שיטות הידור (קומפילציה) 046266

סמסטר חורף תשפ"ד

תרגיל בית רטוב מספר: 3

הסבר כללי על מימוש הקומפיילר:

הקומפיילר שבנינו מבצע תרגום של תוכנית בקובץ יחיד בשפת --C לשפת הסף $Riski$. הקומפיילר שייך למשפחת הקומפיילרים $LALR$ ומבצע תרגום של הקוד $Bottom-up$ במעבר יחיד, עד כדי תיקוני $Backpatching$. לצורך תרגום נכון ומדוייק של התוכנית בשפת המקור אנו משתמשים בכמה משתנים גלובליים שצוברים מידע שנאסף על התוכנית לאורך תהליך הקומפילציה ומאפשרים לנו בסיום תהליך הקומפילציה להפיק קובץ בשפת המטרה.

מבני נתונים וניהול מצב הקומפיילר:

1. $YYSTYPE$ – זהו מבנה המכיל את התכונות הסמנטיות שאנו מייחסים לסימבולים השונים שבדקדוק של שפת –C ובפרט מאפשר להעביר מידע בין סימבולים שונים ובכך "לחלחל" את המידע שנאסף בנקודה כלשהי לאורך תהליך התרגום, לנקודה אחרת שבה יהיה בו צורך. התכונות הסמנטיות שהקצנו:

```
/**
 * @brief The type of each symbol and token. Each symbol and token assign values only to the relevant fieslds
 */
typedef struct {
    string value;   /**< The value associated with the symbol */
    Type type;      /**< The type of the symbol */
    int memOffset;  /**< Offset from frame pointer I1 or F0 */
    int quadAddr;   /**< The quad address in the final riski code, starting from 1 */
    int regNum;     /**< Number of allocated registers for this symbol */

    /** Lists for further backpatching */
    vector<int> nextList;   /**< Next list */
    vector<int> trueList;   /**< True list */
    vector<int> falseList;  /**< False list */

    /** Lists for variables declaraion */
    vector<string> variablesList;   /**< List of names of declared variables of the same type */

    /** Lists for parameters */
    vector<string> paramsList;      /**< List of names of parameters of a function */
    vector<Type> paramsTypeList;    /**< List of type of the parameters of a function */

    /** Lists for arguments */
    vector<int> argsRegList;        /**< List of register numbers that holds the results of the expressions that are passed to a function */
    vector<Type> argsTypeList;      /**< List of the types of the arguments that are passed to a function */

} yystype;
```

- $Value$ – ערך מילולי כלשהו שאנו מייחסים לסימבול, למשל $identifier$.
- $Type$ – הטיפוס של הסימבול, $int, float, void$.
- $memOffset$ – היסט מה - $frame\ pointer$. משויך לערכים ששמורים במחסנית.
- $quadAddr$ – מספר פקודה בקוד שפת הסף. משמש ל - $Backpatching$.

- $regNum$ – מספר הרגיסטר שבו שמור הערך שמשויך לסימבול.
- $nextList$ – רשימה של מספרי פקודות בקוד שפת הסף שיש להטליא עם מספר הפקודה שאחרי בלוק בקרה. משמש ל - $Backpatching$.
- $trueList$ – רשימה של מספרי פקודות בקוד שפת הסף שיש להטליא עם מספר הפקודה שאליה יש לקפוץ אם ביטוי בוליאני התברר כ - $true$. משמש ל - $Backpatching$.
- $falseList$ – רשימה של מספרי פקודות בקוד שפת הסף שיש להטליא עם מספר הפקודה שאליה יש לקפוץ אם ביטוי בוליאני התברר כ - $false$. משמש ל - $Backpatching$.
- $paramsList$ – רשימה של שמות של משתנים ב - $statement$ של הצהרת משתנים או בחתימה של פונקציה שלכולם אותו טיפוס. שמות המשתנים נוספים לרשימה אחד אחרי השני כאשר לפני כל הוספה מתבצעת בדיקה האם שם המשתנה קיים כבר ברשימה ובטבלת הסמלים.
- $paramsList$ – רשימה של שמות הפרמטרים בחתימה של פונקציה ומורכבת מאיחוד של $paramsList$. כאשר הקומפיילר עובר על חתימה של פונקציה מתבצעת בדיקה אם שם הפרמטר הבא שאנו רואים כבר קיים ברשימת הפרמטרים שמכילה את שמות הפרמטרים שכבר ראינו. אם השם קיים אזי ישנם שני פרמטרים עם אותו שם. אם השם לא קיים אז הוא נוסף לרשימה. אם החתימה שייכת להגדרה של פונקציה, אז הרשימה גם מאפשרת לנו ליצור את הקוד שיוצר שדות בטבלת הסמלים לפרמטרים של הפונקציה.
- $TypesList$ – רשימה של טיפוסי הפרמטרים של פונקציה. כאשר הקומפיילר עובר על חתימה של פונקציה, הטיפוסים של הפרמטרים נוספים לרשימה. הרשימה מאפשרת לנו לזהות אי תאימות בין הגדרת פונקציה לבין הצהרת פונקציה. אם החתימה שייכת להגדרה של פונקציה, אז הרשימה גם מאפשרת לנו ליצור את הקוד שיוצר שדות בטבלת הסמלים לפרמטרים של הפונקציה.
- $argsRegList$ – רשימה של מספרי הרגיסטרים בהם שמורים ערכי הביטויים שמועברים לפונקציה כארגומנטים. משמש להעברת ארגומנטים לפונקציה.
- $argsTypeList$ – רשימה של טיפוסי הביטויים שמועברים לפונקציה כארגומנטים. אנו משווים בין הטיפוסים של הפרמטרים של הפונקציה השמורים בטבלת הסמלים לבין רשימה זו ולכן הרשימה משמשת לבדיקת תקינות הארגומנטים של הפונקציה.

2. $Class\ Symbol$ – מייצגת סימבול בטבלת הסמלים של הקומפיילר.

```
/**
 * @brief Represents a symbol
 */
class Symbol {
public:
    Type type;
    int memOffset;
};
```

- $type$ – טיפוס הסימבול $(int, float)$.
- $memOffset$ – ההיסט בבתים שבו שמור הסימבול במחסנית ביחס למיקום בו היה ה - $frame\ pointer$ כאשר ה - $frame$ הנוכחי של הפונקציה הוקצה.

3. $Class\ SymbolTable$ – מייצגת את טבלת הסמלים של הקומפיילר.

```
/**
 * @brief Represents the symbol table of our compiler
 */
class SymbolTable {
private:
    vector<map<string, Symbol>> symbolTable; /**< List of scopes symbol tables. The deeper the scope, the further it is in the list */
```

- $symbolTable$ – רשימה של טבלאות סמלים של $scopes$. ה - map מייצג טבלת סמלים של $scope$. המפתח הוא שם הסמל והערך הוא הסמל עצמו. רשימת טבלאות הסמלים של ה - $scopes$ מנוהלת כך שה - $scope$ הכי פנימי נמצא בסוף הרשימה, לפניו נמצא ה – $scope$ שמכיל אותו וכן הלאה. האיבר הראשון ברשימה הוא ה – $scope$ של הפונקציה הנוכחית.

  טבלת הסמלים תומכת בכמה פעולות חשובות:

  (a) פתיחת טבלת ל - $scope$ חדש. מתבצעת בתחילת הגדרה של פונקציה ובתחילת בלוק.

  (b) סגירת $scope$. מתבצעת ביציאה מבלוק או מסוף הגדרת פונקציה.

  (c) הוספת סימבול חדש ל - $scope$ הפנימי ביותר.

  (d) האם הסימבול קיים ב - $scope$ הנוכחי. שימושי כאשר אנחנו רוצים להקצות משתנה חדש ולבדוק האם משתנה נוסף בעל אותו שם קיים.

(e) החזרת מצביע ל - $scope$ הפנימי ביותר שבו קיים סימבול עם שם כלשהו.

שימושי כאשר אנו רוצים להתייחס לשם כלשהו בתוכנית.

4. $Class\ Function$ – מייצגת פונקציה בתוכנית.

```
/**
 * @brief Represents a function found in the C-- program
 */
class Function {
public:
    int definitionLine;             /**< The number of the first line of the definition in the Risky file */
    Type returnType;                /**< The return type of the function (int, float, void). Part of the signature */
    vector<Type> paramsTypeList;    /**< A list of the types of the parameters. Part of the signature */
    vector<int> calls;              /**< A list of line numbers in the Riski file where the function is called and need to be backpatched */
    bool isDefined;                 /**< True if the function is a definition. Helps to validate that a function is defined once */
};
```

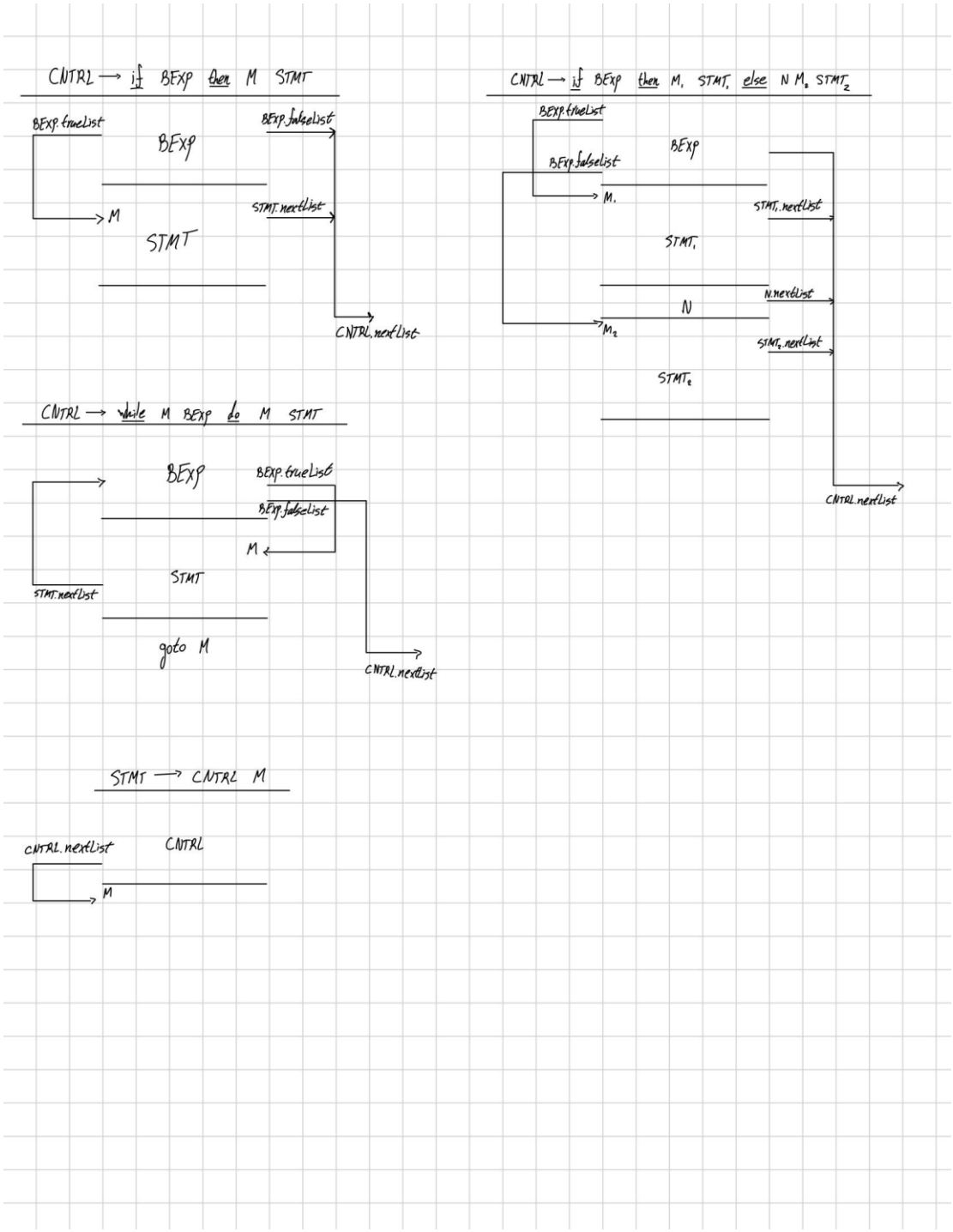
- $definitionLine$ – מספר פקודת ה - $Risky$ הראשונה בפונקציה. משמש לקפיצה לפונקציה בעת קריאה.
- $returnType$ – סוג ערך החזרה של הפונקציה. משמש לבדיקת תאימות בין הצהרה והגדרה ובין טיפוס ערך החזרה המוחזר בפועל לבין טיפוס ערך החזרה הצפוי.
- $paramsTypeList$ – רשימה של טיפוסי הפרמטרים של הפונקציה. משמש לבדיקת תאימות בין הצהרה והגדרה ובין טיפוסי הפרמטרים של הפונקציה לטיפוסי הארגומנטים שמועברים לה בעת קריאה.
- $calls$ – רשימה של מספרי שורות בקובץ ה - $Risky$ שבעת ההידור שלהן לא ידענו איפה נמצאת ההגדרה של הפונקציה לכן לא ידענו לאן לקפוץ. הרשימה משמשת לבניית ה - $header$ של ה - $linker$ שיטליא את הפקודות שבשורות שנמצאות ברשימה.
- $isDefined$ -$true$ אם הפונקציה מוגדרת. משמש לבדיקה שלכל פונקציה אין יותר מהגדרה יחידה.

5. משתנים גלובליים:

```
/** Globals - manages the compilation process */
static Buffer buffer;
static SymbolTable symbolTable;
static map<string, Function> functionTable;
static int currentReturnType;
static int currentMemOffset = 4;
static int nextFreeIntReg = 2;
static int nextFreeFloatReg = 1;
static bool hasDefCreatedScope = false;
```

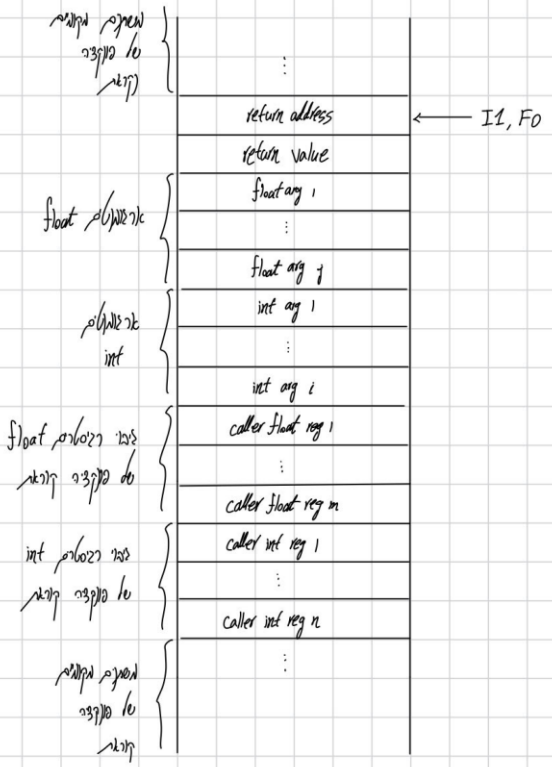
- $buffer$ – שומר את פקודות ה – $Risky$ שהקומפיילר מייצר. בסיום תהליך הקומפילציה תוכן ה - $buffer$ מודפס לקובץ חדש.
- $functionTable$ – טבלת הפונקציות של הקומפיילר שבה נמצאות כל הפונקציות שבתוכנית. המפתח הוא שם הפונקציה והערך הוא מאפייני הפונקציה ששמרנו.
- $currentReturnType$ – טיפוס ערך החזרה של הפונקציה שמתקמפלת כעת. משמש לבדיקה שערך הפונקציה המוחזר בפועל זהה לערך הפונקציה המוצהר.
- $currentMemOffset$ – ההיסט מ - $stack\ pointer$ שבו נמצא התא הפנוי הבא במחסנית
- $nextFreeIntReg$ – המספר של הרגיסטר לערכים שלמים הבא שפנוי.
- $nextFreeFloatReg$ – המספר של הרגיסטר לערכים שבורים הבא שפנוי.
- $hasCreatedScope$ – דגל שמשמש לחלוקת עבודה בהקצאת $scope$ חדש בטבלת הסמלים. ה - $scope$ הראשון נוצר על ידי הגדרת הפונקציה כיוון שהיא כבר מקצה בטבלת הסמלים את הארגומנטים, בפרט עוד לפני שהקומפיילר מגיע לבלוק של ההגדרה. שאר ה - $scopes$ מוקצים על ידי בלוקים.

תיאור פריסות הקוד עבור פקודות בקרה:



בנוסף, בסיום קימפול התוכנית, בפעולות הסמנטיות של סימבול ההתחלה, אנו מבצעים $backpatching$ נוסף עבור קריאות לפונקציות שהוגדרו לאחר שנקראו ולכן בזמן הידור הקריאה הכתובת אליה יש לקבוץ כדי לבצע את הפונקציה עדיין לא הייתה ידועה.

מבנה רשומת ההפעלה:



תפקיד הפונקציה הקוראת בקריאה:

- שמירת כל הרגיסטרים השמורים בזיכרון באופן רציף.
- שמירת כל הארגומנטים לפונקציה הנקראת על פי הסדר שבחתימת הפונקציה.

- שמירת מקום 4 בתים בזיכרון לערך החזרה של הפונקציה הנקראת ו – 4 בתים לכתובת החזרה לפונקציה הקוראת.
- עדכון ה - $frame\ pointer$ להצביע על כתובת הזיכרון שבה תאוחסן כתובת החזרה לפונקציה הקוראת.
- קריאה לפונקציה בעזרת פקודת $JLINK$(כתובת החזרה נמצאת ברגיסטר I_0).

<u>תפקיד הפונקציה הנקראת בקריאה</u>:

- שמירת כתובת החזרה לפונקציה הקוראת מרגיסטר I_0 לכתובת הזיכרון שעליה מצביע ה - $frame\ pointer$.
- הוספת הארגומנטים לטבלת הסמלים של הפונקציה הנקראת, ל - $scope$ החיצוני ביותר כאשר הארגומנטים נמצאים ב - $frame\ pointer - 8$ ומטה.
- ביצוע פעולות הפונקציה.

<u>תפקיד הפונקציה הנקראת בחזרה</u>:

- שמירת ערך החזרה (אם קיים) בכתובת $frame\ pointer - 4$.
- חזרה לפונקציה הקוראת על ידי שחזור כתובת החזרה מכתובת הזיכרון עליה מצביע ה - $frame\ pointer$.

<u>תפקיד הפונקציה הקוראת בחזרה</u>:

- שחזור הרגיסטרים השמורים בזיכרון.
- חילוץ ערך החזרה (אם קיים).
- שחזור ה - $frame\ pointer$ להצביע על הכתובת בזיכרון בה שמורה כתובת החזרה לפונקציה הקוראת לפונקציה הקוראת.

<u>אופן הקצאת רגיסטרים שמורים:</u>

- I_0 – מקבל את כתובת החזרה מפונקציה לאחר ביצוע פקודת $JLINK$.
- I_1 - $integer\ frame\ pointer$. מצביע לכתובת החזרה לפונקציה הקוראת לפונקציה הנוכחית ומשמש להקצאת רגיסטרים זמניים מטיפוס int.
- F_0 - $float\ frame\ pointer$. מצביע לכתובת החזרה לפונקציה הקוראת לפונקציה הנוכחית ומשמש להקצאת רגיסטרים זמניים מטיפוס $float$.

<u>המודולים בקוד הקומפיילר:</u>

- $lexer\_part3.lex$ – המנתח הלקסיקלי שנכתב בחלק הראשון של הפרוייקט
- $bison\_part3.ypp$ – הקובץ הראשי של הקומפיילר. מכיל את המנתח התחבירי והסמנטי.
- $part3\_header.hpp$ – מכיל את הממשקים של המחלקות שבהן השתמשנו ואת ההצהרות והאתחולים של המשתנים הגלובליים בהם השתמשנו.
- $part3\_implementation.cpp$ – מכיל את המימוש של המחלקות שבהן השתמשנו.