תרגיל בית 3 – שפות תכנות

חלק 1- תיאורטי

חלק א

1.1.שיטת הקידוד שמוצגת בקובץ זה היא UTF (**Unicode Transformation Format**)

שיטת קידוד זו מאפשרת לקודד תווים באורך משתנה לunicode. ניתן לקודד כל תו המצוי בתקן unicode ע"י שימוש ב1 עד 4 בתים (תלוי בתו). שיטת קידוד זו מאפשרת הרחבה לקידוד הASCII (ואף תמיכה לאחור בה, שכידוע מוצגים ע"י בית בודד) כך שניתן לראות תווים נוספים כגון א"ב עברי, סימונים מתמטים, לטינית ועוד..

בנוסף שיטת קידוד זו חוסכת בזכרון ביחס לשיטות קידוד אחרות, עמידה בפני איבוד או השחטת מידע,טיפול בקלט לא תקין.

ההתייחסות לתו נעשית ע"י חוקי הקידוד הבאים:

- הסיבית המשמעותית ביותר בבית המייצג תו אחד, היא תמיד 0.
- הסיבית המשמעותית ביותר בבית שהוא חלק ממספר בתים המייצגים תו אחד, היא תמיד 1.
- כל בית הפותח רצף של מספר בתים המגדירים תו בודד, מתחיל ברצף של 1 כמספר הבתים שמייצגים את התו

ואחריהם 0. אם התו ייוצג על ידי שני בתים, יתחיל הבית ב-110, אם ייוצג התו על ידי שלושה בתים, יתחיל הבית ב-

1110 ואם ייוצג התו על ידי ארבעה בתים, יתחיל הבית ב-11110.

- כל בית שהוא בית נוסף (לא הראשון) ברצף של בתים המייצגים תו בודד מתחיל ב-10.

כאשר התווים מחולקים לטווחים מספריים הנמצאים "במרחבים רב לשוניים שונים" כך שהתווים השימושיים יותר ימצאו בטווחים הנמוכים והפחות שימושיים ימצאו בטווחים הגבוהים(וידרשו מס' בתים גדול יותר).

1.2 : שיטת הקידוד היא EBCDIC שיטת קידוד לאותיות ומספרים שהומצאה ע"י IBM בת 8 ביטים לתו (אם זאת לא נעשה שימוש בכל 256 האפשרויות).

:1.3

```
#!/bin/awk -f

BEGIN {

# change the record separator from newline to %

        RS="%";

# convert table

        convert="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

```
# question number
        question=1;
# answer set to A
        answer=1;
# last important line start character
        last="\\";
# mapping from EBCDIC to ASCII
        map["*"]="\\"
        map["+"]="N"
        map["-"]="`"
        }
{
        if ( substr($1,1,1) == map["*"]){
                if ( last == map["+"] || last == map["-"] ){
                        answer=1;
                        question++;
                }
                last=map["*"];
        }
        if ( substr($1,1,1) == map["-"]){
                answer++;
                last=map["-"];
        }
        if ( substr($1,1,1) == map["+"]){
                print question " " substr(convert,answer,1);
                last=map["+"];
        }
}
```

.2

```
#!/bin/awk -f

#taxi cab number

BEGIN{

outer=0;

inner=0;

                for(i=3;;i++){

                outer++;

                        for(j=1;j<i;j++){

                        inner++;

                                array_key=(i*i*i)+(j*j*j);

                                array[array_key]++;

                                if(array[array_key] > 1){

                                        print "the taxicab number is " array_key;

                                        print "total loops count " (outer+inner) " Number of IF's "
inner;

                                        exit;

                                        }

                        }

                }

}
```

פלט:

```
[aviadshiber@t2 hw3]$ ./taxi.awk
the taxicab number is 1729
total loops count 65 Number of IF's 55
```

חלק ב

3. הטיפוס NONE אינו מופיע מפורשות, מופיע בשפה המילה השמורהNone והוא מיועד לסמן **ערך** של משתנה לא מאותחל (כיוון שאין null בשפה), אך זהו ערך ולא טיפוס.
4. סיווג שפת NIM:

קיימות טיפוסית: קיימת טיפוסיות נוקשה(למשל ע"מ להתייחס לenum כמספר יש להשתמש ב ord ולא נעשית המרה אוטומתית כי הם טיפוסים שונים ממש-בניגוד לC), סטטית – כלומר השפה בטוחה.

רמת "תחכום" היא בנאים מתקדמים מאוד : subranges , מערך אסוציאטיבי, אוביקטים, tuples השפה מכילה פולימורפיזמים מכל הצורות.

מנגנון אכיפה מבני בזמן קומפלציה(ע"י בדיקת טיפוסים ,מיקומים,וכתיבת פרגמות) וריצה(ע"י חריגות ושגיאות פאטליות). אחריות הגדרת הטיפוס מונחת על המתכנת אך יש מקרים בהם הקומפיילר מבצע הנחות לגבי הגדרת הטיפוס כאשר משתמשים במילה auto או כשמשתמשים בקבועים, או שימוש בvar עם השמה לליטרלים.

נים היא שפה לא אורטוגנולית (מפלה), לא ניתן ליצור מערכים של פונקציות, אי אפשר לעשות nested open arrays .

נים משתמשת בשקילות מבנית, אם כי במקרים מסוימים (כמו לאוייבקטים, enums , distinct type) משתמשת בשקילות שמית.

גמישות השפה מתבטאת ע"י שימוש בטיפוסים גנרים,טמפלטים, העמסה,דריסה,מקרו,ושימוש במונחה עצמים.

5. סוגי הפולימורפיזם השונים הקיימים ב-NIM הם:

העמסת אופרטורים- לדוגמא האופרטור "+" יכול לחבר בין שני מספרים מסוג int או שני מספרים מסוג double.

העמסת פונקציות- יכולות להיות קיימות 2 פונקציות בעלות אותו השם, למשל func שיקבלו פרמטרים שונים ויבצעו פעולות שונות בהתאם.

תבניות- ניתן לכתוב תבנית אשר תקבל משתנים מכל סוג

פולימורפיזם דקדוקי- יכול להיות משתנה בשם x ופונקציה בשם x והקומפיילר ידע להבדיל ביניהם לפי ההקשר.

פולימורפיזם של ירושה- התייחסות לאוביקט בצורות שונות.

6. Union בשפת C אינה disjoint union מכיוון שמאפשרת type punning, כלומר כל טיפוס במבנה זה למעשה הוא אותו בלוק בזכרון וניתן להשתמש בטיפוסים השונים במקביל וכך ליצור הפרעות טיפוס בלי כוונה ובלי אפשרות לאתר זאת בזמן קומפלציה(מכיוון שאין טיוג למשתנה).
7.

```
type
 eTERMTYPE= enum COMPOUND,ATOM,NUMBER,VARIABLE
 TermType = ref TermObj
 TermObj =object
  case kind_term: eTERMTYPE
  of COMPOUND: compundval : CompoundType
  of NUMBER: numberval : NumberType
  of VARIABLE: varval: string
  of ATOM: atomval: AtomType
```

```
AtomType=distinct string
CompoundType = tuple[atom:AtomType,term:TermType]
eTERMSTYPE=enum none,many
TermsType=ref TermsObj
TermsObj= object
 case kind_terms:eTERMSTYPE
 of none : nil
 of many : manyval:tuple[first:TermType,rest:TermsType]
eNUMBERTYPE = enum INT,REAL
NumberType=ref NumberObj
NumberObj= object
 case kind_num:eNUMBERTYPE
  of INT: intval:int
  of REAL: flaotval:float
```

.8

```
type
  myEnum = enum
    # How many bits for storing car/cdr kind:
    KIND_SIZE = 2,
    # How many bits for index into pool:
    LG2_POOLSIZE = 14

# Will be used for atoms:
type
  charArray = array[0..(1 shl ord(LG2_POOLSIZE)), char]
var atoms: charArray

type
  kind = enum
    NIL, STRING, INTEGER, CONS

type
  cons = object
```

```
  carKind {.bitsize:ord(KIND_SIZE).}: kind
  car {.bitsize:ord(LG2_POOLSIZE).}: cuint
  cdrKind {.bitsize:ord(KIND_SIZE).}: kind
  cdr {.bitsize:ord(LG2_POOLSIZE).}: cuint


# Pool of struct Cons nodes:
type
 poolArray = array[0..(1 shl ord(LG2_POOLSIZE)), cons]
var pool: poolArray
```

9.השאלות שיצאו לנו 2,7:

(2

```
Object parent=new String();
String child=parent;
```

זו אינה שגיאת טיפוס, אך הקומפילר לא יאפשר לקוד זה לעבור קומפלציה בשפת JAVA כיוון שparent הוא מטיפוס Object ויתכן ויכיל טיפוס שהוא לא String. כלומר באמצעות Static typing השפה מונעת שגיאות אפשריות. ע"מ בכל זאת להגיד לקומפיילר שזה אפשרי יש להשתמש בconversion ע"מ להצהיר לקומפילר "אנו יודעים שזו לא שגיאה תאפשר את זה". בדיקה סטטית זו "עוקפת" את תאוריית רייס כיוון שהיא נותנת כלי נוסף למציאת שגיאות אפשריות.

או (דוגמה חזקה יותר:)

קריאה לפרוצדורה שמחזירה ערך ללא שימוש בו בשפת נימרוד, היא ללא שגיאות טיפוס אך יפסל ע"י static typing ללא discard (בנמרוד פונקציות שמחזירות ערך חייב להיות להם שימוש):

```
proc f(a:int):int{
result=1;
}
f(2);
```

7)הפרת void safety נחשבת כtype error מכיוון שכאשר שפה מאפשרת לגשת לישות שאינה קיימת אין לכך שום משמעות, כיוון ש null אינו ישות. ברוב השפות שמפרים void safety נזרק חריגה בזמן ריצה (לדוגמה גאווה).