מסמך תיעוד הקומפיילר

נפתאלי אבידאייב : 319461083 - Naftaly.a@campus.technion.ac.il

אבומוך מוחמד : 300213196 – mfa1603@gmail.com

הקומפיילר בתמונה הגדולה:

תכננו קומפיילר שמקמפל שפת cmm כפי שהוגדרה בדרישות הפרוייקט בחלקיו הקודמים לשפת riski כפי שהוגדרה בחלק השלישי של הפרוייקט.

הקומפיילר מורכב משני חלקים:

1- מודל לניתוח ליקסיקלי (common.lex)
2- מודל לניתוח תחבירי וסימנטי (common.y)

מבני ניתונים:

לניהול הקומפיילר השתמשנו במודולים הבאים, אותם מימשנו בשפת c++:

- **Symbol**: קלאס שמייצג סמל שמכיל:
  - שם הסמל.
  - גודל הסמל.
  - היסט הסמל במחסנית.
- **SymbolTable**: קלאס שמנהל טבלאות של סמלים ומכיל:
  - רשימה של רשימות של סמלים.
  - רמת הסקופ של הסמל – (לניהול סקופינג ומיסוך).
  - היסטים.

  קלאס זה אחראי על ניהול הסמלים בקוד, בודק אם יש מיסוך ומסדיר אותו בין הסקופים השונים ובמקרה של שגיאות של יותר מסמל עם אותו שם, יודע גם לגרום לשגיאה בזמן הקומפילציה.

- **RegisterManager**: קלאס שמנהל הקצאות וספירת ריגסטרים במערכת ומכיל:
  - מונה רגיסטרים.
  - סקופ.

  קלאס זה אחראי על ניהול הרגסטרים, עבור על פונקציה מונה הרגסטרים מתאפס (ל -3 , ראה הסבר בניהול מחסנית).

- **Function**: קלאס שמייצג פונקציה שמכיל:
  - שם פונקציה.
  - ממומשת (כן או לא).
  - פונקציית main (כן או לא).
  - מחזירה ערך (כן או לא).
  - סוג ערך ההחזרה.
  - שורת תחילה.
  - ארגומנטים.
  - איפה בקוד קוראים לפונקציה.

- **Buffer**: באפר שמכיל את שורת הקוד בשפת ה-riski ובנוסף מכיל header כפי שנדרש בתרגיל. קלאס זה גם מנהל רשימות של פונקציות ממומשות ורשימה של לא-ממומשות. קלאס זה מספק מימוש ההטלאות הנדרשות (backpatching).

כל הקלאסים שהוזכרו למעלה, יש להם מופע גלובאלי שחי לאורך כל חיי התוכנית והם מאותחלים ב common.cpp

Backpatching:

כדי ליישם פריסת הקוד הוספנו רשימות למבנה yystype:

- true_list - מכילה את מס' שורות הקוד שצריך לעדכן אותן בכתובת הקפיצה במקרה של קיום התנאי
- false_list - מכילה את מס' שורות הקוד שצריך לעדכן אותן בכתובת הקפיצה במקרה של אי-קיום התנאי
- next_list – מכילה את מס' שורות הקוד שצריך לעדכן אותן בכתובת הבאה שצריך לקפוץ אליה.

וגם היינו צריכים להוסיף עוד שני מרקרים ליישום פעולות הקפיצות כולל עזרה לעידכון השורות ברשימות הקודמות.

השתמשנו ב- backpatching בחוקים הבאים:

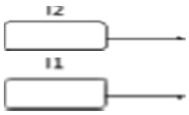
```
BLK : H_OPM STLIST M H_CPM
STLIST : STLIST M STMT
CNTRL : H_IF BEXP H_THEN M STMT H_ELSE N M STMT
      | H_IF BEXP H_THEN M STMT
      | H_WHILE M BEXP H_DO M STMT
BEXP : BEXP H_OR M BEXP
      | BEXP H_AND M BEXP
```

מבנה רשומת ההפעלה:

את ניהול המחסנית עשינו בהתאם להמלצה/דוגמה שמצורפת בתרגיל.
I0 – משמש להחזקת כתובת החזרה.

I1 – משמש להצבעה על בסיס מסגרת המחסנית.

I2 – משמש להצבעה על ראש המחסנית.



Callee locals…
Callee saved registers…
Function return value
Arg1
Arg2
…
Caller’s saved registers…
…

חשוב לציין שהקפיצות וההיסטים אנחנו מחשבים אותם עבור כל סמל, פקודה ובלוק.

נספחים:

YYSTYPE:

```
typedef struct {
    string name;
    unsigned int type;
    unsigned int quad;
    unsigned int node_reg;
    int node_offset;
} DCL_Node;


typedef struct{
    char*   value;
    unsigned int type;
    bool is_const;
    bool is_exp;
    int offset;
    unsigned int quad;
    unsigned int reg;
    vector<DCL_Node> dcl_list;
    set<unsigned int> true_list;
    set<unsigned int> false_list;
    set<unsigned int> next_list;
} yystype;
```

Function:

```
class Function{
    private:
        string m_name;
        bool m_isImplemented;
        bool m_isMain;
        bool m_hasReturn;
        unsigned int m_line;
        unsigned int m_returnType; // 0 - void, 1 - int8, 2 - int16, 4 - int32
        vector<unsigned int> m_args;
        vector<int> callLines;
    public:
    Function() {}
    Function(string name) : m_name(name),
                            m_isImplemented(false),
                            m_isMain(false),
                            m_hasReturn(false),
                            m_line(0),
```

```
                            m_returnType(0),
                            m_args()
    {}
    void clearArgs() {m_args.clear();}
    void setName(string name) {m_name = name;}
    void setReturnType(unsigned int returnType) {
        //cout << "setReturnType: " << returnType << endl;
        m_returnType = returnType;}
    void setIsImplemented(bool isImplemented) {m_isImplemented = isImplemented;}
    //void setIsMain(bool isMain) {m_isMain = isMain;}
    void setHasReturn(bool hasReturn) {m_hasReturn = hasReturn;}
    void setLine(unsigned int line) {m_line = line;}
    unsigned int getReturnType() const {return m_returnType;}
    bool getIsImplemented() const {return m_isImplemented;}
    bool getIsMain() const {return m_isMain;}
    bool getHasReturn() const {return m_hasReturn;}
    void addCallLine(int line) { callLines.push_back(line); };
    vector<int> getCallLine() {return callLines;}
    string getName() {return m_name;}
    unsigned int getLine() const {return m_line;}
    void addArgument(unsigned int arg) {m_args.push_back(arg);}
    void addArguments(vector<unsigned int> args) {m_args.insert(m_args.end(),
args.begin(), args.end());}
    bool operator==(Function &other){
        if ( (this->m_name != other.m_name) ||
             (this->m_returnType != other.m_returnType) ||
             (this->m_args.size() != other.m_args.size()) )
            return false;
        else
        {
            for (int i=0; i < m_args.size(); i++){
                if (this->m_args[i] != other.m_args[i]) return false;
            }
        }
        return true;
    }

    vector<unsigned int> getArguments() { return m_args; }
};
```

Register Manager:

```
class RegisterManager {
private:
    unsigned int counter;
    vector<unsigned int> scope;

public:
    RegisterManager();
    virtual ~RegisterManager();
    int getRegister(); //get next available register
    int getRegistersCount();
    void setRegistersCount(int toSet);
    void startScope();
    void endScope();
};
```

Symbol:

```
class Symbol {
        private:
                string m_name;
                unsigned int m_size;
                int m_offset;
        public:
        Symbol(string name, unsigned int size, int offset = 0);
        ~Symbol();
        bool operator==(const Symbol &symbol);
        string getName() const;
        unsigned int getSize() const;
        int getOffset() const;
        void setName(string name);
        void setSize(unsigned int size);
        void setOffset(int offset);
};
```

Symbol Table:

```
class SymbolTable {
    private:
```

```
    vector< vector<Symbol> > symbols;
    unsigned int level;
    int offset;
        int backwards_offset;
    public:
        SymbolTable();
        ~SymbolTable();
        void startBlock();
        void endBlock();
        int addSymbol(string name, unsigned int size);
        void addArgumentSymbol(string name, unsigned int size);
        Symbol& findSymbol(string name);
        int getOffset() const {
                return offset;
        }
};
```

Buffer:

```
class Buffer {
    private:
        string header;
        vector<string> code;
        unsigned int quad;
        vector<Function> implemented;
        vector<Function> unimplemented;

        public:
        Buffer();
        ~Buffer();
        void emit(string instruction);
        void backPatch(set<unsigned int> &lines, unsigned int &address);
        unsigned int nextQuad();
        unsigned int getQuad();
        void bufferToRiski(string filename);
        //void writeHeader(vector<Function> implemented, vector<Function>
unimplemented);
        void addFunction(Function func);
        Function& findFunction(string name);

};
```