

در این پروژه ما کامپایلری را تا مراحل lexical analyzer & syntax analyzer را پیاده سازی کردیم. این دو را بر اساس گرامر از پیش تعریف شده داده شده به ما انجام دادیم:

```

<S> => Program <VARS> <BLOCKS> end
<VARS> => Var Identifier; <VARS> | Epsilon
<BLOCKS> => Start <STATES> End
<STATES> => <STATE> <M_STATES>
<STATE> => <BLOCKS> | <IF> | <IN> | <OUT> | <ASSIGN> | <LOOP>
<M_STATES> => <STATES> | Epsilon
<OUT> => Print ( <EXPR> ) ;
<IN> => Read ( Identifier ) ;
<IF> => If ( <EXPR> <O> <EXPR> ) { <STATE> }
<LOOP> => Iteration ( <EXPR> <O> <EXPR> ) { <STATE> }
<ASSIGN> => Put Identifier = <EXPR> ;
<O> => < | > | ==
<EXPR> => <EXPR> + <R> | <EXPR> - <R> | <R>
>R> => Identifier | Integer

```

اول این گرامر را رفع ابهام کرده و بعد از آن اول کد tokenize کردن را نوشته تا بتوان ورودی های گرفته شده را به صورت token های مختلف داشته باشیم و آن ها را داخل یک vector از struct های token ای داشته باشیم که خود آن را تعریف کرده، که شامل یک type از جنس TokenType، string ای به نام value که در آن value مربوطه نگهداری می شود و int line که برای نگهداری شماره خط token است.

```

struct Token {
    Token_Type type;
    string value;
    int line = 1;
};

```

حال خود Token\_Type یک Enumeration است که از حرف های مربوط به گرامر تشکیل شده، هر کدام از این عناصر ها در قسمت پایین آن با value های مربوطه به خود برای match کردن و مقایسه عناصر map شده هستند.

```
enum Token_Type {
    PROGRAM, VAR, IDENTIFIER, START, END, PROGRAM_END, PRINT, READ, IF, ITERATION,
    PUT,
    ASSIGN, LPAREN, RPAREN, LBRACE, RBRACE, SEMICOLON, PLUS, MINUS, LESS, GREATER,
    EQUAL,
    INTEGER, UNKNOWN
};

const char* tokenTypeNames[] = {
    "PROGRAM", "VAR", "IDENTIFIER", "START", "END", "PROGRAM_END", "PRINT", "READ",
    "IF", "ITERATION", "PUT",
    "ASSIGN", "LPAREN", "RPAREN", "LBRACE", "RBRACE", "SEMICOLON", "PLUS", "MINUS",
    "LESS", "GREATER", "EQUAL",
    "INTEGER", "UNKNOWN", "PUNCTUATION"
};

map<string, Token_Type> keywords = {
    {"Program", PROGRAM}, {"Var", VAR}, {"Start", START}, {"End", END}, {"end",
PROGRAM_END},
    {"Print", PRINT}, {"Read", READ}, {"If", IF}, {"Iteration", ITERATION}, {"Put",
PUT}
};
```

همچنین لیست tokenTypeNames برای پیدا کردن عنصر مورد نظر با استفاده از index آن است که دقیقاً لیست enum بالا است.

کار اصلی tokenize کردن با استفاده از تابع getNextToken انجام می شود، این تابع رشته ورودی، مکان فعلی lookahead و شماره خط کد را به عنوان ورودی گرفته و با کمک از pos رشته ورودی را بخش بخش کرده و به صورت token های مختلف در میاورد، هر token خاصی به صورت خاص خود چک می شود مثلاً Integer به شکلی که آیا رشته آمده شده تا pos به صورت تماماً عددی است؟ اگر باشد پس Integer است و همین طور به ترتیب تمام قواعد چک می شود تا به آخر رشته برسیم، این token های خارج شده به token vector های ما اضافه می شوند تا ما در مراحل بعدی فقط از vector به دست آمده استفاده کنیم. این token ها بعداً همان ترمینال های می شوند که ما آنها را match میکنیم. این تابع هدف اصلی آن پیدا کردن token بعدی در رشته ما است و خروجی آن هم دقیقاً token بعدی مورد نیاز است، این تابع با دیدن “\n” یک عدد به current\_line اضافه کرده و به این معنی است که یک خط به کد ما اضافه شده. بقیه whitespace ها در این تابع skip شده، چون برای ما مهم نیستند.

```

Token getNextToken(const string& input, size_t& pos, int& current_line) {
    while (pos < input.length() && isspace(input[pos])) {
        string temp;
        temp = temp + input[pos];
        if (temp == "\n") {
            current_line += 1;
        }
        pos++;
    }

    if (pos == input.length()) return { UNKNOWN, "", current_line };

    char currentChar = input[pos];

    if (isalpha(currentChar)) {
        string identifier;
        while (pos < input.length() && isalnum(input[pos])) {
            identifier += input[pos++];
        }
        if (keywords.find(identifier) != keywords.end()) {
            return { keywords[identifier], identifier, current_line };
        }
        else {
            return { IDENTIFIER, identifier, current_line };
        }
    }

    if (isdigit(currentChar)) {
        string number;
        while (pos < input.length() && isdigit(input[pos])) {
            number += input[pos++];
        }
        return { INTEGER, number, current_line };
    }

    switch (currentChar) {
    case '=':
        if (input[pos + 1] == '=') {
            pos += 2;
            return { EQUAL, "==", current_line };
        }
        else {
            pos++;
            return { ASSIGN, "=", current_line };
        }
    case '+': pos++; return { PLUS, "+", current_line };
    case '-': pos++; return { MINUS, "-", current_line };
    case '<': pos++; return { LESS, "<", current_line };
    case '>': pos++; return { GREATER, ">", current_line };
    case '(': pos++; return { LPAREN, "(", current_line };
    case ')': pos++; return { RPAREN, ")", current_line };
    case '{': pos++; return { LBRACE, "{", current_line };
    case '}': pos++; return { RBRACE, "}", current_line };
    case ';': pos++; return { SEMICOLON, ";", current_line };
    default:
        pos++;
        return { UNKNOWN, string(1, currentChar), current_line };
    }
}

```

```
}
```

پس از اینکه خروجی tokenize شده به کاربر نشان داده شد، لیستی از تمام token های به دست آمده به صورت مرتب به کاربر نمایش داده شد.

بعد از کار ما به بخش parse کردن این token ها و رشته ورودی می رسد.

برای پارس کردن ما از کلاسی به نام parser استفاده کردیم. در این کلاس تابعی به نام parse وجود دارد که فرایند parse کردن را شروع می کند این تابع فرآیند پارس کردن را با قاعده شروع ما یعنی S شروع می کند. این پارسر بر اساس قواعدی که در بالا در گرامر ما وجود داشت و تبدیل به کد شده بودند شروع به پارس کردن و تجزیه رشته می کند، تنها نکات مهم این قسمت تابع Expect است، از آنجا که در کد ما هر متغیر گرامر را تبدیل به یک function در class کرده ایم، تابع Expect زمانی استفاده می شود که ما به یک ترمینال رسیده باشیم و بخواهیم آن را match کنیم پس انتظار داریم که آن ترمینال یا token را در آن محل ببینیم. این تابع دقیقاً همین کار را می کند و با دیدن token type تصمیم میگیرد که آیا باید آن را match کند یا نه، اگر این کار را بکند که مشکلی به وجود نمی آید و پارسر به راه خود ادامه می دهد در صورتی که مشکلی پیش بیاید و خطای رخ دهد این پارسر آن را به ما نشان می دهد و flag قبول کردن رشته را false می کند و رشته را reject می کند. همچنین در قسمت اول این کلاس مقادیر token و parserAccept Boolean برای مشخص کردن accept کردن و یا reject کردن را در تابع constructor خود را از فراخوان دریافت میکند و به آنها مقدار می دهد. که این اتفاق ها در قسمت public خود اتفاق می افتد.

```
class Parser {
public:
    Parser(const vector<Token>& tokens, bool parserAccept) : tokens(tokens),
currentTokenIndex(0), parserAccept(parserAccept) {}

    void parse() {
        SetConsoleTextAttribute(hand, bluecolor);
        cout << "\nPlease enter any button to continue ... \n\n";
        SetConsoleTextAttribute(hand, defcolor);
        _getch();
        S();
        cout << "\n\nParsing complete." << endl;
        if (parserAccept) {
            SetConsoleTextAttribute(hand, bluecolor);
            cout << "\n --- Parser accepted the given string!\n";
        }
        else {
            SetConsoleTextAttribute(hand, redcolor);
            cout << "\n --- Parser rejected the given string!";
        }
        SetConsoleTextAttribute(hand, defcolor);
    }
private:
```

```

vector<Token> tokens;
size_t currentTokenIndex;
bool parserAccept;

void parserReject() {
    parserAccept = false;
}

Token getNextToken() {
    if (currentTokenIndex < tokens.size()) {
        return tokens[currentTokenIndex++];
    }
    return { UNKNOWN, "" };
}

void S() {
    expect(PROGRAM);
    vars();
    blocks();
    expect(PROGRAM_END);
}

void vars() {
    if (lookahead(VAR)) {
        expect(VAR);
        expect(IDENTIFIER);
        expect(SEMICOLON);
        vars();
    }
}

void blocks() {
    expect(START);
    states();
    expect(END);
}

void states() {
    state();
    m_states();
}

void state() {
    if (lookahead(START)) {
        blocks();
    }
    else if (lookahead(IF)) {
        if_state();
    }
    else if (lookahead(READ)) {
        in_state();
    }
    else if (lookahead(PRINT)) {
        out_state();
    }
    else if (lookahead(PUT)) {
        assign_state();
    }
}

```

```

        else if (lookahead(ITERATION)) {
            loop_state();
        }
    }

    void m_states() {
        //if (lookahead(START) || lookahead(IF) || lookahead(READ) ||
lookahead(PRINT) || lookahead(PUT) || lookahead(ITERATION) )
        if (lookahead(START) || lookahead(IF) || lookahead(READ) || lookahead(PRINT)
|| lookahead(PUT) || lookahead(ITERATION)) {
            states();
        }
    }

    void out_state() {
        expect(PRINT);
        expect(LPAREN);
        expr();
        expect(RPAREN);
        expect(SEMICOLON);
    }

    void in_state() {
        expect(READ);
        expect(LPAREN);

        expect(IDENTIFIER);
        expect(RPAREN);
        expect(SEMICOLON);
    }

    void if_state() {
        expect(IF);
        expect(LPAREN);
        expr();
        o();
        expr();
        expect(RPAREN);
        expect(LBRACE);
        state();
        expect(RBRACE);
    }

    void loop_state() {
        expect(ITERATION);
        expect(LPAREN);
        expr();
        o();
        expr();
        expect(RPAREN);
        expect(LBRACE);
        state();
        expect(RBRACE);
    }

    void assign_state() {
        expect(PUT);
        expect(IDENTIFIER);
    }

```

```

        expect(ASSIGN);
        expr();
        expect(SEMICOLON);
    }

    void o() {
        if (lookahead(LESS)) {
            expect(LESS);
        }
        else if (lookahead(GREATER)) {
            expect(GREATER);
        }
        else if (lookahead(EQUAL)) {
            expect(EQUAL);
        }
    }

    void expr() {
        r();
        if (lookahead(PLUS)) {
            expect(PLUS);
            expr();
        }
        else if (lookahead(MINUS)) {
            expect(MINUS);
            expr();
        }
    }

    void r() {
        if (lookahead(IDENTIFIER)) {
            expect(IDENTIFIER);
        }
        else if (lookahead(INTEGER)) {
            expect(INTEGER);
        }
    }

    bool lookahead(Token_Type type) {
        if (currentTokenIndex < tokens.size()) {
            return tokens[currentTokenIndex].type == type;
        }
        return false;
    }

    void expect(Token_Type type) {
        Token token = getNextToken();
        if (token.type != type) {
            SetConsoleTextAttribute(hand, redcolor);
            cout << "Syntax error: expected " << tokenTypeNames[type] << ", got " <<
tokenTypeNames[token.type] << ", at token index of " << currentTokenIndex << ", at
token line of " << token.line << endl;
            SetConsoleTextAttribute(hand, defcolor);
            parserReject();
            //exit(1);
        }
        else {
            cout << "match(" << tokenTypeNames[token.type] << ")\n";

```

```

    }
}

const char* tokenTypeNames[25] = {
    "PROGRAM", "VAR", "IDENTIFIER", "START", "END", "PROGRAM_END", "PRINT",
    "READ", "IF", "ITERATION", "PUT",
    "ASSIGN", "LPAREN", "RPAREN", "LBRACE", "RBRACE", "SEMICOLON", "PLUS",
    "MINUS", "LESS", "GREATER", "EQUAL",
    "INTEGER", "UNKNOWN", "PUNCTUATION"
};
};

```

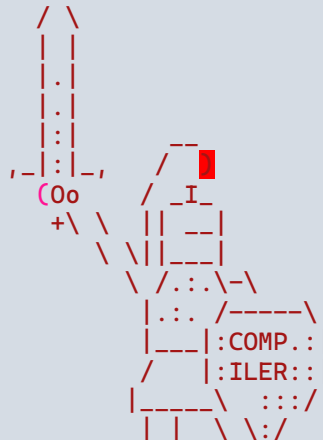
این دو قسمت پایه اصلی پروژه ما هستند، حال قسمت های دیگری هم وجود دارند که بیشتر در جهت زیبایی و خوانایی کار اضافه شدند.

قسمت Main menu منوی اصلی برنامه است که در آن می‌توانید شکل ورود رشته ورودی را مشخص کنید، اینکه آیا آن را می‌خواهید دستی وارد کنید یا اینکه آن را از فایل بخوانید. این تابع خروجی string دارد که همان input ما برای رشته است.

```
string mainMenu() {
    int midwayPointX = 60;
    int midwayPointY = 10;

    system("MODE 102,60");
    string message = "Compiler Final Project";
    gotoxy(midwayPointX - message.length(), midwayPointY);
    SetConsoleTextAttribute(hand, yellowcolor);
    cout << message;
    message = "Designed and programmed by Kiamehr Behnia";
    gotoxy(midwayPointX - (message.length() / 1.25) + 2, midwayPointY + 1);
    cout << message;

    SetConsoleTextAttribute(hand, bluecolor);
    gotoxy(0, midwayPointY + 3);
    cout << R"(
```





```

    | | | | |
    \ / \ / \
    '-'');

```

```

_getch();
SetConsoleTextAttribute(hand, defcolor);
system("CLS");

message = "Please select one of the commands: ";
gotoxy(midwayPointX - message.length() / 1.5, midwayPointY);
SetConsoleTextAttribute(hand, yellowcolor);
cout << message;

message = "1- Write your own string. ";
gotoxy(midwayPointX - 20, midwayPointY + 2);
SetConsoleTextAttribute(hand, greencolor);
cout << message;

message = "2- Read from a file. ";
gotoxy(midwayPointX - 20, midwayPointY + 3);
SetConsoleTextAttribute(hand, bluecolor);
cout << message;

message = "3- quit. ";
gotoxy(midwayPointX - 20, midwayPointY + 4);
SetConsoleTextAttribute(hand, redcolor);
cout << message;
gotoxy(midwayPointX - 20, midwayPointY + 5);

SetConsoleTextAttribute(hand, defcolor);
string option;
string input;
cin >> option;

while (true) {
    if (option == "1") {
        gotoxy(midwayPointX - message.length() / 1.5, midwayPointY + 6);
        cout << "Please enter a string: ";
        cin >> input;
        return input;
    }

    else if (option == "2") {
        string filename = "input.txt";
        ifstream f(filename);
        SetConsoleTextAttribute(hand, graycolor);
        gotoxy(midwayPointX - 20, midwayPointY + 8);
        cout << "Reading from " << filename << ":\n\n";
        SetConsoleTextAttribute(hand, defcolor);
        if (!f.is_open()) {
            cerr << "Error opening the file!";
            exit(0);
        }
        string str((istreambuf_iterator<char>(f),
            istreambuf_iterator<char>()),
            cout << str;
        _getch();
    }
}

```

```

        return str;
    }

    else if (option == "3") {
        exit(0);
    }
}

```

سیستم حرکت مکان نوشتن در محیط command prompt که با استفاده از تابع gotoxy() کنترل می شود.

```

void gotoxy(short x, short y) {
    COORD pos = { x,y };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos);
}

```

سیستم عوض کردن رنگ که از handle از پیش تعیین شده استفاده میکند و با استفاده از #define های اول فایل کد رنگ آن کنترل می شود و همچنین از:

```
SetConsoleTextAttribute(handle, color);
```

استفاده می شود تا رنگ متن محیط command prompt عوض شود.

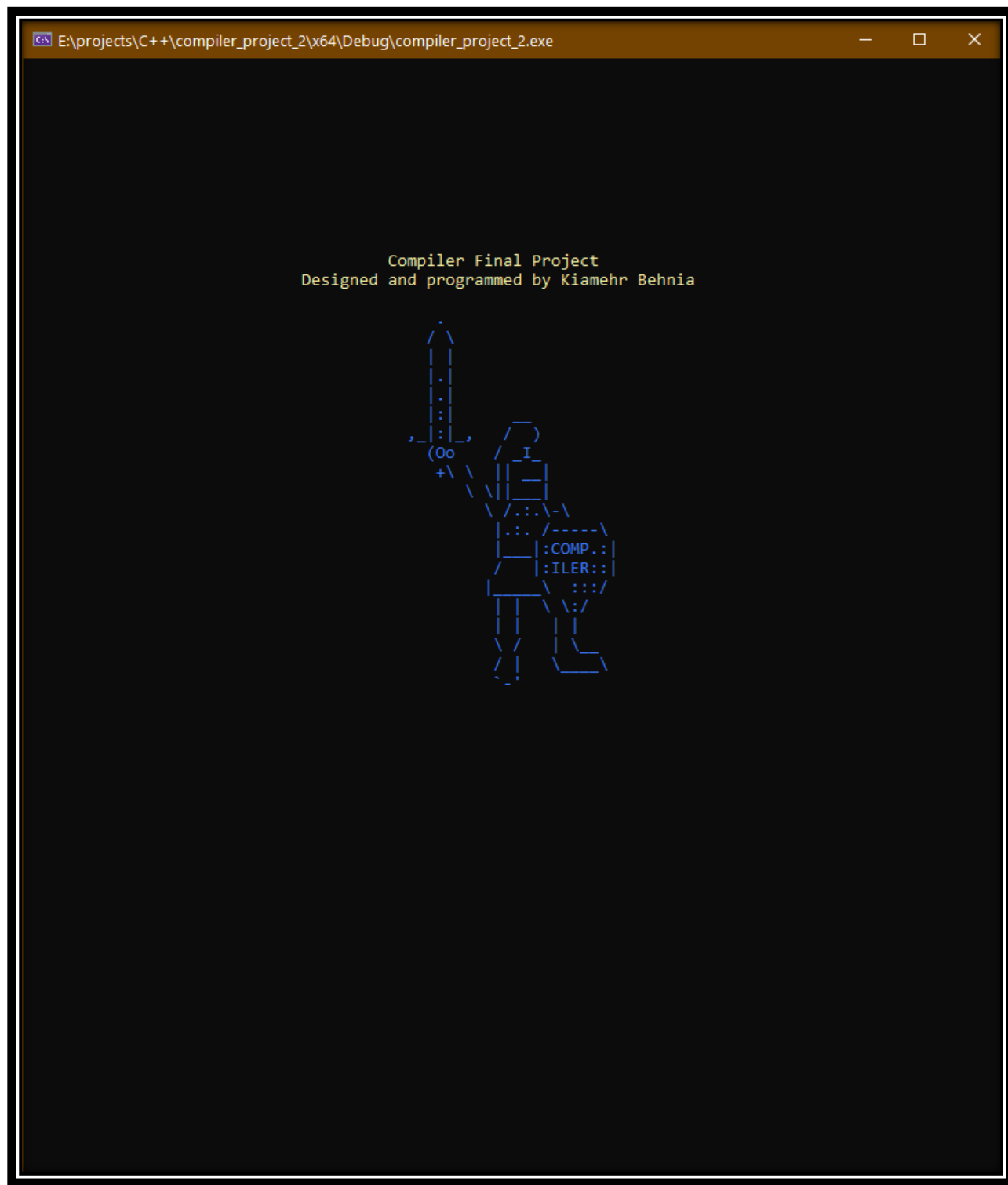
```

#define defcolor 7
#define graycolor 8
#define greencolor 2
#define bluecolor 9
#define yellowcolor 14
#define brightwhitecolor 15
#define redcolor 4

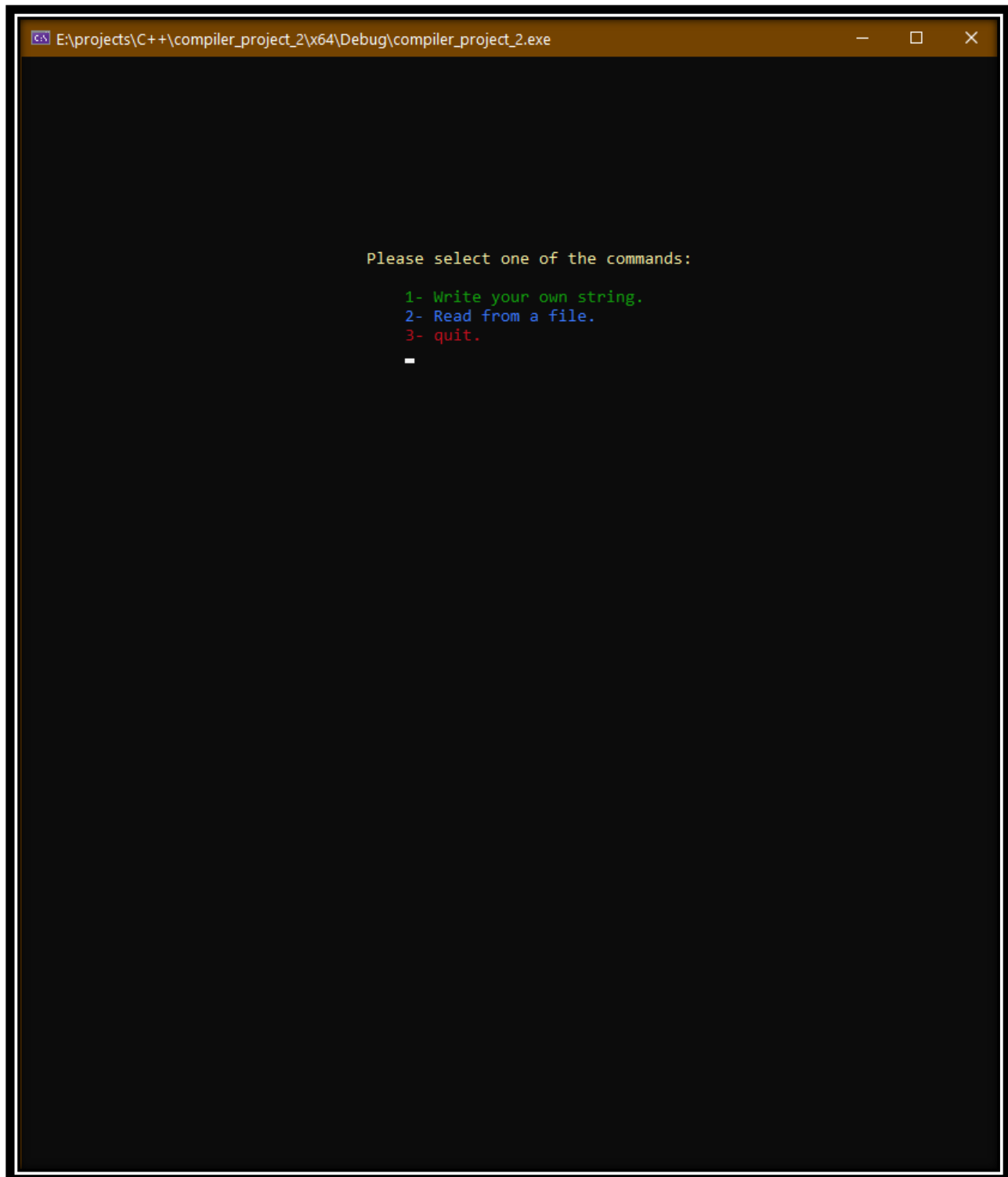
HANDLE hand = GetStdHandle(STD_OUTPUT_HANDLE);

```

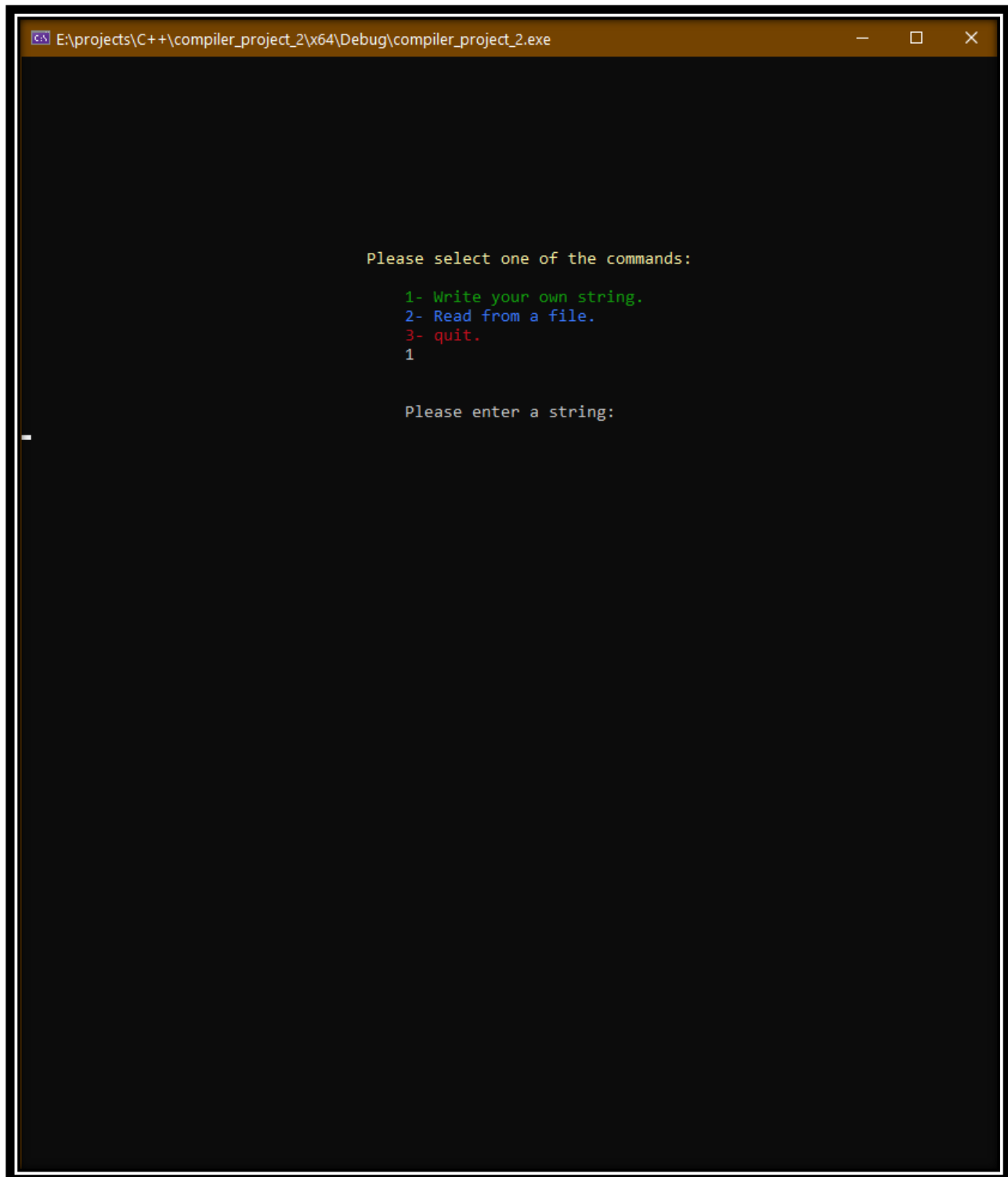
صفحه اولی که آن را میبینیم و منوی اصلی ما را تشکیل می دهد:



در این منو ما حق انتخاب بین وارد کردن رشته به صورت دستی، خواندن رشته از فایل و خروج از برنامه را داریم:



در صورتی که ما گزینه اول را انتخاب کنیم، برنامه از ما رشته ورودی را می گیرد.



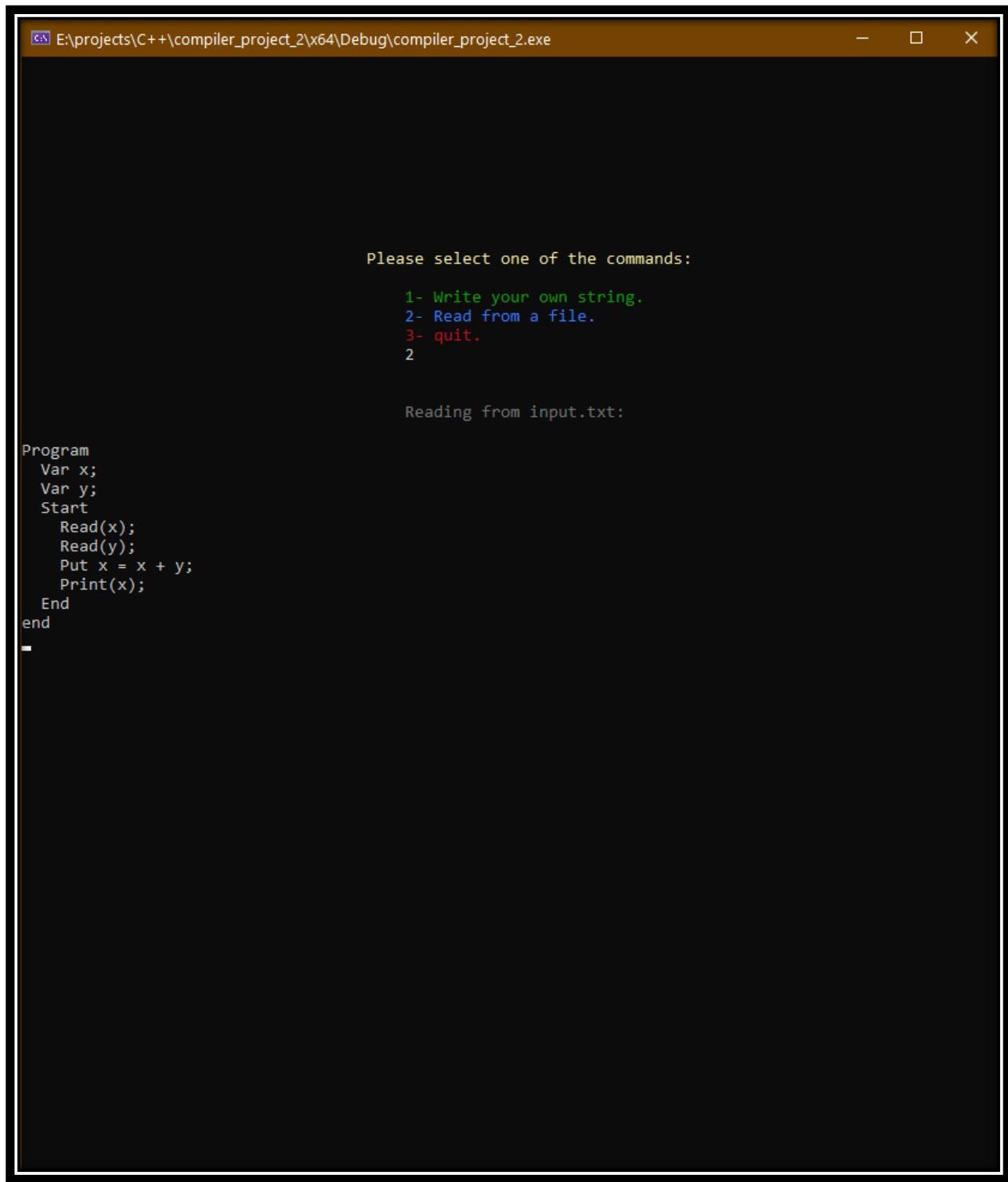
```
E:\projects\C++\compiler_project_2\x64\Debug\compiler_project_2.exe

Please select one of the commands:

1- Write your own string.
2- Read from a file.
3- quit.
1

Please enter a string:
-
```

ولی اگر ما گزینه دوم را انتخاب کنیم برنامه به صورت خودکار از فایل Input.txt رشته را خوانده و به عنوان Input گرفته.



The screenshot shows a window titled "E:\projects\C++\compiler\_project\_2\x64\Debug\compiler\_project\_2.exe". The window contains a menu with three options: "1- Write your own string.", "2- Read from a file.", and "3- quit.". The number "2" is entered, and the text "Reading from input.txt:" is displayed. Below this, the program code is shown, including variable declarations, input/output functions, and a loop structure.

```
Program
Var x;
Var y;
Start
  Read(x);
  Read(y);
  Put x = x + y;
  Print(x);
End
end
-
```

در مرحله بعد جدول token table را از token های ورودی دریافت کرده و درست می کند:

```
E:\projects\C++\compiler_project_2\x64\Debug\compiler_project_2.exe
Input:
~
Program
Var x;
Var y;
Start
  Read(x);
  Read(y);
  Put x = x + y;
  Print(x);
End
end
~

Tokenized input:

Token Type:      PROGRAM|      Value:      Program|      Token line: 1|
Token Type:      VAR|      Value:      Var|      Token line: 2|
Token Type:      IDENTIFIER|      Value:      x|      Token line: 2|
Token Type:      SEMICOLON|      Value:      ;|      Token line: 2|
Token Type:      VAR|      Value:      Var|      Token line: 3|
Token Type:      IDENTIFIER|      Value:      y|      Token line: 3|
Token Type:      SEMICOLON|      Value:      ;|      Token line: 3|
Token Type:      START|      Value:      Start|      Token line: 4|
Token Type:      READ|      Value:      Read|      Token line: 5|
Token Type:      LPAREN|      Value:      (|      Token line: 5|
Token Type:      IDENTIFIER|      Value:      x|      Token line: 5|
Token Type:      RPAREN|      Value:      )|      Token line: 5|
Token Type:      SEMICOLON|      Value:      ;|      Token line: 5|
Token Type:      READ|      Value:      Read|      Token line: 6|
Token Type:      LPAREN|      Value:      (|      Token line: 6|
Token Type:      IDENTIFIER|      Value:      y|      Token line: 6|
Token Type:      RPAREN|      Value:      )|      Token line: 6|
Token Type:      SEMICOLON|      Value:      ;|      Token line: 6|
Token Type:      PUT|      Value:      Put|      Token line: 7|
Token Type:      IDENTIFIER|      Value:      x|      Token line: 7|
Token Type:      ASSIGN|      Value:      =|      Token line: 7|
Token Type:      IDENTIFIER|      Value:      x|      Token line: 7|
Token Type:      PLUS|      Value:      +|      Token line: 7|
Token Type:      IDENTIFIER|      Value:      y|      Token line: 7|
Token Type:      SEMICOLON|      Value:      ;|      Token line: 7|
Token Type:      PRINT|      Value:      Print|      Token line: 8|
Token Type:      LPAREN|      Value:      (|      Token line: 8|
Token Type:      IDENTIFIER|      Value:      x|      Token line: 8|
Token Type:      RPAREN|      Value:      )|      Token line: 8|
Token Type:      SEMICOLON|      Value:      ;|      Token line: 8|
Token Type:      END|      Value:      End|      Token line: 9|
Token Type:      PROGRAM_END|      Value:      end|      Token line: 10|
-----
Please enter any button to continue ...
```

پس از فشردن هر کلیدی ما به بخش parse کردن می رویم و شروع به parse کردن رشته می کنیم:

```

Microsoft Visual Studio Debug Console
Token Type: IDENTIFIER Value: x Token line: 7
Token Type: ASSIGN Value: = Token line: 7
Token Type: IDENTIFIER Value: x Token line: 7
Token Type: PLUS Value: + Token line: 7
Token Type: IDENTIFIER Value: y Token line: 7
Token Type: SEMICOLON Value: ; Token line: 7
Token Type: PRINT Value: Print Token line: 8
Token Type: LPAREN Value: ( Token line: 8
Token Type: IDENTIFIER Value: x Token line: 8
Token Type: RPAREN Value: ) Token line: 8
Token Type: SEMICOLON Value: ; Token line: 8
Token Type: END Value: End Token line: 9
Token Type: PROGRAM_END Value: end Token line: 10

Please enter any button to continue ...

match(PROGRAM)
match(VAR)
match(IDENTIFIER)
match(SEMICOLON)
match(VAR)
match(IDENTIFIER)
match(SEMICOLON)
match(START)
match(READ)
match(LPAREN)
match(IDENTIFIER)
match(RPAREN)
match(SEMICOLON)
match(READ)
match(LPAREN)
match(IDENTIFIER)
match(RPAREN)
match(SEMICOLON)
match(PUT)
match(IDENTIFIER)
match(ASSIGN)
match(IDENTIFIER)
match(PLUS)
match(IDENTIFIER)
match(SEMICOLON)
match(PRINT)
match(LPAREN)
match(IDENTIFIER)
match(RPAREN)
match(SEMICOLON)
match(END)
match(PROGRAM_END)

Parsing complete.

--- Parser accepted the given string!

E:\projects\C++\compiler_project_2\x64\Debug\compiler_project_2.exe (process 18316) exited with code 0
.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatical

```



از آنجایی که معلوم است تمام token ها match شده و در آخر با پیغام آبی رنگ به ما اعلام می کند که آیا parser رشته را accept کرده است یا خیر. در ادامه ما همین رشته را با حذف کردن قسمت آخر آن یعنی End دوم را حذف می کنیم و دوباره اجرا خواهیم گرفت:

```

Microsoft Visual Studio Debug Console
Token Type: SEMICOLON Value: ; Token line: 6
Token Type: PUT Value: Put Token line: 7
Token Type: IDENTIFIER Value: x Token line: 7
Token Type: ASSIGN Value: = Token line: 7
Token Type: IDENTIFIER Value: x Token line: 7
Token Type: PLUS Value: + Token line: 7
Token Type: IDENTIFIER Value: y Token line: 7
Token Type: SEMICOLON Value: ; Token line: 7
Token Type: PRINT Value: Print Token line: 8
Token Type: LPAREN Value: ( Token line: 8
Token Type: IDENTIFIER Value: x Token line: 8
Token Type: RPAREN Value: ) Token line: 8
Token Type: SEMICOLON Value: ; Token line: 8
Token Type: PROGRAM_END Value: end Token line: 10

Please enter any button to continue ...

match(PROGRAM)
match(VAR)
match(IDENTIFIER)
match(SEMICOLON)
match(VAR)
match(IDENTIFIER)
match(SEMICOLON)
match(START)
match(READ)
match(LPAREN)
match(IDENTIFIER)
match(RPAREN)
match(SEMICOLON)
match(READ)
match(LPAREN)
match(IDENTIFIER)
match(RPAREN)
match(SEMICOLON)
match(PUT)
match(IDENTIFIER)
match(ASSIGN)
match(IDENTIFIER)
match(PLUS)
match(IDENTIFIER)
match(SEMICOLON)
match(PRINT)
match(LPAREN)
match(IDENTIFIER)
match(RPAREN)
match(SEMICOLON)
Syntax error: expected END, got PROGRAM_END, at token index of 31, at token line of 10
Syntax error: expected PROGRAM_END, got UNKNOWN, at token index of 31, at token line of 1

Parsing complete.

--- Parser rejected the given string!
E:\projects\C++\compiler_project_2\x64\Debug\compiler_project_2.exe (process 12504) exited with code 0
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatical

```

همان طور که معلوم است در این رشته که شکل صحیح و اشتباه آن به شکل زیر است:

Incorrect format	Correct format
<pre>Program   Var x;   Var y;   Start     Read(x);     Read(y);     Put x = x + y;     Print(x);  end</pre>	<pre>Program   Var x;   Var y;   Start     Read(x);     Read(y);     Put x = x + y;     Print(x);   End end</pre>

در این جا برنامه از قسمت End ایراد خواهد گرفت و پیغام زیر را به ما به رنگ قرمز نشان خواهد داد:

**Syntax error: expected END, got PROGRAM\_END, at token index of 31, at token line of 10**

این پیغام با این معنا است که خطای سینتکسی به وجود آمده و در این قسمت که در خط 10 و شماره token 31 قرار دارد ما انتظار داشته بودیم که END را ببینیم ولی PROGRAM\_END یا همان end را دیده ایم که این با آنچه که بر اساس گرامر باید می دیدیم فرق دارد پس در آخر رشته ما reject خواهد شد:

**--- Parser rejected the given string!**

```

Microsoft Visual Studio Debug Console
Token Type: SEMICOLON Value: ; Token line: 6
Token Type: PUT Value: Put Token line: 7
Token Type: IDENTIFIER Value: x Token line: 7
Token Type: ASSIGN Value: = Token line: 7
Token Type: IDENTIFIER Value: x Token line: 7
Token Type: PLUS Value: + Token line: 7
Token Type: IDENTIFIER Value: y Token line: 7
Token Type: SEMICOLON Value: ; Token line: 7
Token Type: PRINT Value: Print Token line: 8
Token Type: LPAREN Value: ( Token line: 8
Token Type: IDENTIFIER Value: x Token line: 8
Token Type: RPAREN Value: ) Token line: 8
Token Type: SEMICOLON Value: ; Token line: 8
Token Type: PROGRAM_END Value: end Token line: 10
-----
Please enter any button to continue ...

match(PROGRAM)
match(VAR)
match(IDENTIFIER)
match(SEMICOLON)
match(VAR)
match(IDENTIFIER)
match(SEMICOLON)
match(START)
match(READ)
match(LPAREN)
match(IDENTIFIER)
match(RPAREN)
match(SEMICOLON)
match(READ)
match(LPAREN)
match(IDENTIFIER)
match(RPAREN)
match(SEMICOLON)
match(PUT)
match(IDENTIFIER)
match(ASSIGN)
match(IDENTIFIER)
match(PLUS)
match(IDENTIFIER)
match(SEMICOLON)
match(PRINT)
match(LPAREN)
match(IDENTIFIER)
match(RPAREN)
match(SEMICOLON)
Syntax error: expected END, got PROGRAM_END, at token index of 31, at token line of 10
Syntax error: expected PROGRAM_END, got UNKNOWN, at token index of 31, at token line of 1
Parsing complete.
--- Parser rejected the given string!
E:\projects\C++\compiler_project_2\x64\Debug\compiler_project_2.exe (process 12504) exited with code 0
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatical

```

در آخر مثال های بیشتری در پوشه پروژه قرار دارد، که آنها در پوشه Examples قرار دارند.