segment –двузв’язний список, який містить клас tab\_segment.

**Поля класу** dyn\_tab\_segment:

tab\_segment Inf – об’єкт класу tab\_segment, який містить 1 строку таблиці сегментів

class dyn\_tab\_segment \*Next – вказівник на наступний елемент

class dyn\_tab\_segment \*Prev – вказівник на попередній елемент

**Залежності**:

#include "main.h"

**Виконавчий файл:** Segment.cpp

**Файл** **ident.h**

Клас “ TabIdentificator ” – клас в якому містяться 1 строка таблиці ідентифікаторів.

**Методи класу “** TabIdentificator **”**:

TabIdentificator() -конструктор

void SetName(char\*) – встановлення імені ідентифікатора

char\* GetName() – повернення імені ідентифікатора

bool CmpName(char\*) – перевіряє чи є таке ім’я в таблиці ідентифікаторів

void SetAttr(char\*) – встановлює належність ідентифікатора сегменту

char\* GetAttr() – повертає ім’я сегменту, якому належить ідентифікатор void SetValue(int) – встановлює вагу ідентифікатора

int GetValue() – повертає вагу ідентифікатора

static void IncAmount() – інкрементація кількості ідентифікаторів

static int GetAmount() – повертає кількість ідентифікаторів

**Поля класу** TabIdentificator:

char\* Name - ім’я ідентифікатора

char\* Type – тип ідентифікатора

int Value –вага ідентифікатора

char\* Attr - ім’я сегменту, якому належить ідентифікатор

static int amount – кількість ідентифікаторів

----------------------------------------------------------------------------------------------------

Включає клас DynTabIdentificator –двузв’язний список, який містить клас TabIdentificator.

**Поля класу** DynTabIdentificator:

TabIdentificator Inf – об’єкт класу TabIdentificator, який містить 1 строку таблиці ідентифікаторів

class DynTabIdentificator \*Next – вказівник на наступний елемент

class DynTabIdentificator \*Prev – вказівник на попередній елемент

**Залежності**:

#include "main.h"

**Виконавчий файл:** ident.cpp

**Файл** **Code.h**

Клас “ instruction ” – клас в якому міститься методи для обробки мненокодів та встановлення їх машинного коду.

**Методи класу “** instruction **”**:

instruction () –конструктор

static void SetCLI(DynTab\_sint \*) – встановлення машинного коду для команди CLI

static void SetINC(DynTab\_sint \*, DynTab\*)– встановлення машинного коду для команди INC

static void SetDEC(DynTab\_sint \*, DynTab\*)– встановлення машинного коду для команди DEC

static void SetOR(DynTab\_sint \*, DynTab\*, DynTab\*)– встановлення машинного коду для команди OR

static void SetCMP(DynTab\_sint \*, DynTab \*, DynTab\*)– встановлення машинного коду для команди CMP

static void SetAND(DynTab\_sint \*, DynTab \*, DynTab\*)– встановлення машинного коду для команди AND

static void SetMOV(DynTab\_sint \*, DynTab \*, DynTab\*)– встановлення машинного коду для команди MOV

static void SetADD(DynTab\_sint \*, DynTab \*, DynTab\*)– встановлення машинного коду для команди ADD

static void SetJNA(DynTab\_sint \*,AllTab\*)– встановлення машинного коду для команди JNA

static int ModR\_M(DynTab\_sint \*, DynTab\*,int,int) - розраховує байт ModR/M

static int instruction::Base(DynTab\*) – розраховує базу

static int instruction::ChangeSeg(DynTab \*) – розраховує байт заміни сегменту

**Функції поза класом**:

int BINtoDEC(char\* str) – перетворення числа з BIN в DEC

int HEXtoDEC(char\*str) - перетворення числа з HEX в DEC

int\* DECtoHEX(int\* outArr, char \*str) - перетворення числа з DEC в HEX

**Залежності**:

#include "main.h"

#include "error.h"

**Виконавчий файл:** code.cpp , instruction.cpp

**Файл error.h**

Клас “ error ” – клас в якому містяться методи обробки помилок.

**Методи класу “** error **”**:

static void overflow(int) – друк на екран помилки (строка більше ніж 120 символів)

static void Illegal\_characters(int) – друк на екран помилки (заборонені символи)

static void Invalid\_start\_line(int) - друк на екран помилки (помилковий початок речення)

static void Invalid\_mnemonic(int) - друк на екран помилки (помилковий мнемо код)

static void Extra\_characters(int) - друк на екран помилки (зайві символи)

static void expected(int, char\*) - друк на екран помилки (очікувалось: )

static void Invalid\_const(int ) - друк на екран помилки (помилкова константа)

static void NoLebel(char \*) - друк на екран помилки ( неоголошений ідентифікатор)

static void constant(int,char\*,int) – виявляє переповнення в константі

static void memory(AllTab\*, DynTab\_sint \*, int,int) – перевірює правильність звернення до пам’яті

static void operands(AllTab\*,DynTab\_sint \*,int) –виявляє помилки в операндах

static void Exit() – завершує роботу програми при наявності помилки

**Залежності**:

#include "main.h"

#include "Code.h"

**Виконавчий файл:** error.cpp

**Код програми:**

**Main.h**

#ifndef MAIN\_H

#define MAIN\_H

#include "error.h"

#include "func\_lex.h"

#include "sint\_func.h"

#include "segment.h"

#include "ident.h"

#include "Code.h"

#include <windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <crtdbg.h> // For \_CrtSetReportMode

#include <errno.h>

#include <iostream>

class AllTab{

public:

AllTab();

~AllTab();

DynTab\* FindLex(int);

bool IsID(char\*);

int WeightLabel(int);

class DynTab \*LexTab;

int number\_str\_lex;

class DynTab\_sint \*TabSentStruct;

class dyn\_tab\_segment \*SegmentTab;

class DynTabIdentificator \*TabIdentificator;

};

extern int lex\_analizing(AllTab \*tab);

extern int sint\_analizing(AllTab \*tab);

extern void Segs(AllTab \*tab);

extern void writeLex(AllTab \*tab);

extern void writeSentStruct(AllTab \*tab);

extern void weight(AllTab \*tab);

extern void writeListing1(AllTab \*tab);

extern void writeSegments(AllTab \*tab);

extern void MakeTabIdent(AllTab \*tab);

extern void code(AllTab\* tab);

int degree(int x, int n, int result);

#endif

**Main\_func.cpp**

#include "main.h"

#include <iostream>

#include <fstream>

#include <cstdlib>

AllTab::AllTab()

{

LexTab = NULL;

TabSentStruct = NULL;

SegmentTab = NULL;

number\_str\_lex = 0;

TabIdentificator = NULL;

}

AllTab::~AllTab()

{

delete LexTab;

delete TabSentStruct;

delete SegmentTab;

delete TabIdentificator;

}

DynTab\* AllTab::FindLex(int number)

{

DynTab \*Run;

Run = LexTab;

for (int i = 1; i < number\_str\_lex; i++)

{

if (Run->Inf.Get\_number() == number)

break;

Run = Run->Next;

}

return Run;

}

bool AllTab::IsID(char\* str)

{

DynTabIdentificator \*RunID;

RunID = this->TabIdentificator;

do

{

if (RunID->Inf.CmpName(str))

return false;

RunID = RunID->Next;

} while (RunID != this->TabIdentificator);

return true;

}

int AllTab::WeightLabel(int x)

{

DynTab\_sint \*Run;

Run = this->TabSentStruct;

for (int i = 0; i < tab\_sint::Get\_number\_sent(); i++)

{

if (Run->Inf.Get\_number\_lex\_field\_label() == x)

return Run->Prev->Inf.GetWeight();

Run = Run->Next;

}

return 0;

}

int degree(int x, int n, int result)

{

if (n < 1)

return result;

return degree(x, --n, result\*x);

}

**Main.cpp**

#include "main.h"

using std::cout;

using std::cin;

using std::endl;

int main()

{

AllTab TAB;

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

lex\_analizing(&TAB);

writeLex(&TAB);

sint\_analizing(&TAB);

weight(&TAB);

Segs(&TAB);

MakeTabIdent(&TAB);

code(&TAB);

writeSentStruct(&TAB);

writeListing1(&TAB);

writeSegments(&TAB);

system("pause");

}

**Func\_lex.h**

#ifndef FUNC\_LEX\_H

#define FUNC\_LEX\_H

#include <windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <crtdbg.h> // For \_CrtSetReportMode

#include <errno.h>

#include <iostream>

class tab\_lex{

friend int cuter(char\* str, tab\_lex \*tab, int i, int \*numb);

public:

tab\_lex();

~tab\_lex();

void Set\_lex( char\*, int);

void Set\_type\_lex(char\*);

void check\_type\_lex(char\*);

void Set\_number(int);

int Get\_number();

bool cmpLex(char\*);

char\* GetLex();

int Get\_len\_lex();

char\* Get\_type\_lex();

bool type\_lexIs(char\*);

static void SetEND(char\*, int);

static void SetSTART(char\*, int);

static int GetEND(char\*);

static int GetSTART(char\*);

static void SetIDEND(char\*, int);

static void SetIDSTART(char\*, int);

static int GetIDEND(char\*);

static int GetIDSTART(char\*);

void print();

private:

int number;

char \*lex;

int len\_lex;

char \*type\_lex;

static int DataEnd;

static int IDdataEnd;

static int DataStart;

static int IDdataStart;

static int CodeEnd;

static int IDcodeEnd;

static int CodeStart;

static int IDcodeStart;

};

class DynTab {

public:

DynTab();

tab\_lex Inf;

class DynTab \*Next;

class DynTab \*Prev;

} ;

#endif

**Func\_lex.cpp**

#include "func\_lex.h"

using std::cout;

using std::endl;

int tab\_lex::DataEnd = 0;

int tab\_lex::DataStart = 0;

int tab\_lex::CodeEnd = 0;

int tab\_lex::CodeStart = 0;

int tab\_lex::IDcodeStart = 0;

int tab\_lex::IDdataEnd = 0;

int tab\_lex::IDdataStart = 0;

int tab\_lex::IDcodeEnd = 0;

tab\_lex::tab\_lex()

{

number = 0;

lex = new char[1];

lex[0] = '\0';

len\_lex = 0;

type\_lex = new char[1];

type\_lex[0] = '\0';

}

void tab\_lex::SetEND(char\* seg, int x)

{

if (!strcmp(seg, "data"))

DataEnd = x;

else

CodeEnd = x;

}

void tab\_lex::SetIDEND(char\* seg, int x)

{

if (!strcmp(seg, "data"))

IDdataEnd = x;

else

IDcodeEnd = x;

}

void tab\_lex::SetSTART(char\* seg, int x)

{

if (!strcmp(seg, "data"))

DataStart = x;

else

CodeStart = x;

}

void tab\_lex::SetIDSTART(char\* seg, int x)

{

if (!strcmp(seg, "data"))

IDdataStart = x;

else

IDcodeStart = x;

}

int tab\_lex::GetIDSTART(char\* seg)

{

if (!strcmp(seg, "data"))

return IDdataStart;

else

return IDcodeStart;

}

int tab\_lex::GetIDEND(char\* seg)

{

if (!strcmp(seg, "data"))

return IDdataEnd;

else

return IDcodeEnd;

}

int tab\_lex::GetEND(char\* seg)

{

if (!strcmp(seg, "data"))

return DataEnd;

else

return CodeEnd;

}

int tab\_lex::GetSTART(char\* seg)

{

if (!strcmp(seg, "data"))

return DataStart;

else

return CodeStart;

}

tab\_lex::~tab\_lex()

{

delete[] lex;

delete[] type\_lex;

}

void tab\_lex::Set\_lex( char\* string, int n)

{

lex = new char[n+1];

strncpy(lex, string, n);

lex[n]='\0';

len\_lex = n;

}

char\* tab\_lex::GetLex()

{

return lex;

}

void tab\_lex::Set\_type\_lex(char\* string)

{

char \*bufStr;

int size\_type\_lex = strlen(type\_lex);

if (size\_type\_lex != 0)

{

bufStr = new char[size\_type\_lex+1];

strcpy(bufStr,type\_lex);

delete[] type\_lex;

type\_lex = new char[size\_type\_lex + strlen(string) +1];

strcpy(type\_lex,bufStr);

strcat(type\_lex,string);

delete[] bufStr;

}

else

{

delete[] type\_lex;

type\_lex = new char[strlen(string) +1];

strcpy(type\_lex,string);

}

}

void tab\_lex::check\_type\_lex(char \*str)

{

if ((str[0] >= 'A' && str[0] <= 'Z') || str[0] == '\_' || str[0] == '@' || str[0] == '?' || str[0] == '$')

{

if (!strcmp(str, "AX") || !strcmp(str, "CX") || !strcmp(str, "DX") || !strcmp(str, "BX")

|| !strcmp(str, "SP") || !strcmp(str, "BP") || !strcmp(str, "SI") || !strcmp(str, "DI"))

{

Set\_type\_lex(" Ідентифікатор 16- розрядного регістра");

return;

}

if (!strcmp(str, "AL") || !strcmp(str, "CL") || !strcmp(str, "DL") || !strcmp(str, "BL")

|| !strcmp(str, "AH") || !strcmp(str, "CH") || !strcmp(str, "DH") || !strcmp(str, "BH"))

{

Set\_type\_lex( "Ідентифікатор 8- розрядного регістра");

return;

}

if (!strcmp(str, "ES") || !strcmp(str, "CS") || !strcmp(str, "SS") || !strcmp(str, "DS")

|| !strcmp(str, "FS") || !strcmp(str, "GS"))

{

Set\_type\_lex( "Ідентифікатор сегментного регістра ");

return;

}

if (!strcmp(str, "CLI") || !strcmp(str, "INC") || !strcmp(str, "DEC") || !strcmp(str, "OR")

|| !strcmp(str, "CMP") || !strcmp(str, "AND") || !strcmp(str, "MOV") || !strcmp(str, "ADD")

|| !strcmp(str, "JNA"))

{

Set\_type\_lex( "Ідентифікатор мнемокоду машинної інструкції ");

return;

}

if (!strcmp(str, "SEGMENT") || !strcmp(str, "ENDS") || !strcmp(str, "END") || !strcmp(str, "EQU")

|| !strcmp(str, "EQ") || !strcmp(str, "IF") || !strcmp(str, "ELSE") || !strcmp(str, "ENDIF"))

{

Set\_type\_lex( " Ідентифікатор директиви ");

if (!strcmp(str, "ENDS"))

if (GetIDEND("data") == 0)

SetIDEND("data", this->number);

else

if (GetIDEND("code") == 0)

SetIDEND("code", this->number);

if (!strcmp(str, "SEGMENT") )

if (GetIDSTART("data") == 0)

SetIDSTART("data", this->number);

else

if ( GetIDSTART("code") == 0)

SetIDSTART("code", this->number);

return;

}

if (!strcmp(str, "PTR"))

{

Set\_type\_lex( " Ідентифікатор оператора визначення типу ");

return;

}

if (!strcmp(str, "DB") || !strcmp(str, "DW") || !strcmp(str, "DD"))

{

switch (str[1])

{

case 'B':{Set\_type\_lex( " Ідентифікатор директиви даних тип 1"); break; }

case 'W':{Set\_type\_lex( " Ідентифікатор директиви даних тип 2"); break; }

case 'D':{Set\_type\_lex( " Ідентифікатор директиви даних тип 4"); break; }

}

return;

}

if (!strcmp(str, "BYTE") || !strcmp(str, "WORD") || !strcmp(str, "DWORD "))

{

switch (str[0])

{

case 'B':{Set\_type\_lex( " Ідентифікатор розміру 1"); break; }

case 'W':{Set\_type\_lex( " Ідентифікатор розміру 2"); break; }

case 'D':{Set\_type\_lex( " Ідентифікатор розміру 4"); break; }

}

return;

}

Set\_type\_lex( "Ідентифікатор користувача або не визначений ");

}

if (str[0] >= '0' && str[0] <= '9')

{

if (str[strlen(str) - 1] >= '0' && str[strlen(str) - 1] <= '9')

Set\_type\_lex( " Десяткова константа");

else

switch (str[strlen(str) - 1])

{

case 'B':{Set\_type\_lex( " Двійкова константа"); break; }

case 'D':{Set\_type\_lex( " Десяткова константа"); break; }

case 'H':{Set\_type\_lex( " Шістнадцятирічна константа"); break; }

}

}

if (str[0] == '\'' || str[0] == '\"')

{

Set\_type\_lex( "text константа"); return;

}

if (str[0] == ':' || str[0] == ',' || str[0] == '+' || str[0] == '-' || str[0] == '[' || str[0] == ']')

{

Set\_type\_lex( "односимвольна");

}

}

void tab\_lex::Set\_number(int numb)

{

number=numb;

}

int tab\_lex::Get\_number()

{

return number;

}

int tab\_lex::Get\_len\_lex()

{

return len\_lex;

}

char\* tab\_lex::Get\_type\_lex()

{

return type\_lex;

}

bool tab\_lex::cmpLex(char \*str)

{

if (strcmp(str,lex) == 0)

return true;

return false;

}

bool tab\_lex::type\_lexIs(char \*str)

{ char \*s;

s=strstr(type\_lex, str);

if ( s == 0)

return false;

return true;

}

void tab\_lex::print()

{

cout << number << " ";

cout << " " << lex;

cout << " " << len\_lex;

cout << " " << type\_lex << endl;

}

DynTab::DynTab()

:Inf()

{

Next=NULL;

Prev=NULL;

}

int cuter(char\* str, tab\_lex \*tab, int i, int \*numb)

{

char substr[80];

int k1=i, k2 = 0, j, n = strlen(str), flag = 0;

tab->number = \*numb;

if (str[i] == '\'' || str[i] == '"')

{

i++;

while (i<n && (str[i] != '\'' && str[i] != '"'))

{

i++;

}

if (i == n)

i--;

k2 = i;

(\*numb)++;

j = k2 - k1;

strncpy\_s(substr, &str[k1], j);

tab->check\_type\_lex(substr);

j--;

strncpy\_s(substr, &str[k1+1], j);

i++;

}

else

{

for (i = i; i < n; i++)

if (str[i] == ' ' || str[i] == '\t' || str[i] == ':' || str[i] == ','

|| str[i] == '+' || str[i] == '-' || str[i] == '[' || str[i] == ']')

{

break;

}

if (i == n && str[i - 1] == '\n')

i--;

k2 = i;

(\*numb)++;

j = k2 - k1;

strncpy\_s(substr, &str[k1], j);

\_strupr\_s(substr, sizeof(substr));

tab->check\_type\_lex(substr);

}

tab->Set\_lex( substr, j);

tab->len\_lex = j;

return i;

}

**lex.cpp**

#include "main.h"

#include "error.h"

int lex\_analizing(AllTab \*tab);

int lex\_analizing(AllTab \*tab)

{

int i, n, \*numb, first, last;

int number\_str= 0;

DynTab \*Run\_tab, \*Run2\_tab;

FILE \*file;

char \*fname = "TEST.asm";

char str[400];

numb = (int \*)malloc(sizeof(int));

\*numb = 0;

tab->number\_str\_lex=0;

tab->LexTab = NULL;

fopen\_s(&file,fname, "r");

if (file == 0)

{

printf("Файл '%s' не существует", fname);

return 0;

}

while (!feof(file))

{

fgets(str, sizeof(str), file);

number\_str++;

if (strlen(str) > 120)

{

error::overflow(number\_str);

}

first = 0;

last = 0;

for (i = 0; i < strlen(str); i++)

if (str[i] == '\'')

if (first == 0)

first = i;

else if (last == 0)

last = i;

for (i = 0; i < strlen(str); i++)

{

if (i<first || i>last)

if (isalpha(str[i]) == 0 && isdigit(str[i]) == 0 && str[i] != ' ' && str[i] !='\t' && str[i] != '\n' && str[i] !='$'

&& str[i] != '@' && str[i] != '?' && str[i] != ':' && str[i] != '+' && str[i] != '-' && str[i] != '\*'

&& str[i] != '[' && str[i] != ']' && str[i] != ',')

error::Illegal\_characters(number\_str);

}

n = strlen(str);

i = 0;

while (i < n-1)

{

Run\_tab = new DynTab;

if (tab->LexTab == NULL)

tab->LexTab = Run\_tab;

else

{

Run2\_tab->Next = Run\_tab;

Run\_tab->Prev = Run2\_tab;

}

while (str[i] == ' ' || str[i]=='\t')

i++;

if (str[i] == '\n' || str[i] == ';')

{

delete Run\_tab;

break;

}

if (str[i] == ':' || str[i] == ',' || str[i]=='+' || str[i]=='-' || str[i]=='[' || str[i]==']')

{

Run\_tab->Inf.Set\_number(\*numb);

Run\_tab->Inf.Set\_lex(&str[i],1);

Run\_tab->Inf.check\_type\_lex(&str[i]);

i++;

Run\_tab->Inf.Set\_number(\*numb);

(\*numb)++;

}

else

i = cuter(str, &(Run\_tab->Inf), i, numb);

Run2\_tab = Run\_tab;

tab->number\_str\_lex++;

}

}

Run\_tab->Next = tab->LexTab;

tab->LexTab->Prev = Run\_tab;

fclose(file);

return 0;

}

**sint\_func.h**

#ifndef SINT\_FUNC\_H

#define SINT\_FUNC\_H

#include "error.h"

#include "func\_lex.h"

#include <windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <crtdbg.h> // For \_CrtSetReportMode

#include <errno.h>

#include <iostream>

using namespace std;

static struct equ{

char \*name;

int value;

static int n;

int str\_number;

} equ;

static struct Dyn\_equ{

struct equ Inf;

struct Dyn\_equ\* Next;

struct Dyn\_equ\* Prev;

} Dyn\_equ;

class DynTab\_sint;

class tab\_sint{

friend int cuter(char\* str, tab\_lex \*tab\_lex, tab\_sint \*tab, int \*numb, DynTab\_sint\*,int);

public:

tab\_sint();

~tab\_sint();

void Set\_number\_lex\_field\_label(int);

void Set\_number\_first\_lex\_field\_mnemocode(int);

void Set\_amount\_lex\_field\_mnemocode(int);

void Set\_number\_first\_lex\_operand1(int);

void Set\_amount\_lex\_operand1(int);

void Set\_number\_first\_lex\_operand2(int);

void Set\_amount\_lex\_oprerand2(int);

void check\_lex(char\*, DynTab\*, int\*, int, tab\_sint\*,int);

static int Get\_number\_sent();

static void INC\_number\_sent();

static void SetEQU(char\*,char\*,int\*,int);

static int CmpEQU(char\*,int);

static void IncAmountEQU();

static int GetAmountEQU();

static struct Dyn\_equ\* GetEQU();

void print();

int Get\_number\_lex\_field\_label();

int Get\_number\_first\_lex\_field\_mnemocode();

int Get\_amount\_lex\_field\_mnemocode();

int Get\_number\_first\_lex\_operand1();

int Get\_amount\_lex\_operand1();

int Get\_number\_first\_lex\_operand2();

int Get\_amount\_lex\_operand2();

void AddWeight(int);

int GetWeight();

void SetCode(int,int);

int GetCode(int);

void CreateCode(int);

void SetCodeData(int);

void SetCodeData(int\*);

void SetCodeTXT(int, int);

int Get\_nCode();

int flag;

bool flagstr;

private:

int number\_lex\_field\_label;

int number\_first\_lex\_field\_mnemocode;

int amount\_lex\_field\_mnemocode;

int number\_first\_lex\_operand1;

int amount\_lex\_operand1;

int number\_first\_lex\_operand2;

int amount\_lex\_operand2;

int weight;

int \*Code;

int nCode;

static int number\_sent;

static struct Dyn\_equ \*EQU;

static int amount\_EQU;

};

class DynTab\_sint {

public:

DynTab\_sint();

tab\_sint Inf;

class DynTab\_sint \*Next;

class DynTab\_sint \*Prev;

} ;

int cuter\_sint(char\* str, DynTab \*tab\_lex, tab\_sint \*tab, int \*numb, int number\_str, DynTab\_sint\*,int);

void directEQU(char\* str, int\* numb,int str\_number);

int directIF(char \*str, DynTab\_sint \*tab, int \*numb);

void amountWord(char \*str, int\* numb);

extern int degree(int x, int n, int result);

#endif

**sint\_func.cpp**

#include "sint\_func.h"

#include <math.h>

using std::cout;

using std::endl;

int equ::n = 0;

int tab\_sint::number\_sent = 0;

struct Dyn\_equ\* tab\_sint::EQU = NULL;

int tab\_sint::amount\_EQU = 0;

tab\_sint::tab\_sint()

{

number\_lex\_field\_label=-1;

number\_first\_lex\_field\_mnemocode=-1;

amount\_lex\_field\_mnemocode=-1;

number\_first\_lex\_operand1=-1;

amount\_lex\_operand1=-1;

number\_first\_lex\_operand2=-1;

amount\_lex\_operand2=-1;

nCode = 0;

flagstr = false;

weight = 0x0000;

Code = NULL;

}

void tab\_sint::CreateCode(int x)

{

Code = new int[x];

for (int i = 0; i < x; i++)

Code[i] = 0;

}

int tab\_sint::Get\_nCode()

{

return nCode;

}

void tab\_sint::SetCode(int x, int n)

{

Code[n] = x;

nCode++;

}

void tab\_sint::SetCodeData(int x)

{

Code[0] = x;

nCode = 1;

}

void tab\_sint::SetCodeData(int \*x)

{

for (int i = 1; i <=x[0]; i++)

{

Code[i] = x[i];

nCode++;

}

if (Code[x[0]] < 16)

{

int buf = Code[x[0] ];

Code[x[0]] = 0;

Code[x[0]+1] = buf;

nCode++;

}

}

void tab\_sint::SetCodeTXT(int x, int n)

{

Code[n] = x;

nCode++;

}

int tab\_sint::GetCode(int i)

{

return Code[i];

}

void tab\_sint::AddWeight(int x)

{

weight += x;

}

int tab\_sint::GetWeight()

{

return weight;

}

tab\_sint::~tab\_sint()

{}

void tab\_sint::Set\_number\_first\_lex\_field\_mnemocode(int a)

{

number\_first\_lex\_field\_mnemocode = a;

}

void tab\_sint::Set\_number\_lex\_field\_label(int a)

{

number\_lex\_field\_label = a;

}

void tab\_sint::Set\_amount\_lex\_field\_mnemocode(int a)

{

amount\_lex\_field\_mnemocode = a;

}

void tab\_sint::Set\_number\_first\_lex\_operand1(int a)

{

number\_first\_lex\_operand1 = a;

}

void tab\_sint::Set\_amount\_lex\_operand1(int a)

{

amount\_lex\_operand1 = a;

}

void tab\_sint::Set\_number\_first\_lex\_operand2(int a)

{

number\_first\_lex\_operand2 = a;

}

void tab\_sint::Set\_amount\_lex\_oprerand2(int a)

{

amount\_lex\_operand2 = a;

}

int tab\_sint::Get\_number\_sent()

{

return number\_sent;

}

int tab\_sint::Get\_number\_lex\_field\_label()

{

return number\_lex\_field\_label;

}

int tab\_sint::Get\_number\_first\_lex\_field\_mnemocode()

{

return number\_first\_lex\_field\_mnemocode;

}

int tab\_sint::Get\_amount\_lex\_field\_mnemocode()

{

return amount\_lex\_field\_mnemocode;

}

int tab\_sint::Get\_number\_first\_lex\_operand1()

{

return number\_first\_lex\_operand1;

}

int tab\_sint::Get\_amount\_lex\_operand1()

{

return amount\_lex\_operand1;

}

int tab\_sint::Get\_number\_first\_lex\_operand2()

{

return number\_first\_lex\_operand2;

}

int tab\_sint::Get\_amount\_lex\_operand2()

{

return amount\_lex\_operand2;

}

void tab\_sint::INC\_number\_sent()

{

number\_sent++;

}

void tab\_sint::SetEQU(char\* name, char\* value, int\* numb, int str\_number)

{

struct Dyn\_equ \*Run;

int x;

Run = new struct Dyn\_equ;

Run->Inf.name = new char[strlen(name)+1];

if (EQU == NULL)

{

EQU = Run;

EQU->Next = EQU;

EQU->Prev = EQU;

}

else

{

EQU->Prev->Next = Run;

Run->Prev = EQU->Prev;

Run->Next = EQU;

EQU->Prev = Run;

}

strcpy(Run->Inf.name, name);

switch (value[strlen(value) - 1])

{

case 'B':{

value[strlen(value) - 1] = '\0';

x = BINtoDEC(value);

break;

}

case 'H':{

value[strlen(value) - 1] = '\0';

x = HEXtoDEC(value);

break;

}

default:{

x = atoi(value);

break;

}

}

Run->Inf.value = x;

Run->Inf.str\_number = str\_number;

equ.n=(\*numb);

}

int tab\_sint::CmpEQU( char\* name, int val)

{

struct Dyn\_equ \*Run;

Run = EQU;

do

{

if (strcmp(name, Run->Inf.name) == 0)

{

if (val == Run->Inf.value)

return 0;

if (val > Run->Inf.value)

return 1;

if (val < Run->Inf.value)

return -1;

}

Run = Run->Next;

} while (Run != EQU);

return 2;

}

struct Dyn\_equ\* tab\_sint::GetEQU()

{

struct Dyn\_equ \*Run;

Run = EQU;

return Run;

}

void tab\_sint::print()

{

if (number\_lex\_field\_label != -1)

printf("%7d",number\_lex\_field\_label);

else

printf(" -");

if (number\_first\_lex\_field\_mnemocode != -1)

printf("%12d",number\_first\_lex\_field\_mnemocode);

else

printf(" -");

if (amount\_lex\_field\_mnemocode != -1)

printf("%9d",amount\_lex\_field\_mnemocode);

else

printf(" -");

if (number\_first\_lex\_operand1 != -1)

printf("%13d",number\_first\_lex\_operand1);

else

printf(" -");

if (amount\_lex\_operand1 != -1)

printf("%10d",amount\_lex\_operand1);

else

printf(" -");

if (number\_first\_lex\_operand2 != -1)

printf("%11d",number\_first\_lex\_operand2);

else

printf(" -");

if (amount\_lex\_operand2 != -1)

printf("%10d",amount\_lex\_operand2);

else

printf(" -");

cout << endl;

}

DynTab\_sint::DynTab\_sint()

:Inf()

{

Next=NULL;

Prev=NULL;

}

void tab\_sint::check\_lex(char \*str, DynTab \*lex\_tab, int \*kol, int number\_str, tab\_sint \* Pr,int z)

{

DynTab \*Run;

int n=lex\_tab->Prev->Inf.Get\_number();

Run = lex\_tab;

for (int i = 0; i < \*kol; i++)

Run=Run->Next;

if ( Run->Inf.cmpLex(str) == true)

{

if (flag == 0 && Run->Inf.type\_lexIs("Ідентифікатор") == false)

{

error::Invalid\_start\_line(number\_str);

}

if (flag == 0 && Run->Inf.type\_lexIs("користувача") == true)

{

number\_lex\_field\_label = Run->Inf.Get\_number();

flag = 1;

return;

}

if (flag == 0)

flag = 1;

if (Run->Prev->Inf.type\_lexIs("користувача") == true)

if (Run->Inf.cmpLex(":"))

return;

flagstr = true;

if (Run->Inf.type\_lexIs("мнемокод") == true || Run->Inf.type\_lexIs("директиви") == true)

{

number\_first\_lex\_field\_mnemocode = Run->Inf.Get\_number();

amount\_lex\_field\_mnemocode = 1;

return;

}

if (Run->Inf.type\_lexIs("константа") == true || Run->Inf.type\_lexIs("розміру") == true || Run->Inf.type\_lexIs("регістра") == true

|| Run->Inf.type\_lexIs("односимвольна") == true || Run->Inf.type\_lexIs("визначення типу") == true

|| Run->Inf.type\_lexIs("користувача"))

{

if (Run->Inf.type\_lexIs("константа") == true)

error::constant(number\_str+1, Run->Inf.GetLex(),z);

if (str[0] == ',')

{

flag = 2;

return;

}

if (flag == 1 && number\_first\_lex\_operand1 == -1)

{

number\_first\_lex\_operand1 = Run->Inf.Get\_number();

amount\_lex\_operand1 = 1;

return;

}

else

{

if (flag == 1 && number\_first\_lex\_operand1 != -1)

{

amount\_lex\_operand1++;

return;

}

if (flag == 2 && number\_first\_lex\_operand2 == -1)

{

number\_first\_lex\_operand2 = Run->Inf.Get\_number();

amount\_lex\_operand2 = 1;

return;

}

else

{

if (flag == 2 && number\_first\_lex\_operand1 != -1)

{

amount\_lex\_operand2++;

return;

}

}

}

}

}

}

int cuter\_sint(char\* str, DynTab \*tab\_lex, tab\_sint \*tab, int \*numb, int number\_str, DynTab\_sint \* Pr,int z)

{

char substr[120];

int k1,i=0, k2 = 0, j=0, n = strlen(str);

tab->flag = 0;

while (str[i] == ' ' || str[i] == '\t')

i++;

k1=i;

for ( i=i; i< n; i++)

{

if (str[i-1] == '\'' || str[i-1] == '"')

{

k1=i;

while (i<n && str[i] != '\'' && str[i] != '"')

{

i++;

}

k2 = i+1;

j = k2 - k1-1;

strncpy\_s(substr, &str[k1], j);

k1=k2+1;

tab->check\_lex(substr, tab\_lex, numb, number\_str, &(Pr->Inf),z);

(\*numb)++;

while (str[i] == ' ' || str[i] == '\t')

i++;

i++;

}

else

if (i == n-1 || str[i] == ' ' || str[i] == '\t' || str[i-1] == ':' || str[i-1] == ',' || str[i] == ','

|| str[i-1] == '+' || str[i-1] == '-' || str[i-1] == '[' || str[i-1] == ']' )

{

k2 = i;

j = k2 - k1;

if (str[i-1] == ':' || str[i-1] == '+' || str[i-1] == '-' || str[i-1] == '[' || str[i-1] == ']')

{ char c[2];

c[0]=str[i-1];

c[1]='\0';

strncpy\_s(substr, &str[k1], j-1);

\_strupr\_s(substr, sizeof(substr));

tab->check\_lex(substr, tab\_lex, numb, number\_str, &(Pr->Inf),z);

(\*numb)++;

tab->check\_lex(c, tab\_lex, numb, number\_str, &(Pr->Inf),z);

(\*numb)++;

}

else

{

strncpy\_s(substr, &str[k1], j);

k1=k2+1;

\_strupr\_s(substr, sizeof(substr));

tab->check\_lex(substr, tab\_lex, numb, number\_str,&(Pr->Inf),z);

(\*numb)++;

}

while (str[i] == ' ' || str[i] == '\t')

i++;

k1=i;

}

}

return i;

}

void directEQU(char\* str, int\* numb, int str\_number)

{

int k1=0,k2,i=0,j,n=strlen(str);

char substr[30],\*name,\*value;

name = NULL;

value = NULL;

while (str[i] == ' ' || str[i] == '\t')

i++;

k1=i;

for (i = i; i < n; i++)

{

if ( i == n-1 || str[i] == ' ' || str[i] == '\t')

{

k2 = i;

j = k2 - k1;

strncpy\_s(substr, &str[k1], j);

\_strupr\_s(substr, sizeof(substr));

if (name == NULL)

{

name = new char[strlen(substr)+1];

strcpy(name, substr);

}

else

if ( strcmp(substr,"EQU") != 0 )

{

value = new char[strlen(substr)+1];

strcpy(value, substr);

}

while (str[i] == ' ' || str[i] == '\t')

i++;

k1 = i;

}

}

tab\_sint::SetEQU(name,value,numb,str\_number);

tab\_sint::IncAmountEQU();

delete[] name;

delete[] value;

}

int tab\_sint::GetAmountEQU()

{

return amount\_EQU;

}

void tab\_sint::IncAmountEQU()

{

amount\_EQU++;

}

int directIF(char \*str, DynTab\_sint \*tab, int\* numb)

{

int k1,k2,i=0,j,n=strlen(str);

char substr[30],\*name,\*value,\*oper;

DynTab\_sint \*Run;

name = NULL;

value = NULL;

oper = NULL;

while (str[i] == ' ' || str[i] == '\t')

i++;

i+=2;

(\*numb)++;

while (str[i] == ' ' || str[i] == '\t')

i++;

k1=i;

for (i = i; i < n; i++)

{

if ( i == n-1 || str[i] == ' ' || str[i] == '\t')

{

k2 = i;

j = k2 - k1;

strncpy\_s(substr, &str[k1], j);

\_strupr\_s(substr, sizeof(substr));

if (name == NULL)

{

name = new char[strlen(substr)+1];

strcpy(name, substr);

(\*numb)++;

}

else

if ( oper == NULL )

{

oper = new char[strlen(substr)+1];

strcpy(oper, substr);

(\*numb)++;

}

else

{ value = new char[strlen(substr)+1];

strcpy(value, substr);

(\*numb)++;

}

while (str[i] == ' ' || str[i] == '\t')

i++;

k1 = i;

}

}

Run = tab;

switch (oper[0])

{

case 'E': {

if (tab\_sint::CmpEQU(name, atoi(value)) == 0)

return 1;

return 0;

}

case 'N': {

if (tab\_sint::CmpEQU(name, atoi(value)) != 0)

return 1;

return 0;

}

case 'L':{

if (oper[1]=='T')

{

if (tab\_sint::CmpEQU(name, atoi(value)) == 1)

return 1;

return 0;

}

else

{

if (tab\_sint::CmpEQU(name, atoi(value)) >= 0)

return 1;

return 0;

}

}

case 'G':{

if (oper[1] == 'T')

{

if (tab\_sint::CmpEQU(name, atoi(value)) == -1)

return 1;

return 0;

}

else

{

if (tab\_sint::CmpEQU(name, atoi(value)) <= 0)

return 1;

return 0;

}

}

}

return -1;

}

void amountWord(char \*str, int\* numb)

{

int k1=0,i=0,n=strlen(str);

while (str[i] == ' ' || str[i] == '\t')

i++;

for (i = i; i < n; i++)

{

if (i == n-1 || str[i] == ' ' || str[i] == '\t' || str[i-1] == ':' || str[i-1] == ',' || str[i] == ','

|| str[i-1] == '+' || str[i-1] == '-' || str[i-1] == '[' || str[i-1] == ']' )

{

(\*numb)++;

if ( str[i-1] == ':' || str[i-1] == ',' || str[i-1] == ','

|| str[i-1] == '+' || str[i-1] == '-' || str[i-1] == '[' || str[i-1] == ']')

(\*numb)++;

while (str[i] == ' ' || str[i] == '\t')

i++;

k1 = i;

}

}

}

**sint.cpp**

#include "main.h"

#include "sint\_func.h"

int sint\_analizing(AllTab \*tab);

int sint\_analizing(AllTab \*tab)

{ int n, \*numb, number\_str=0;

int numbEnter = 0;

DynTab\_sint \*Run\_tab, \*Run2\_tab;

FILE \*file;

char \*fname = "TEST.asm";

char str[120];

int IFflag = -1;

numb= new int;

\*numb=0;

tab->TabSentStruct = NULL;

fopen\_s(&file,fname, "r");

if (file == 0)

{

printf("Файл '%s' не существует", fname);

return 0;

}

while (!feof(file))

{

fgets(str, sizeof(str), file);

if (strstr(str, "Segment") != 0 || strstr(str, "segment") != 0 || strstr(str, "SEGMENT") != 0)

{

if (tab\_lex::GetSTART("data") == 0)

if (number\_str == 0)

tab\_lex::SetSTART("data", 1);

else

tab\_lex::SetSTART("data", number\_str);

else

tab\_lex::SetSTART("code", number\_str - numbEnter);

fgets(str, sizeof(str), file);

number\_str++;

(\*numb) += 2;

}

while (strcmp(str,"\n") == 0 || strstr(str,"ends") !=0 || strstr(str,"ENDS") !=0 )

{

if (strstr(str, "ends") != 0 || strstr(str, "ENDS") != 0)

{

if (tab\_lex::GetEND("data") == 0)

tab\_lex::SetEND("data", number\_str);

else

tab\_lex::SetEND("code", number\_str - numbEnter-1);

(\*numb)++;

(\*numb)++;

}

numbEnter++;

fgets(str, sizeof(str), file);

number\_str++;

}

if (!(strstr(str,"endif") || strstr(str,"ENDIF")) && ( strstr(str,"if") !=0 || strstr(str,"IF") !=0 || strstr(str,"If") !=0 || strstr(str,"iF") !=0) )

{

IFflag = directIF(str,tab->TabSentStruct, numb);

fgets(str, sizeof(str), file);

number\_str++;

numbEnter++;

}

if (IFflag == 1)

{

if (strstr(str,"else") != 0 || strstr(str,"ELSE") != 0 )

{

while ( strstr(str,"endif") ==0 && strstr(str,"ENDIF") ==0)

{

amountWord(str,numb);

fgets(str, sizeof(str), file);

number\_str++;

numbEnter++;

}

}

if (strstr(str,"endif") != 0 || strstr(str,"ENDIF") != 0 )

{

amountWord(str,numb);

fgets(str, sizeof(str), file);

number\_str++;

numbEnter++;

IFflag = -1;

}

}

if (IFflag == 0)

{

while ( strstr(str,"else") ==0 && strstr(str,"ELSE") ==0)

{

number\_str++;

numbEnter++;

amountWord(str, numb);

fgets(str, sizeof(str), file);

}

number\_str++;

numbEnter++;

amountWord(str, numb);

fgets(str, sizeof(str), file);

IFflag = -1;

}

while (strstr(str,"equ") !=0 || strstr(str,"EQU") !=0 )

{ directEQU(str,numb,number\_str);

fgets(str, sizeof(str), file);

number\_str++;

numbEnter++;

(\*numb)+=3;

}

while (strcmp(str,"\n") == 0 || strstr(str,"Segment") !=0 || strstr(str,"segment") !=0)

{

if (strstr(str, "Segment") != 0 || strstr(str, "segment") != 0)

{

if (tab\_lex::GetSTART("data") == 0)

tab\_lex::SetSTART("data", number\_str);

else

tab\_lex::SetSTART("code", number\_str - numbEnter);

(\*numb)++;

(\*numb)++;

}

fgets(str, sizeof(str), file);

numbEnter++;

number\_str++;

}

if (strcmp(str,"end") == 0)

break;

n = strlen(str);

Run\_tab = new DynTab\_sint;

if (tab->TabSentStruct == NULL)

tab->TabSentStruct = Run\_tab;

else

{

Run2\_tab->Next = Run\_tab;

Run\_tab->Prev = Run2\_tab;

}

if (tab\_lex::GetEND("data") == 0)

cuter\_sint(str, tab->LexTab, &(Run\_tab->Inf), numb, number\_str, Run\_tab->Prev,0);

else

cuter\_sint(str, tab->LexTab, &(Run\_tab->Inf), numb, number\_str, Run\_tab->Prev, 1);

tab\_sint::INC\_number\_sent();

Run2\_tab = Run\_tab;

number\_str++;

if (Run\_tab->Inf.Get\_amount\_lex\_field\_mnemocode() == -1)

if (tab->FindLex(Run\_tab->Inf.Get\_number\_lex\_field\_label())->Next->Inf.cmpLex(":") == false)

error::Invalid\_mnemonic(number\_str);

error::operands(tab, Run\_tab, number\_str);

}

Run\_tab->Next = tab->TabSentStruct;

tab->TabSentStruct->Prev = Run\_tab;

return 0;

}

**weight.h**

#ifndef WEIGHT\_H

#define WEIGHT\_H

#include "main.h"

#endif

**weight.cpp**

#include "weight.h"

void weight(AllTab \*tab)

{

DynTab\_sint \*RunSent;

DynTab \*RunLex, \*RunLex2;

bool flagEnd = true;

RunSent = tab->TabSentStruct;

for (int i = 1; i <= tab\_sint::Get\_number\_sent(); i++)

{

if (flagEnd == true && i == tab\_lex::GetEND("data") )

{

flagEnd = false;

}

else

if (i != 0)

RunSent->Inf.AddWeight(RunSent->Prev->Inf.GetWeight());

if (RunSent->Inf.Get\_number\_first\_lex\_field\_mnemocode() == -1)

{

RunSent = RunSent->Next;

continue;

}

RunLex = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_field\_mnemocode());

if (RunLex->Inf.type\_lexIs("директиви даних"))

{

switch (RunLex->Inf.Get\_type\_lex()[strlen(RunLex->Inf.Get\_type\_lex())-1])

{

case '1':{

int k = 1;

int x = 1;

RunLex = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1());

if (RunLex->Inf.type\_lexIs("text константа"))

x = RunLex->Inf.Get\_len\_lex();

RunSent->Inf.AddWeight(k \* x);

break;

}

case '2':{

int k = 2;

int x = 1;

RunLex = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1());

if (RunLex->Inf.type\_lexIs("text константа"))

x = RunLex->Inf.Get\_len\_lex();

RunSent->Inf.AddWeight(k \* x);

break;

}

case '4':{

int k = 4;

int x = 1;

RunLex = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1());

if (RunLex->Inf.type\_lexIs("text константа"))

x = RunLex->Inf.Get\_len\_lex();

RunSent->Inf.AddWeight(k \* x);

break;

}

}

}

if (RunLex->Inf.type\_lexIs("мнемокод"))

{

if (RunLex->Inf.cmpLex("CLI"))

RunSent->Inf.AddWeight(1);

if (RunLex->Inf.cmpLex("JNA") || RunLex->Inf.cmpLex("OR"))

RunSent->Inf.AddWeight(2);

if ( RunLex->Inf.cmpLex("MOV"))

{

if (tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1())->Inf.type\_lexIs("8"))

{

int x = 0;

struct Dyn\_equ \*RunEqu;

RunEqu = tab\_sint::GetEQU();

for (int q = 0; q < tab->TabSentStruct->Inf.GetAmountEQU(); q++)

{ if (tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2())->Inf.cmpLex(RunEqu->Inf.name))

x = RunEqu->Inf.value;

RunEqu = RunEqu->Next;

}

if (x == 0)

x = atoi(tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2())->Inf.GetLex());

if (x <= 255)

RunSent->Inf.AddWeight(2);

else

RunSent->Inf.AddWeight(3);

}

else

RunSent->Inf.AddWeight(3);

}

if (RunLex->Inf.cmpLex("INC"))

{

RunLex2 = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1());

if (RunLex2->Inf.type\_lexIs("8"))

RunSent->Inf.AddWeight(2);

if (RunLex2->Inf.type\_lexIs("16"))

RunSent->Inf.AddWeight(1);

}

if (RunLex->Inf.cmpLex("DEC") || RunLex->Inf.cmpLex("AND") || RunLex->Inf.cmpLex("ADD"))

{

bool flag = true;

int k = 2;

RunLex2 = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1());

for (int j = 0; j < RunSent->Inf.Get\_amount\_lex\_operand1(); j++)

{

if (RunLex2->Inf.type\_lexIs("сегментного") )

{

if ((RunLex2->Inf.cmpLex("DS") && !RunLex2->Next->Next->Next->Next->Next->Inf.cmpLex("BP"))

|| (RunLex2->Inf.cmpLex("SS") && RunLex2->Next->Next->Next->Next->Next->Inf.cmpLex("BP")))

break;

else

{

RunSent->Inf.AddWeight(1);

break;

}

}

RunLex2 = RunLex2->Next;

}

RunLex2 = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1());

for (int j = 0; j < RunSent->Inf.Get\_amount\_lex\_operand1(); j++)

{

if (RunLex2->Inf.cmpLex("+"))

{

flag = false;

k++;

if (RunLex2->Next->Inf.type\_lexIs("константа") && atoi(RunLex2->Next->Inf.GetLex()) > 255)

k++;

break;

}

RunLex2 = RunLex2->Next;

}

if (flag == true)

if (RunLex->Inf.cmpLex("ADD"))

{

k++;

int x = atoi(tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2())->Inf.GetLex());

if (x <= 255)

RunSent->Inf.AddWeight(k);

else

{

k++;

RunSent->Inf.AddWeight(k);

}

}

else

RunSent->Inf.AddWeight(k);

else

if (RunLex->Inf.cmpLex("ADD"))

{

int x = 0;

struct Dyn\_equ \*RunEqu;

RunEqu = tab\_sint::GetEQU();

for (int q = 0; q < tab->TabSentStruct->Inf.GetAmountEQU(); q++)

{

if (tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2())->Inf.cmpLex(RunEqu->Inf.name))

x = RunEqu->Inf.value;

RunEqu = RunEqu->Next;

}

if (x == 0)

x = atoi(tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2())->Inf.GetLex());

k++;

if (x <= 255)

RunSent->Inf.AddWeight(k);

else

{

k++;

RunSent->Inf.AddWeight(k);

}

}

else

RunSent->Inf.AddWeight(k);

}

if (RunLex->Inf.cmpLex("CMP"))

{

bool flag = true;

RunLex2 = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2());

for (int j = 0; j < RunSent->Inf.Get\_amount\_lex\_operand2(); j++)

{

if (RunLex2->Inf.cmpLex("+") )

{

flag = false;

break;

}

RunLex2 = RunLex2->Next;

}

if (flag == true)

RunSent->Inf.AddWeight(2);

else

RunSent->Inf.AddWeight(3);

RunLex2 = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2());

for (int j = 0; j < RunSent->Inf.Get\_amount\_lex\_operand2(); j++)

{

if (RunLex2->Inf.type\_lexIs("сегментного"))

{

if ((RunLex2->Inf.cmpLex("DS") && !RunLex2->Next->Next->Next->Next->Next->Inf.cmpLex("BP"))

|| (RunLex2->Inf.cmpLex("SS") && RunLex2->Next->Next->Next->Next->Next->Inf.cmpLex("BP")))

break;

else

{

RunSent->Inf.AddWeight(1);

break;

}

}

RunLex2 = RunLex2->Next;

}

}

}

RunSent = RunSent->Next;

}

}

**segment.h**

#ifndef SEGMENT\_H

#define SEGMENT\_H

#include "main.h"

class tab\_segment{

public:

tab\_segment();

void SetName(char\*);

char\* GetName();

void SetLength(int);

int GetLength();

private:

char \*Name;

int Length;

char \*Align;

char \*CombineClass;

};

class dyn\_tab\_segment{

public:

dyn\_tab\_segment();

tab\_segment Inf;

dyn\_tab\_segment \*Next;

dyn\_tab\_segment \*Prev;

};

#endif

**segment.cpp**

#include "segment.h"

tab\_segment::tab\_segment()

{

Name = 0;

Length = 0x0;

Align = 0;

CombineClass = 0;

}

void tab\_segment::SetName(char\* str)

{

Name = new char[strlen(str)];

strcpy(Name, str);

}

char\* tab\_segment::GetName()

{

return Name;

}

void tab\_segment::SetLength(int x)

{

Length = x;

}

int tab\_segment::GetLength()

{

return Length;

}

dyn\_tab\_segment::dyn\_tab\_segment()

{

Next = 0;

Prev = 0;

}

void Segs(AllTab \*tab)

{

DynTab \*Run;

DynTab\_sint \*RunSent;

dyn\_tab\_segment \*RunSeg,\*RunSeg2;

bool flagData = true;

Run = tab->LexTab;

for (int i = 1; i < tab->number\_str\_lex; i++)

{

if (Run->Inf.cmpLex("SEGMENT"))

{

RunSeg = new dyn\_tab\_segment;

if (tab->SegmentTab == 0)

tab->SegmentTab = RunSeg;

else

{

RunSeg->Prev = RunSeg2;

RunSeg2->Next = RunSeg;

RunSeg->Next = tab->SegmentTab;

tab->SegmentTab->Prev=RunSeg;

}

RunSeg->Inf.SetName(Run->Prev->Inf.GetLex());

RunSent = tab->TabSentStruct;

if (flagData == true)

{

for (int j = 0; j < tab\_lex::GetEND("data"); j++)

{

RunSent = RunSent->Next;

}

flagData = false;

}

else

for (int j = 0; j < tab\_lex::GetEND("code"); j++)

{

RunSent = RunSent->Next;

}

RunSeg->Inf.SetLength(RunSent->Prev->Inf.GetWeight());

RunSeg2 = RunSeg;

}

Run = Run->Next;

}

}

**ident.h**

#ifndef IDENT\_H

#define IDENT\_H

#include "main.h"

class TabIdentificator{

public:

TabIdentificator();

void SetName(char\*);

char\* GetName();

bool CmpName(char\*);

void SetAttr(char\*);

char\* GetAttr();

void SetType(char\*);

char\* GetType();

void SetValue(int);

int GetValue();

static void IncAmount();

static int GetAmount();

private:

char\* Name;

char\* Type;

int Value;

char\* Attr;

static int amount;

};

class DynTabIdentificator{

public:

DynTabIdentificator();

class TabIdentificator Inf;

class DynTabIdentificator \*Next;

class DynTabIdentificator \*Prev;

};

#endif

**ident.cpp**

#include "ident.h"

int TabIdentificator::amount = 0;

TabIdentificator::TabIdentificator()

{

Name = 0;

Type = 0;

Attr = 0;

}

void TabIdentificator::SetName(char\* str)

{

Name = new char[strlen(str)];

strcpy(Name, str);

}

void TabIdentificator::IncAmount()

{

amount++;

}

int TabIdentificator::GetAmount()

{

return amount;

}

char\* TabIdentificator::GetName()

{

return Name;

}

bool TabIdentificator::CmpName(char\* str)

{

if (strcmp(Name, str) == 0)

return true;

return false;

}

void TabIdentificator::SetAttr(char\* str)

{

Attr = new char[strlen(str)];

strcpy(Attr, str);

}

char\* TabIdentificator::GetAttr()

{

return Attr;

}

void TabIdentificator::SetType(char\* str)

{

Type = new char[strlen(str)];

strcpy(Type, str);

}

char\* TabIdentificator::GetType()

{

return Type;

}

void TabIdentificator::SetValue(int x)

{

Value = x;

}

int TabIdentificator::GetValue()

{

return Value;

}

DynTabIdentificator::DynTabIdentificator()

{

Next = 0;

Prev = 0;

}

void MakeTabIdent(AllTab \*tab)

{

DynTab \*Run;

DynTab\_sint \*RunSent;

DynTabIdentificator \*RunID, \*RunID2;

bool flagID = false;

RunSent = tab->TabSentStruct;

for (int i = 1; i <= tab\_sint::Get\_number\_sent(); i++)

{

Run = tab->FindLex(RunSent->Inf.Get\_number\_lex\_field\_label());

if (Run->Inf.type\_lexIs("користувача"))

{

if (flagID == true && !tab->IsID(Run->Inf.GetLex()))

{

Run = Run->Next;

continue;

}

RunID = new DynTabIdentificator;

if (tab->TabIdentificator == 0)

{

tab->TabIdentificator = RunID;

RunID->Next = RunID;

RunID->Prev = RunID;

flagID = true;

}

else

{

RunID->Prev = RunID2;

RunID2->Next = RunID;

RunID->Next = tab->TabIdentificator;

tab->TabIdentificator->Prev = RunID;

}

RunID->Inf.SetName(Run->Inf.GetLex());

if (Run->Inf.Get\_number() < tab\_lex::GetIDEND("data"))

{

RunID->Inf.SetAttr(tab->FindLex(tab\_lex::GetIDEND("data") - 1)->Inf.GetLex());

if (Run->Inf.Get\_number() - 1 == tab\_lex::GetIDSTART("data"))

RunID->Inf.SetValue(0);

else

RunID->Inf.SetValue(RunSent->Prev->Inf.GetWeight());

}

else

if (Run->Inf.Get\_number() > tab\_lex::GetIDSTART("code"))

{

RunID->Inf.SetAttr(tab->FindLex(tab\_lex::GetIDEND("code") - 1)->Inf.GetLex());

if (Run->Inf.Get\_number() - 1 == tab\_lex::GetIDSTART("code"))

RunID->Inf.SetValue(0);

else

RunID->Inf.SetValue(RunSent->Prev->Inf.GetWeight());

}

RunID2 = RunID;

TabIdentificator::IncAmount();

}

RunSent = RunSent->Next;

}

struct Dyn\_equ \*RunEQU;

RunEQU = tab\_sint::GetEQU();

for (int i = 0; i < tab\_sint::GetAmountEQU(); i++)

{

RunID = new DynTabIdentificator;

if (tab->TabIdentificator == 0)

{

tab->TabIdentificator = RunID;

RunID->Next = RunID;

RunID->Prev = RunID;

flagID = true;

}

else

{

RunID->Prev = RunID2;

RunID2->Next = RunID;

RunID->Next = tab->TabIdentificator;

tab->TabIdentificator->Prev = RunID;

}

RunID->Inf.SetName(RunEQU->Inf.name);

RunID->Inf.SetAttr("");

RunID->Inf.SetValue(RunEQU->Inf.value);

RunEQU = RunEQU->Next;

RunID2 = RunID;

TabIdentificator::IncAmount();

}

}

**code.h**

#ifndef CODE\_H

#define CODE\_H

#include "main.h"

#include "error.h"

class DynTab\_sint;

class DynTab;

class AllTab;

class instruction{

public:

instruction();

static void SetCLI(DynTab\_sint \*);

static void SetINC(DynTab\_sint \*, DynTab\*);

static void SetDEC(DynTab\_sint \*, DynTab\*);

static void SetOR(DynTab\_sint \*, DynTab\*, DynTab\*);

static void SetCMP(DynTab\_sint \*, DynTab \*, DynTab\*);

static void SetAND(DynTab\_sint \*, DynTab \*, DynTab\*);

static void SetMOV(DynTab\_sint \*, DynTab \*, DynTab\*);

static void SetADD(DynTab\_sint \*, DynTab \*, DynTab\*);

static void SetJNA(DynTab\_sint \*,AllTab\*);

private:

static int ModR\_M(DynTab\_sint \*, DynTab\*,int,int);

static int instruction::Base(DynTab\*);

static int instruction::ChangeSeg(DynTab \*);

};

extern int degree(int x, int n, int result);

int makeDataCode(int val);

int BINtoDEC(char\* str);

int HEXtoDEC(char\*str);

int\* DECtoHEX(int\* outArr, char \*str);

#endif

**instruction.cpp**

#include "code.h"

int instruction::ModR\_M(DynTab\_sint \*RunSent, DynTab\* operand, int disp, int digit)

{

int ModR\_m;

if (operand->Inf.cmpLex("SI"))

{

ModR\_m = 0x44;

}

if (operand->Inf.cmpLex("DI"))

{

ModR\_m = 0x45;

}

if (operand->Inf.cmpLex("BP"))

{

ModR\_m = 0x46;

}

if (operand->Inf.cmpLex("BX"))

{

ModR\_m = 0x47;

}

if (disp == 8)

return ModR\_m + digit \* 0x8;

else

return ModR\_m + digit \* 0x8 + 0x40;

}

int instruction::Base(DynTab \*reg1)

{

if (reg1->Inf.cmpLex("AX") || reg1->Inf.cmpLex("AL"))

{

return 0x00;

}

if (reg1->Inf.cmpLex("CX") || reg1->Inf.cmpLex("CL"))

{

return 0x01;

}

if (reg1->Inf.cmpLex("DX") || reg1->Inf.cmpLex("DL"))

{

return 0x02;

}

if (reg1->Inf.cmpLex("BX") || reg1->Inf.cmpLex("BL"))

{

return 0x03;

}

if (reg1->Inf.cmpLex("SP") || reg1->Inf.cmpLex("AH"))

{

return 0x04;

}

if (reg1->Inf.cmpLex("BP") || reg1->Inf.cmpLex("CH"))

{

return 0x05;

}

if (reg1->Inf.cmpLex("SI") || reg1->Inf.cmpLex("DH"))

{

return 0x06;

}

if (reg1->Inf.cmpLex("DI") || reg1->Inf.cmpLex("BH"))

{

return 0x07;

}

return 0;

}

int instruction::ChangeSeg(DynTab \*operand)

{

if (operand->Inf.cmpLex("ES"))

return 0x26;

if (operand->Inf.cmpLex("CS"))

return 0x2E;

if (operand->Inf.cmpLex("SS"))

return 0x36;

if (operand->Inf.cmpLex("DS"))

return 0x3E;

/\* if (operand->Inf.cmpLex("FS"))

return 0x64;

if (operand->Inf.cmpLex("GS"))

return 0x65; \*/

return 0;

}

void instruction::SetCLI(DynTab\_sint \*RunSent)

{

RunSent->Inf.CreateCode(RunSent->Inf.GetWeight());

RunSent->Inf.SetCode(0xFA, 0);

}

void instruction::SetINC(DynTab\_sint \*RunSent, DynTab \*operand)

{

int weight = RunSent->Inf.GetWeight() - RunSent->Prev->Inf.GetWeight();

int Opcode;

RunSent->Inf.CreateCode(weight);

if (weight == 1)

{

Opcode = 0x40;

int base;

if (operand->Inf.cmpLex("AX"))

{

base = 0x0;

RunSent->Inf.SetCode(Opcode += base, 0);

return;

}

if (operand->Inf.cmpLex("CX"))

{

base = 0x1;

RunSent->Inf.SetCode(Opcode += base, 0);

return;

}

if (operand->Inf.cmpLex("DX"))

{

base = 0x2;

RunSent->Inf.SetCode(Opcode += base, 0);

return;

}

if (operand->Inf.cmpLex("BX"))

{

base = 0x3;

RunSent->Inf.SetCode(Opcode += base, 0);

return;

}

if (operand->Inf.cmpLex("SP"))

{

base = 0x4;

RunSent->Inf.SetCode(Opcode += base, 0);

return;

}

if (operand->Inf.cmpLex("BP"))

{

base = 0x5;

RunSent->Inf.SetCode(Opcode += base, 0);

return;

}

if (operand->Inf.cmpLex("SI"))

{

base = 0x6;

RunSent->Inf.SetCode(Opcode += base, 0);

return;

}

if (operand->Inf.cmpLex("DI"))

{

base = 0x7;

RunSent->Inf.SetCode(Opcode += base, 0);

return;

}

}

else

{

int reg;

Opcode = 0xFE;

RunSent->Inf.SetCode(Opcode, 0);

if (operand->Inf.cmpLex("AL"))

{

reg = 0xC0;

RunSent->Inf.SetCode(reg, 1);

return;

}

if (operand->Inf.cmpLex("CL"))

{

reg = 0xC1;

RunSent->Inf.SetCode(reg, 1);

return;

}

if (operand->Inf.cmpLex("DL"))

{

reg = 0xC2;

RunSent->Inf.SetCode(reg, 1);

return;

}

if (operand->Inf.cmpLex("BL"))

{

reg = 0xC3;

RunSent->Inf.SetCode(reg, 1);

return;

}

if (operand->Inf.cmpLex("AH"))

{

reg = 0xC4;

RunSent->Inf.SetCode(reg, 1);

return;

}

if (operand->Inf.cmpLex("CH"))

{

reg = 0xC5;

RunSent->Inf.SetCode(reg, 1);

return;

}

if (operand->Inf.cmpLex("DH"))

{

reg = 0xC6;

RunSent->Inf.SetCode(reg, 1);

return;

}

if (operand->Inf.cmpLex("BH"))

{

reg = 0xC7;

RunSent->Inf.SetCode(reg, 1);

return;

}

}

}

void instruction::SetDEC(DynTab\_sint \*RunSent, DynTab \*operand)

{

int weight = RunSent->Inf.GetWeight() - RunSent->Prev->Inf.GetWeight();

int Opcode = 0xFE;

int segment = 0;

int k = 0;

if (operand->Inf.type\_lexIs("сегментного"))

{

weight++;

k++;

if (!((operand->Inf.cmpLex("DS") && !operand->Next->Next->Next->Next->Next->Inf.cmpLex("BP"))

|| (operand->Inf.cmpLex("SS") && operand->Next->Next->Next->Next->Next->Inf.cmpLex("BP"))))

{

segment = ChangeSeg(operand);

}

else

segment--;

operand = operand->Next->Next;

}

operand = operand->Next->Next->Next;

int x1;

if (operand->Next->Next->Inf.type\_lexIs("Двійкова")) {

x1 = BINtoDEC(operand->Next->Next->Inf.GetLex());

}

if (operand->Next->Next->Inf.type\_lexIs("Шістнадцятирічна")) {

x1 = HEXtoDEC(operand->Next->Next->Inf.GetLex());

}

if (operand->Next->Next->Inf.type\_lexIs("Десяткова")) {

x1 = atoi(operand->Next->Next->Inf.GetLex());

}

if (x1 >= 256)

{

RunSent->Inf.CreateCode(weight - 1);

RunSent->Inf.SetCode(ModR\_M(RunSent, operand, 16, 1),k + 1);

}

else

{

RunSent->Inf.CreateCode(weight);

RunSent->Inf.SetCode(ModR\_M(RunSent, operand, 8, 1), k + 1);

}

if (segment != 0)

RunSent->Inf.SetCode(segment, 0);

RunSent->Inf.SetCode(Opcode, k);

RunSent->Inf.SetCode(x1, k + 2);

}

void instruction::SetOR(DynTab\_sint \*RunSent, DynTab\* reg1, DynTab\*reg2)

{

int weight = RunSent->Inf.GetWeight() - RunSent->Prev->Inf.GetWeight();

int Opcode;

int base;

int k = 1;

RunSent->Inf.CreateCode(weight);

if (reg1->Inf.type\_lexIs("8"))

{

Opcode = 0x0A;

}

else

Opcode = 0x0B;

RunSent->Inf.SetCode(Opcode, 0);

base = Base(reg1);

if (reg2->Inf.cmpLex("AX") || reg2->Inf.cmpLex("AL"))

{

RunSent->Inf.SetCode( 0xC0 + base\*8, 1);

}

if (reg2->Inf.cmpLex("CX") || reg2->Inf.cmpLex("CL"))

{

RunSent->Inf.SetCode( 0xC1 + base \* 8, 1);

}

if (reg2->Inf.cmpLex("DX") || reg2->Inf.cmpLex("DL"))

{

RunSent->Inf.SetCode(0xC2 + base \* 8, 1);

}

if (reg2->Inf.cmpLex("BX") || reg2->Inf.cmpLex("BL"))

{

RunSent->Inf.SetCode( 0xC3 + base \* 8, 1);

}

if (reg2->Inf.cmpLex("SP") || reg2->Inf.cmpLex("AH"))

{

RunSent->Inf.SetCode( 0xC4 + base \* 8, 1);

}

if (reg2->Inf.cmpLex("BP") || reg2->Inf.cmpLex("CH"))

{

RunSent->Inf.SetCode( 0xC5 + base \* 8, 1);

}

if (reg2->Inf.cmpLex("SI") || reg2->Inf.cmpLex("DH"))

{

RunSent->Inf.SetCode( 0xC6 + base \* 8, 1);

}

if (reg2->Inf.cmpLex("DI") || reg2->Inf.cmpLex("BH"))

{

RunSent->Inf.SetCode( 0xC7 + base \* 8, 1);

}

}

void instruction::SetCMP(DynTab\_sint \*RunSent, DynTab \*operand, DynTab\* reg)

{

int weight = RunSent->Inf.GetWeight() - RunSent->Prev->Inf.GetWeight();

int Opcode;

int segment = 0;

int k = 0;

int base;

if (reg->Inf.type\_lexIs("8"))

{

Opcode = 0x3A;

}

else

Opcode = 0x3B;

base = Base(reg);

if (operand->Inf.type\_lexIs("сегментного"))

{

weight++;

k++;

if (!((operand->Inf.cmpLex("DS") && !operand->Next->Next->Next->Next->Next->Inf.cmpLex("BP"))

|| (operand->Inf.cmpLex("SS") && operand->Next->Next->Next->Next->Next->Inf.cmpLex("BP"))))

{

segment = ChangeSeg(operand);

}

else

segment--;

operand = operand->Next->Next;

}

operand = operand->Next->Next->Next;

int x1;

if (operand->Next->Next->Inf.type\_lexIs("Двійкова")) {

x1 = BINtoDEC(operand->Next->Next->Inf.GetLex());

}

if (operand->Next->Next->Inf.type\_lexIs("Шістнадцятирічна")) {

x1 = HEXtoDEC(operand->Next->Next->Inf.GetLex());

}

if (operand->Next->Next->Inf.type\_lexIs("Десяткова")) {

x1 = atoi(operand->Next->Next->Inf.GetLex());

}

if (x1 >= 256)

{

RunSent->Inf.CreateCode(weight - 1);

RunSent->Inf.SetCode(ModR\_M(RunSent, operand, 16, base), k + 1);

}

else

{

RunSent->Inf.CreateCode(weight);

RunSent->Inf.SetCode(ModR\_M(RunSent, operand, 8, base), k + 1);

}

if (segment != 0)

RunSent->Inf.SetCode(segment, 0);

RunSent->Inf.SetCode(Opcode, k);

RunSent->Inf.SetCode(x1, k + 2);

}

void instruction::SetAND(DynTab\_sint \*RunSent, DynTab \*operand, DynTab\* reg)

{

int weight = RunSent->Inf.GetWeight() - RunSent->Prev->Inf.GetWeight();

int Opcode;

int segment = 0;

int k = 0;

int base;

if (reg->Inf.type\_lexIs("8"))

{

Opcode = 0x20;

}

else

Opcode = 0x21;

base = Base(reg);

if (operand->Inf.type\_lexIs("сегментного"))

{

weight++;

k++;

if (!((operand->Inf.cmpLex("DS") && !operand->Next->Next->Next->Next->Next->Inf.cmpLex("BP"))

|| (operand->Inf.cmpLex("SS") && operand->Next->Next->Next->Next->Next->Inf.cmpLex("BP"))))

{

segment = ChangeSeg(operand);

}

else

segment--;

operand = operand->Next->Next;

}

operand = operand->Next->Next->Next;

int x1;

if (operand->Next->Next->Inf.type\_lexIs("Двійкова")) {

x1 = BINtoDEC(operand->Next->Next->Inf.GetLex());

}

if (operand->Next->Next->Inf.type\_lexIs("Шістнадцятирічна")) {

x1 = HEXtoDEC(operand->Next->Next->Inf.GetLex());

}

if (operand->Next->Next->Inf.type\_lexIs("Десяткова")) {

x1 = atoi(operand->Next->Next->Inf.GetLex());

}

if (x1 >= 256)

{

RunSent->Inf.CreateCode(weight - 1);

RunSent->Inf.SetCode(ModR\_M(RunSent, operand, 16, base), k + 1);

}

else

{

RunSent->Inf.CreateCode(weight);

RunSent->Inf.SetCode(ModR\_M(RunSent, operand, 8, base), k + 1);

}

if (segment != 0)

RunSent->Inf.SetCode(segment, 0);

RunSent->Inf.SetCode(Opcode, k);

RunSent->Inf.SetCode(x1, k + 2);

}

void instruction::SetMOV(DynTab\_sint \*RunSent, DynTab\* reg, DynTab \*imm)

{

int weight = RunSent->Inf.GetWeight() - RunSent->Prev->Inf.GetWeight();

int Opcode;

int k = 0;

int x;

bool equFlag = false;

int base;

RunSent->Inf.CreateCode(weight);

if (reg->Inf.type\_lexIs("8"))

{

Opcode = 0xB0;

}

else

Opcode = 0xB8;

base = Base(reg);

RunSent->Inf.SetCode(Opcode + base, k);

struct Dyn\_equ \*RunEqu;

RunEqu = tab\_sint::GetEQU();

for (int q = 0; q < tab\_sint::GetAmountEQU(); q++)

{

if (imm->Inf.cmpLex(RunEqu->Inf.name))

{

x = RunEqu->Inf.value;

equFlag = true;

}

RunEqu = RunEqu->Next;

}

if (equFlag == true)

RunSent->Inf.SetCode(x, k + 1);

else

{

int x;

if (imm->Inf.type\_lexIs("Двійкова")) {

x = BINtoDEC(imm->Inf.GetLex());

}

if (imm->Inf.type\_lexIs("Шістнадцятирічна")) {

x = HEXtoDEC(imm->Inf.GetLex());

}

if (imm->Inf.type\_lexIs("Десяткова")) {

x = atoi(imm->Inf.GetLex());

}

RunSent->Inf.SetCode(x, k + 1);

}

}

void instruction::SetADD(DynTab\_sint \*RunSent, DynTab \*operand, DynTab\*imm)

{

int weight = RunSent->Inf.GetWeight() - RunSent->Prev->Inf.GetWeight();

int Opcode;

int segment = 0;

int k = 0;

int x;

bool equFlag = false;

if (operand->Inf.type\_lexIs("сегментного"))

{

weight++;

k++;

if (!((operand->Inf.cmpLex("DS") && !operand->Next->Next->Next->Next->Next->Inf.cmpLex("BP"))

|| (operand->Inf.cmpLex("SS") && operand->Next->Next->Next->Next->Next->Inf.cmpLex("BP"))))

{

segment = ChangeSeg(operand);

}

else

segment--;

operand = operand->Next->Next;

}

operand = operand->Next->Next->Next;

int x1;

if (operand->Next->Next->Inf.type\_lexIs("Двійкова")) {

x1 = BINtoDEC(operand->Next->Next->Inf.GetLex());

}

if (operand->Next->Next->Inf.type\_lexIs("Шістнадцятирічна")) {

x1 = HEXtoDEC(operand->Next->Next->Inf.GetLex());

}

if (operand->Next->Next->Inf.type\_lexIs("Десяткова")) {

x1 = atoi(operand->Next->Next->Inf.GetLex());

}

if (x1 >= 256)

{

RunSent->Inf.CreateCode(weight - 1);

RunSent->Inf.SetCode(ModR\_M(RunSent, operand, 16, 0), k + 1);

}

else

{

RunSent->Inf.CreateCode(weight);

RunSent->Inf.SetCode(ModR\_M(RunSent, operand, 8, 0), k + 1);

}

if (segment != 0)

RunSent->Inf.SetCode(segment, 0);

RunSent->Inf.SetCode(x1, k + 2);

struct Dyn\_equ \*RunEqu;

RunEqu = tab\_sint::GetEQU();

for (int q = 0; q < tab\_sint::GetAmountEQU(); q++)

{

if (imm->Inf.cmpLex(RunEqu->Inf.name))

{

x = RunEqu->Inf.value;

equFlag = true;

}

RunEqu = RunEqu->Next;

}

if (equFlag == true)

{

if (x >= 256)

{

Opcode = 0x81;

}

else

Opcode = 0x83;

RunSent->Inf.SetCode(x, k + 3);

}

else

{

int x;

if (imm->Inf.type\_lexIs("Двійкова")) {

x = BINtoDEC(imm->Inf.GetLex());

}

if (imm->Inf.type\_lexIs("Шістнадцятирічна")) {

x = HEXtoDEC(imm->Inf.GetLex());

}

if (imm->Inf.type\_lexIs("Десяткова")) {

x = atoi(imm->Inf.GetLex());

}

if (x >= 256)

{

Opcode = 0x81;

}

else

Opcode = 0x83;

RunSent->Inf.SetCode(x, k + 3);

}

RunSent->Inf.SetCode(Opcode, k);

}

void instruction::SetJNA(DynTab\_sint \*RunSent, AllTab\* tab)

{

int weight = RunSent->Inf.GetWeight() - RunSent->Prev->Inf.GetWeight();

int Opcode = 0x76;

bool flag = false;

DynTabIdentificator \*RunID;

DynTab \*Run;

RunSent->Inf.CreateCode(weight);

RunID = tab->TabIdentificator;

Run = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1());

for (int i = 0; i < TabIdentificator::GetAmount(); i++)

{

if (Run->Inf.cmpLex(RunID->Inf.GetName()))

{

flag = true;

break;

}

RunID = RunID->Next;

}

if (flag == false)

error::NoLebel(Run->Inf.GetLex());

RunSent->Inf.SetCode(Opcode, 0);

if (RunID->Inf.GetValue() > RunSent->Inf.GetWeight())

RunSent->Inf.SetCode(RunID->Inf.GetValue() - RunSent->Inf.GetWeight(), 1);

else

RunSent->Inf.SetCode(0xFF - RunSent->Inf.GetWeight() + RunID->Inf.GetValue() +0x1, 1);

}

**code.cpp**

#include "Code.h"

using namespace std;

void code(AllTab\* tab)

{

DynTab\_sint \*RunSent;

DynTab \*RunLex;

bool flagEnd = true;

RunSent = tab->TabSentStruct;

for (int i = 1; i <= tab\_sint::Get\_number\_sent(); i++)

{

if (flagEnd == true && i == tab\_lex::GetEND("data"))

{

flagEnd = false;

}

if (flagEnd == true)

{

RunLex = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1());

if (RunLex->Inf.type\_lexIs("Двійкова") == true)

{

int arr[9];

RunSent->Inf.CreateCode(RunSent->Inf.GetWeight());

if (RunLex->Prev->Inf.type\_lexIs("1") == true)

RunSent->Inf.SetCodeData(BINtoDEC(RunLex->Inf.GetLex()));

if (RunLex->Prev->Inf.type\_lexIs("2") == true)

RunSent->Inf.SetCodeData(BINtoDEC(RunLex->Inf.GetLex()));

if (RunLex->Prev->Inf.type\_lexIs("4") == true)

{

char \*str =new char[9];

itoa(BINtoDEC(RunLex->Inf.GetLex()), str, 10);

RunSent->Inf.SetCodeData(DECtoHEX(arr,str));

delete[] str;

}

}

if (RunLex->Inf.type\_lexIs("Десяткова") == true)

{

int arr[9];

RunSent->Inf.CreateCode(RunSent->Inf.GetWeight());

if (RunLex->Prev->Inf.type\_lexIs("1") == true)

RunSent->Inf.SetCodeData(atoi(RunLex->Inf.GetLex()));

if (RunLex->Prev->Inf.type\_lexIs("2") == true)

RunSent->Inf.SetCodeData(atoi(RunLex->Inf.GetLex()));

if (RunLex->Prev->Inf.type\_lexIs("4") == true)

RunSent->Inf.SetCodeData(DECtoHEX(arr,RunLex->Inf.GetLex()));

}

if (RunLex->Inf.type\_lexIs("Шістнадцятирічна") == true)

{

int arr[9];

RunSent->Inf.CreateCode(RunSent->Inf.GetWeight()+1);

if (RunLex->Prev->Inf.type\_lexIs("1") == true)

RunSent->Inf.SetCodeData(HEXtoDEC(RunLex->Inf.GetLex()));

if (RunLex->Prev->Inf.type\_lexIs("2") == true)

RunSent->Inf.SetCodeData(HEXtoDEC(RunLex->Inf.GetLex()));

if (RunLex->Prev->Inf.type\_lexIs("4") == true)

{

char str1[80];

itoa(HEXtoDEC(RunLex->Inf.GetLex()), str1, 10);

RunSent->Inf.SetCodeData(DECtoHEX(arr, str1));

}

}

if (RunLex->Inf.type\_lexIs("text") == true)

{

if (i > 1)

RunSent->Inf.CreateCode(RunSent->Inf.GetWeight()-RunSent->Prev->Inf.GetWeight());

else

RunSent->Inf.CreateCode(RunSent->Inf.GetWeight());

int n = RunLex->Inf.Get\_len\_lex();

char \*str = new char[n + 1];

strcpy(str, RunLex->Inf.GetLex());

for (int j = 0; j < n; j++)

RunSent->Inf.SetCodeTXT((static\_cast<int>(str[j]) < 0 ? 256 + static\_cast<int>(str[j]) : static\_cast<int>(str[j])), j);

delete[] str;

}

}

else

{

if (RunSent->Inf.Get\_number\_first\_lex\_field\_mnemocode() != -1)

{

RunLex = tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_field\_mnemocode());

if (RunLex->Inf.cmpLex("CLI"))

{

instruction::SetCLI(RunSent);

}

if (RunLex->Inf.cmpLex("INC"))

{

instruction::SetINC(RunSent, tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1()));

}

if (RunLex->Inf.cmpLex("DEC"))

{

instruction::SetDEC(RunSent, tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1()));

}

if (RunLex->Inf.cmpLex("OR"))

{

instruction::SetOR(RunSent, tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1()), tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2()));

}

if (RunLex->Inf.cmpLex("CMP"))

{

instruction::SetCMP(RunSent, tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2()), tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1()));

}

if (RunLex->Inf.cmpLex("AND"))

{

instruction::SetAND(RunSent, tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1()), tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2()));

}

if (RunLex->Inf.cmpLex("MOV"))

{

instruction::SetMOV(RunSent, tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1()), tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2()));

}

if (RunLex->Inf.cmpLex("ADD"))

{

instruction::SetADD(RunSent, tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1()), tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2()));

}

if (RunLex->Inf.cmpLex("JNA"))

{

instruction::SetJNA(RunSent, tab);

}

}

}

RunSent = RunSent->Next;

}

}

int BINtoDEC(char\* str)

{

char \*bin;

int dec = 0, n = strlen(str);

bin = new char[n + 1];

strcpy(bin, str);

for (int i = n-1; i >= 0; i--)

{

dec += atoi(&bin[i]) \* degree(2, n-i-1,1);

bin[i] = '\0';

}

return dec;

}

int HEXtoDEC(char\*str)

{

char \*hex;

int dec = 0, n = strlen(str);

hex = new char[n + 1];

strcpy(hex, str);

// hex[n-1] = '\0';

//n--;

for (int i = 0; i < n; i++)

switch (str[i])

{

case '0':{

dec += 0;

break;

}

case '1': {

dec += 1 \* degree(16, n - i-1, 1);

break;

}

case '2':{

dec += 2 \* degree(16, n - i-1, 1);

break;

}

case '3':{

dec += 3 \* degree(16, n - i-1, 1);

break;

}

case '4':{

dec += 4 \* degree(16, n - i-1, 1);

break;

}

case '5':{

dec += 5 \* degree(16, n - i-1, 1);

break;

}

case '6':{

dec += 6 \* degree(16, n - i-1, 1);

break;

}

case '7':{

dec += 7 \* degree(16, n - i-1, 1);

break;

}

case '8':{

dec += 8 \* degree(16, n - i-1, 1);

break;

}

case '9':{

dec += 9 \* degree(16, n - i-1, 1);

break;

}

case 'A':{

dec += 10 \* degree(16, n - i-1, 1);

break;

}

case 'B':{

dec += 11 \* degree(16, n - i-1, 1);

break;

}

case 'C':{

dec += 12 \* degree(16, n - i-1, 1);

break;

}

case 'D':{

dec += 13 \* degree(16, n - i-1, 1);

break;

}

case 'E':{

dec += 14 \* degree(16, n - i-1, 1);

break;

}

case 'F':{

dec += 15 \* degree(16, n - i-1, 1);

break;

}

}

return dec;

}

int\* DECtoHEX(int\* outArr,char \*str)

{

int div = 0,mod = 0,x;

int Arr[8];

int nArr=0,i =7;

x = atoi(str);

while (x >= 16)

{

mod = x % 256;

div = x / 256;

Arr[i] = mod;

nArr++;

i--;

x = div;

}

if (div != 0)

{

Arr[i] = div;

nArr++;

}

x = 0;

outArr[0] = nArr;

for (i = 8 -1; i>=8-nArr ; i--)

outArr[8-i] = Arr[i];

return outArr;

}

**error.h**

#ifndef ERROR\_H

#define ERROR\_H

#include "main.h"

#include "Code.h"

class AllTab;

class DynTab\_sint;

class error{

public:

static void overflow(int);

static void Illegal\_characters(int);

static void Invalid\_start\_line(int);

static void Invalid\_mnemonic(int);

static void Extra\_characters(int);

static void expected(int, char\*);

static void Invalid\_const(int );

static void NoLebel(char \*);

static void constant(int,char\*,int);

static void memory(AllTab\*, DynTab\_sint \*, int,int);

static void operands(AllTab\*,DynTab\_sint \*,int);

static void Exit();

static bool lex\_error\_flag;

};

#endif

**error.cpp**

#include "error.h"

using namespace std;

bool error::lex\_error\_flag = false;

void error::overflow(int number\_str)

{

cout << "ERROR IN LINE(" << number\_str << ") : more than 120 characters" << endl;

Exit();

}

void error::Illegal\_characters(int number\_str)

{

cout << "ERROR IN LINE(" << number\_str << ") : Illegal characters" << endl;

Exit();

}

void error::Invalid\_start\_line(int number\_str)

{

cout << "SINTACSIC ERROR IN LINE(" << number\_str << ")" << endl;

Exit();

}

void error::Invalid\_mnemonic(int number\_str)

{

cout << "ERROR IN LINE(" << number\_str << ") : Invalid\_mnemonic" << endl;

Exit();

}

void error::Extra\_characters(int number\_str)

{

cout << "ERROR IN LINE(" << number\_str << ") : Extra characters on line" << endl;

Exit();

}

void error::expected(int number\_str, char\* str)

{

cout << "ERROR IN LINE(" << number\_str << ") : Expected " << str << endl;

Exit();

}

void error::Invalid\_const(int number\_str)

{

cout << "ERROR IN LINE(" << number\_str << ") : const overflow " << endl;

Exit();

}

void error::NoLebel(char \*str)

{

cout << "ERROR : Symbol not defined " << str << endl;

Exit();

}

void error::constant(int number\_str,char\* str,int z)

{

double zz;

int x,n = strlen(str);

switch (str[n - 1])

{

case 'B': {

str[n - 1] = '\0';

x = BINtoDEC(str);

break;

}

case 'H': {

str[n - 1] = '\0';

x = HEXtoDEC(str);

break;

}

default: { x = atoi(str);

break;

}

}

if (z == 1)

zz = 65535;

else

zz = 4294967295;

if (x > zz)

Invalid\_const(number\_str);

}

void error::Exit()

{

system("pause");

exit(0);

}

void error::operands(AllTab\* tab, DynTab\_sint \*Run, int number\_str)

{

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_field\_mnemocode())->Inf.cmpLex("CLI"))

{

if (Run->Inf.Get\_amount\_lex\_operand1() != -1)

Extra\_characters(number\_str);

}

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_field\_mnemocode())->Inf.cmpLex("INC"))

{

if (Run->Inf.Get\_amount\_lex\_operand2() != -1)

Extra\_characters(number\_str);

if (Run->Inf.Get\_amount\_lex\_operand1() > 1)

expected(number\_str, "reg");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1())->Inf.type\_lexIs("регістр") == false)

expected(number\_str,"reg");

}

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_field\_mnemocode())->Inf.cmpLex("DEC"))

{

if (Run->Inf.Get\_amount\_lex\_operand2() != -1)

Extra\_characters(number\_str);

memory(tab, Run, number\_str,1);

}

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_field\_mnemocode())->Inf.cmpLex("OR"))

{

if (Run->Inf.Get\_amount\_lex\_operand1() > 1)

expected(number\_str, "reg");

if (Run->Inf.Get\_amount\_lex\_operand2() > 1)

expected(number\_str, "reg");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1())->Inf.type\_lexIs("регістр") == false)

expected(number\_str, "reg");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.type\_lexIs("регістр") == false)

expected(number\_str, "reg");

}

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_field\_mnemocode())->Inf.cmpLex("CMP"))

{

if (Run->Inf.Get\_amount\_lex\_operand1() > 1)

expected(number\_str, "reg");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1())->Inf.type\_lexIs("регістр") == false)

expected(number\_str, "reg");

memory(tab, Run, number\_str,2);

}

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_field\_mnemocode())->Inf.cmpLex("AND"))

{

memory(tab, Run, number\_str, 1);

if (Run->Inf.Get\_amount\_lex\_operand2() > 1)

expected(number\_str, "reg");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.type\_lexIs("регістр") == false)

expected(number\_str, "reg");

}

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_field\_mnemocode())->Inf.cmpLex("MOV"))

{

if (Run->Inf.Get\_amount\_lex\_operand1() > 1)

expected(number\_str, "reg");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1())->Inf.type\_lexIs("регістр") == false)

expected(number\_str, "reg");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.type\_lexIs("константа") == false &&

tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.type\_lexIs("користувача") == false)

expected(number\_str, "imm");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.type\_lexIs("користувача") == true)

{

struct Dyn\_equ \*RunEqu;

RunEqu = tab\_sint::GetEQU();

bool flagEqu = false;

for (int q = 0; q < tab->TabSentStruct->Inf.GetAmountEQU(); q++)

{

if (tab->FindLex(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.cmpLex(RunEqu->Inf.name)))

flagEqu = true;

RunEqu = RunEqu->Next;

}

if (flagEqu == false)

expected(number\_str, "imm");

}

}

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_field\_mnemocode())->Inf.cmpLex("ADD"))

{

memory(tab, Run, number\_str, 1);

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.type\_lexIs("константа") == false &&

tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.type\_lexIs("користувача") == false)

expected(number\_str, "imm");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.type\_lexIs("користувача") == true)

{

struct Dyn\_equ \*RunEqu;

RunEqu = tab\_sint::GetEQU();

bool flagEqu = false;

for (int q = 0; q < tab->TabSentStruct->Inf.GetAmountEQU(); q++)

{

if (tab->FindLex(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.cmpLex(RunEqu->Inf.name)))

flagEqu = true;

RunEqu = RunEqu->Next;

}

if (flagEqu == false)

expected(number\_str, "imm");

}

}

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_field\_mnemocode())->Inf.cmpLex("JNA"))

{

if (Run->Inf.Get\_amount\_lex\_operand2() != -1)

Extra\_characters(number\_str);

if (Run->Inf.Get\_amount\_lex\_operand1() >1)

Extra\_characters(number\_str);

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1())->Inf.type\_lexIs("користувача") == false)

expected(number\_str, "label");

}

}

void error::memory(AllTab\* tab, DynTab\_sint \*Run, int number\_str,int operand)

{

if (operand == 1)

{

if (Run->Inf.Get\_amount\_lex\_operand1() == 7)

{

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1())->Inf.type\_lexIs("розміру") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 1)->Inf.cmpLex("PTR") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 2)->Inf.cmpLex("[") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 3)->Inf.type\_lexIs("розрядного регістра") == true)

{

if (!(strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 3)->Inf.GetLex(), "SI")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 3)->Inf.GetLex(), "DI")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 3)->Inf.GetLex(), "BP")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 3)->Inf.GetLex(), "BX")))

expected(number\_str, "mem");

}

else

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 4)->Inf.cmpLex("+") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 5)->Inf.type\_lexIs("константа") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 6)->Inf.cmpLex("]") == false)

expected(number\_str, "mem");

}

else if (Run->Inf.Get\_amount\_lex\_operand1() == 9)

{

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1())->Inf.type\_lexIs("сегментного") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 1)->Inf.cmpLex(":") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 2)->Inf.type\_lexIs("розміру") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 3)->Inf.cmpLex("PTR") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 4)->Inf.cmpLex("[") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 5)->Inf.type\_lexIs("розрядного регістра") == true)

{

if (!(strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 5)->Inf.GetLex(), "SI")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 5)->Inf.GetLex(), "DI")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 5)->Inf.GetLex(), "BP")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 5)->Inf.GetLex(), "BX")))

expected(number\_str, "mem");

}

else

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 6)->Inf.cmpLex("+") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 7)->Inf.type\_lexIs("константа") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand1() + 8)->Inf.cmpLex("]") == false)

expected(number\_str, "mem");

}

else

expected(number\_str, "mem");

}

else

{

if (Run->Inf.Get\_amount\_lex\_operand2() == 7)

{

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.type\_lexIs("розміру") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 1)->Inf.cmpLex("PTR") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 2)->Inf.cmpLex("[") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 3)->Inf.type\_lexIs("розрядного регістра") == true)

{

if (!(strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 3)->Inf.GetLex(), "SI")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 3)->Inf.GetLex(), "DI")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 3)->Inf.GetLex(), "BP")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 3)->Inf.GetLex(), "BX")))

expected(number\_str, "mem");

}

else

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 4)->Inf.cmpLex("+") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 5)->Inf.type\_lexIs("константа") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 6)->Inf.cmpLex("]") == false)

expected(number\_str, "mem");

}

else if (Run->Inf.Get\_amount\_lex\_operand2() == 9)

{

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2())->Inf.type\_lexIs("сегментного") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 1)->Inf.cmpLex(":") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 2)->Inf.type\_lexIs("розміру") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 3)->Inf.cmpLex("PTR") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 4)->Inf.cmpLex("[") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 5)->Inf.type\_lexIs("розрядного регістра") == true)

{

if (!(strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 5)->Inf.GetLex(), "SI")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 5)->Inf.GetLex(), "DI")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 5)->Inf.GetLex(), "BP")

|| strcmp(tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 5)->Inf.GetLex(), "BX")))

expected(number\_str, "mem");

}

else

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 6)->Inf.cmpLex("+") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 7)->Inf.type\_lexIs("константа") == false)

expected(number\_str, "mem");

if (tab->FindLex(Run->Inf.Get\_number\_first\_lex\_operand2() + 8)->Inf.cmpLex("]") == false)

expected(number\_str, "mem");

}

else

expected(number\_str, "mem");

}

}

**makeLST.cpp**

#include <iostream>

#include "main.h"

#include <fstream>

#include <cstdlib>

#include <iomanip>

#include <time.h>

using std::cout;

using std::cin;

using std::ios;

using std::cerr;

using std::endl;

using std::ofstream;

using std::setw;

void writeListing1(AllTab \*tab)

{

DynTab\_sint \*RunSent;

DynTab \*RunLex;

int j,i = 0;

bool flagData = true;

bool flagCodeStart = true;

bool flagEQU = true;

ofstream outClientFile("test1.lst", ios::out);

if (!outClientFile)

{

cerr << "Файл не мог быть открыт" << endl;

exit(1);

}

time\_t lt;

lt = time(NULL);

outClientFile << "Ivanov Alexander (R) " << ctime(&lt) <<"\n\n" << endl;

RunSent = tab->TabSentStruct;

RunLex = tab->LexTab;

for (j = 1; j <= tab\_sint::Get\_number\_sent(); j++)

{

if (j == tab\_lex::GetSTART("data"))

{

int n;

if (RunSent->Inf.Get\_number\_lex\_field\_label() > -1)

n = RunSent->Inf.Get\_number\_lex\_field\_label();

else

n = RunSent->Inf.Get\_number\_first\_lex\_field\_mnemocode();

outClientFile << "0000" << " ";

outClientFile << tab->FindLex(tab\_lex::GetIDSTART("data") - 1)->Inf.GetLex() << " "

<< tab->FindLex(tab\_lex::GetIDSTART("data"))->Inf.GetLex() << endl;;

}

if (j == tab\_lex::GetEND("data"))

{

outClientFile.unsetf(ios::dec);

outClientFile.setf(ios::hex);

outClientFile.setf(ios::uppercase);

outClientFile << (RunSent->Prev->Inf.GetWeight() >= 16 ? "00" : "000") << RunSent->Prev->Inf.GetWeight() << " ";

outClientFile.setf(ios::dec);

outClientFile << tab->FindLex(tab\_lex::GetIDEND("data") - 1)->Inf.GetLex() << " "

<< tab->FindLex(tab\_lex::GetIDEND("data"))->Inf.GetLex() << '\n' << endl;;

flagData = false;

}

if (flagEQU == true && flagData == false && tab\_sint::GetAmountEQU() > 0)

{

flagEQU = false;

struct Dyn\_equ \*Run;

Run = tab\_sint::GetEQU();

for (i = 0; i < tab\_sint::GetAmountEQU(); i++)

{

outClientFile.unsetf(ios::dec);

outClientFile.setf(ios::hex);

outClientFile.setf(ios::uppercase);

if (Run->Inf.value >= 16)

if (Run->Inf.value >= 256)

if (Run->Inf.value >= 4096)

outClientFile <<"= "<< Run->Inf.value << " ";

else

outClientFile <<"= 0"<< Run->Inf.value << " ";

else

outClientFile << "= 00" << Run->Inf.value << " ";

else

outClientFile<< "= 000"<< Run->Inf.value << " ";

outClientFile.setf(ios::dec);

outClientFile << Run->Inf.name << " " << "EQU" << " " << Run->Inf.value << endl;

Run = Run->Next;

}

outClientFile << endl;

}

if (j == tab\_lex::GetSTART("code"))

{

outClientFile << "0000" << " ";

outClientFile << tab->FindLex(tab\_lex::GetIDSTART("code") - 1)->Inf.GetLex() << " "

<< tab->FindLex(tab\_lex::GetIDSTART("code"))->Inf.GetLex() << endl;

flagCodeStart = false;

outClientFile << "0000" << " ";

}

else

{

outClientFile.unsetf(ios::dec);

outClientFile.setf(ios::hex);

outClientFile.setf(ios::uppercase);

if (RunSent == tab->TabSentStruct)

{

outClientFile << "0000" << " ";

flagCodeStart = true;

}

else

outClientFile << (RunSent->Prev->Inf.GetWeight() >= 16 ? "00" : "000") << RunSent->Prev->Inf.GetWeight() << " ";

outClientFile.setf(ios::dec);

}

outClientFile.unsetf(ios::dec);

outClientFile.setf(ios::hex);

outClientFile.setf(ios::uppercase);

bool f = false;

if (tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1())->Inf.type\_lexIs("сегментного")

|| tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2())->Inf.type\_lexIs("сегментного"))

f = true;

for (int q = 0; q < RunSent->Inf.Get\_nCode(); q++)

{

if (flagData == true)

if (tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_field\_mnemocode())->Inf.cmpLex("DD"))

outClientFile << RunSent->Inf.GetCode(q+1) ;

else

outClientFile << RunSent->Inf.GetCode(q) << " ";

else

if (f == true)

{

f = false;

if (RunSent->Inf.GetCode(q) != -1)

outClientFile << RunSent->Inf.GetCode(q) << ": ";

}

else

if (RunSent->Inf.GetCode(q)>=16)

if (RunSent->Inf.GetCode(q) >= 256)

outClientFile << "0" << RunSent->Inf.GetCode(q) << " ";

else

outClientFile << RunSent->Inf.GetCode(q) << " ";

else

outClientFile << "0" << RunSent->Inf.GetCode(q) << " ";

}

outClientFile << " ";

outClientFile.setf(ios::dec);

outClientFile << (RunSent->Inf.Get\_number\_lex\_field\_label() != -1 ?

tab->FindLex(RunSent->Inf.Get\_number\_lex\_field\_label())->Inf.GetLex() : "") << " ";

outClientFile << (RunSent->Inf.Get\_number\_first\_lex\_field\_mnemocode() != -1 ?

tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_field\_mnemocode())->Inf.GetLex() : "") << " ";

if (RunSent->Inf.Get\_amount\_lex\_field\_mnemocode() > 1)

{

for (i = 1; i < RunSent->Inf.Get\_amount\_lex\_field\_mnemocode(); i++)

{

outClientFile << tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_field\_mnemocode() + i)->Inf.GetLex() << ' ';

}

}

outClientFile << (RunSent->Inf.Get\_number\_first\_lex\_operand1() != -1 ?

tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1())->Inf.GetLex() : "");

if (RunSent->Inf.Get\_amount\_lex\_operand1() > 1)

{

for (i = 1; i < RunSent->Inf.Get\_amount\_lex\_operand1(); i++)

{

if (i < RunSent->Inf.Get\_amount\_lex\_operand1() - 1 &&

(tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1() + i)->Inf.cmpLex("PTR")))

outClientFile << ' ';

outClientFile << tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand1() + i)->Inf.GetLex();

}

}

if (RunSent->Inf.Get\_number\_first\_lex\_operand2() != -1)

{

outClientFile << ',';

outClientFile << tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2())->Inf.GetLex();

}

if (RunSent->Inf.Get\_amount\_lex\_operand2() > 1)

{

for (i = 1; i < RunSent->Inf.Get\_amount\_lex\_operand2(); i++)

{

if (i < RunSent->Inf.Get\_amount\_lex\_operand2() - 1 &&

(tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2() + i)->Inf.cmpLex("PTR")))

outClientFile << ' ';

outClientFile << tab->FindLex(RunSent->Inf.Get\_number\_first\_lex\_operand2() + i)->Inf.GetLex();

}

}

outClientFile << endl;

RunSent = RunSent->Next;

}

outClientFile.unsetf(ios::dec);

outClientFile.setf(ios::hex);

outClientFile.setf(ios::uppercase);

outClientFile << (RunSent->Prev->Inf.GetWeight() >= 16 ? "00" : "000") << RunSent->Prev->Inf.GetWeight() << " ";

outClientFile.setf(ios::dec);

outClientFile << tab->FindLex(tab\_lex::GetIDEND("code") - 1)->Inf.GetLex() << " "

<< tab->FindLex(tab\_lex::GetIDEND("code"))->Inf.GetLex()

<< '\n' << " end" << endl;

outClientFile << "\n\n\n\n" << endl;

outClientFile << "Identificators : \n" << endl;

DynTabIdentificator \*RunID;

RunID = tab->TabIdentificator;

for (i = 0; i < TabIdentificator::GetAmount(); i++)

{

outClientFile << RunID->Inf.GetName() << " "

<< "............................................ ";

outClientFile.unsetf(ios::dec);

outClientFile.setf(ios::hex);

outClientFile.setf(ios::uppercase);

outClientFile << (RunID->Inf.GetValue() >= 16 ? "00" : "000") << RunID->Inf.GetValue() << " ";

outClientFile.setf(ios::dec);

outClientFile << RunID->Inf.GetAttr() << endl;

RunID = RunID->Next;

}

}

void writeLex(AllTab \*tab)

{

DynTab \*Run\_tab;

ofstream outClientFile("lex.lst", ios::out);

if (!outClientFile)

{

cerr << "Файл не мог быть открыт" << endl;

exit(1);

}

outClientFile << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

<<"---------------------------------Таблиця Лексем---------------------------------\n"

<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

<<" |№ п/п| |Лексема| |Довжина | |Тип лексеми|\n"

<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

Run\_tab = tab->LexTab;

for (int i = 0; i < tab->number\_str\_lex; i++)

{

outClientFile << " " << Run\_tab->Inf.Get\_number()

<< " " << Run\_tab->Inf.GetLex();

outClientFile.width(15);

outClientFile << " " << Run\_tab->Inf.Get\_len\_lex();

outClientFile << " " << Run\_tab->Inf.Get\_type\_lex() << endl; '\n';

Run\_tab = Run\_tab->Next;

}

}

void writeSentStruct(AllTab \*tab)

{

DynTab\_sint \*Run\_tab;

ofstream outClientFile("sint.lst", ios::out);

if (!outClientFile)

{

cerr << "Файл не мог быть открыт" << endl;

exit(1);

}

outClientFile << "\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

<< "\n---------------------------------Таблиця структури речення----------------\n"

<< "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

<< "|Поле міток| | Поле | | 1-й | | 2-й |\n"

<< "| (імені) | | мнемокоду | | операнд | | операнд |\n"

<< "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

<< "| № | |№першої|Кількість| |№ першої|Кількість| |№ першої|Кількість|\n"

<< "| лексеми | |лексеми| лексем | | лексеми| лексем | | лексеми| лексем |\n"

<< "| поля | | поля | поля | |операнда|операнда | |операнда|операнда |\n"

<< "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n" << endl;

Run\_tab = tab->TabSentStruct;

for (int i = 0; i < tab->TabSentStruct->Inf.Get\_number\_sent(); i++)

{

outClientFile << " "

<< Run\_tab->Inf.Get\_number\_lex\_field\_label() << setw(12)

<< Run\_tab->Inf.Get\_number\_first\_lex\_field\_mnemocode() << setw(12)

<< Run\_tab->Inf.Get\_amount\_lex\_field\_mnemocode() << setw(12)

<< Run\_tab->Inf.Get\_number\_first\_lex\_operand1() << setw(12)

<< Run\_tab->Inf.Get\_amount\_lex\_operand1() << setw(12)

<< Run\_tab->Inf.Get\_number\_first\_lex\_operand2() << setw(12)

<< Run\_tab->Inf.Get\_amount\_lex\_operand2() << endl;

Run\_tab = Run\_tab->Next;

}

}

void writeSegments(AllTab \*tab)

{

dyn\_tab\_segment \*RunSeg;

ofstream outClientFile("testing.lst", ios::out);

if (!outClientFile)

{

cerr << "Файл не мог быть открыт" << endl;

exit(1);

}

RunSeg = tab->SegmentTab;

do

{

outClientFile << RunSeg->Inf.GetName() << "........................";

outClientFile.unsetf(ios::dec);

outClientFile.setf(ios::hex);

outClientFile.setf(ios::uppercase);

outClientFile << (RunSeg->Inf.GetLength() >= 16 ? "00" : "000") << RunSeg->Inf.GetLength() << " " << endl;

outClientFile.setf(ios::dec);

RunSeg = RunSeg->Next;

} while (RunSeg != tab->SegmentTab);

}

**Тестування**:

Код програми

Data1 Segment

Vb db 10011b

String db 'Рядок – new'

Vw dw 4567d

Vd dd 0d7856fdah

Data1 ends

val equ 30

Code Segment

label1:

cli

IF val eq 33

Inc ax

ELSE

Dec ds:byte ptr[bp+277]

ENDIF

Or ax,dx

Cmp Ax, cs:word ptr[si+2]

Jna Label2

And word ptr[si+2],ax

Inc Bx

Add word ptr[si+256], 5

Mov dx, val

cmp ax, word ptr[si+2]

Jna label1

Label2:

Code ends

End

Лістинг

vanov Alexander (R) Thu May 21 09:23:36 2015

0000 DATA1 SEGMENT

0000 13 VB DB 10011B

0001 D0 FF E4 EE EA 20 96 20 6E 65 77 STRING DB Ðÿäîê – new

000C 11D7 VW DW 4567D

000E 0 VD DD 0D7856FDAH

0012 DATA1 ENDS

= 001E VAL EQU 30

0000 CODE SEGMENT

0000 LABEL1

0000 FA CLI

0001 3E: FE 8E 115 DEC DS:BYTE PTR[BP+277]

0006 ENDIF

0006 B C2 OR AX,DX

0008 2E: 3B 44 2 CMP AX,CS:WORD PTR[SI+2]

000C 76 11 JNA LABEL2

000E 21 44 2 AND WORD PTR[SI+2],AX

0011 43 INC BX

0012 83 84 100 5 ADD WORD PTR[SI+256],5

0017 BA 1E MOV DX,VAL

001A 3B 44 2 CMP AX,WORD PTR[SI+2]

001D 76 E1 JNA LABEL1

001F LABEL2

001F CODE ENDS

end

Identificators :

VB ............................................ 0000 DATA1

STRING ............................................ 0001 DATA1

VW ............................................ 000C DATA1

VD ............................................ 000E DATA1

LABEL1 ............................................ 0000 CODE

LABEL2 ............................................ 001F CODE

VAL ............................................ 001E

Лістинг MASM

Microsoft (R) Macro Assembler Version 5.00 5/21/15 09:24:02

Page 1-1

0000 Data1 Segment

0000 13 Vb db 10011b

0001 D0 FF E4 EE EA 20 96 String db 'Ðÿäîê – new'

20 6E 65 77

000C 11D7 Vw dw 4567d

000E DA6F85D7 Vd dd 0d7856fdah

0012 Data1 ends

= 001E val equ 30

assume ds:data1,cs:code

0000 Code Segment

0000 label1:

0000 FA cli

ELSE

0001 3E: FE 8E 0115 Dec ds:byte ptr[bp+277]

ENDIF

0006 0B C2 Or ax,dx

0008 2E: 3B 44 02 Cmp Ax, cs:word ptr[si+2]

000C 76 11 Jna Label2

000E 21 44 02 And word ptr[si+2],ax

0011 43 Inc Bx

0012 83 84 0100 05 Add word ptr[si+256], 5

0017 BA 001E Mov dx, val

001A 3B 44 02 cmp ax, word ptr[si+2]

001D 76 E1 Jna label1

001F Label2:

001F Code ends

end

Microsoft (R) Macro Assembler Version 5.00 5/21/15 09:24:02

Symbols-1

Segments and Groups:

N a m e Length Align Combine Class

CODE . . . . . . . . . . . . . . 001F PARA NONE

DATA1 . . . . . . . . . . . . . 0012 PARA NONE

Symbols:

N a m e Type Value Attr

LABEL1 . . . . . . . . . . . . . L NEAR 0000 CODE

LABEL2 . . . . . . . . . . . . . L NEAR 001F CODE

STRING . . . . . . . . . . . . . L BYTE 0001 DATA1

VAL . . . . . . . . . . . . . . NUMBER 001E

VB . . . . . . . . . . . . . . . L BYTE 0000 DATA1

VD . . . . . . . . . . . . . . . L DWORD 000E DATA1

VW . . . . . . . . . . . . . . . L WORD 000C DATA1

@FILENAME . . . . . . . . . . . TEXT test

36 Source Lines

36 Total Lines

11 Symbols

51026 + 465518 Bytes symbol space free

0 Warning Errors

0 Severe Errors