PROGRAMING LANGUAGES

Proje-2

ÖĞRENCİ NO – AD SOYAD

05180000075 – EMİRCAN BAHADIR

05180000057 – AHMET FURKAN ERDEM

05180000744 – RAGIP BURAK GENÇ

İçindekiler

1. Analiz

BigAdd programlama dili için Interpreter yazmamız isteniyor.

Bu Interpreter, myscript.ba ve yourscript.ba isimli dosyalarından adı girileni alıp, içindeki lexemeleri tanıyıp türlerine göre inceleyip, array oluşturuyor. Oluşturulan array'in gerekli kontrollerini yapıp çıktısını oluşturuyor.

Kontrollerimizi token array için loop kullanarak gerçekleştirdik. Döngümüzü parser mantığında kurduk. Her loop'a girdiğinde bir satırı kontrol ediyor. Bu kontrolde satırın ilk elemanına göre gelmesi gereken patern'in oluşup oluşmadığını ve atama, ekleme gibi işlemlerde değişken verildiyse bu değişkenin oluşturulup oluşturulmadığını kontrol ediyor.

Kodun çalışması için parser mantığında token array kullandığımız için parse tree kullanımına ihtiyaç duymadık.

2. Tasarım

Kaynak Kod:

#include <stdio.h>

```
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdbool.h> //C99 standardı olması şart!

#define MAX_DIGIT 100
#define INITIAL_MAX 10

#ifndef max
#define max(a, b) ((a) > (b) ? (a) : (b)) //büyük olanı gönderir
#endif

struct token { //tanımlama
   char *type; //identifier, keyword, integer, parenthesis, eol, comma, string
   char *value;
   int line;
   int column;
};

struct symbol {
   char *name;
   char *value;
};

struct stack {
   int max;
   char *elements;
```

```
    int top;
};

char pop(struct stack *st);

void push(struct stack *st, char c);

char *add(char *a, char *b);

char *sub(char *a, char *b);

//stringleri n karakter kaydırır
//n=-1 bir karakter sola, n=2 iki karakter sağa kaydırır
char *shiftstr(char *str, int n);

int error(int type, char *info, struct token t);

int stop();

char *valueof(struct token target); //tokenin decimal halini döndürür, token identifier ya da integer olabilir
char *get(char *target_name); //identifier'ın değerini çağırır
int set(char *target_name, char *new_val); //identifier'ın değerini girer
int compare(char *a, char *b); //karşlaştırır b büyük is -1, a büyük ise 1 döndürür

struct symbol symbol_table[100];
int symbol_count = 0;
```

```
int main(int argc, char *argv[]) {
   char keywords[10][10] = {"int", "move", "add", "to", "sub", "from", "loop", "times", "out",
"newline"};
   int line_count = 1;
   char *filename;
   bool arg_handled = false;

   if (argc <= 1) {
      filename = malloc(100 * sizeof(char));
      *(filename) = '\0';
   } else if (argc == 2)
      filename = argv[1];
   else {
      //programın çalışma şekli her zaman ilk argümandır
      printf("Too many arguments given. Maximum one argument expected.");
      return stop();
   }

   while (true) { //düzgün bir dosya adı girilene kadar döngüden çıkılamaz
      if (argc <= 1 || arg_handled) {
         printf("Enter a file name: ");
         scanf(" %[^\n]s", filename); //dosya adında boşluk kullanımına izin verir
      } else {
         arg_handled = true;
      }
      bool filename_ok = false;
      if (strstr(filename, ".") != NULL) {
         strrev(filename);
```

```
        if (filename[0] == 'a' && filename[1] == 'b' && filename[2] == '.') {
            strrev(filename);
            filename_ok = true;
        } else {
            printf("Error: Wrong extension!\n");
        }
    } else { //dosya adını alırken uzantısını almayıp kendimiz ekliyoruz
        int len = (int) strlen(filename);
        filename[len] = '.';
        filename[len + 1] = 'b';
        filename[len + 2] = 'a';
        filename[len + 3] = '\0';
        filename_ok = 1;
    }
    if (filename_ok) {
        if (access(filename, F_OK) == -1) {
            printf("File doesn't exist\n");
        } else {
            break; //dosya bulunduğunda döngüyü kırar
        }
    }
}

// vize projesi
FILE *source_code = fopen(filename, "r"); //lex-analyzer output
char lexeme[105]; //lexeme için geçici karakter dizisi
struct token tokens[1000];
```

```
int token_count = 0;
int eof_col = 0;//hataları bildirmek için

int i = 0, col = 0, start_column = 0;
bool is_reading_comment = false, is_reading_string = false, is_integer = true;
while (1) {
    char c = (char) fgetc(source_code); //kaynaktan tek karakter okur

    if (c == '{' && !is_reading_comment && !is_reading_string) { //stringdeki parantezleri
okumaması için
        //fputs("{ is a parenthesis\n", output_file);
        is_reading_comment = true;
        continue;
    } else if (c == '}' && is_reading_comment && !is_reading_string) {
        //fputs("} is a parenthesis\n", output_file);
        is_reading_comment = false;
        continue;
    }

    col++;
    if (i == 0) { //lexeme'in ilk karakteri
        start_column = col;
    }
    if (c == '\n') {
        eof_col = col; //hata bildirimi
        col = 0;
        line_count++;
    }
```

```
if (!is_reading_comment) { //commentleri okumamak adına lexeme ve hata kontrolü
    if (c == '"') { //" “ ” aynı değil print(“str”) C'de geçersiz!
        if (is_reading_string) { //string sonu
            lexeme[i] = '\0';
            tokens[token_count].type = "string";
            tokens[token_count].value = strdup(lexeme); //lexeme kopyalar, sonradan
değiştirilecek
            token_count++;
            i = 0;
        }
        is_reading_string = !is_reading_string;
        continue;
    }
    if (is_reading_string) {
        lexeme[i++] = c;
    }

    if (!is_reading_string) {
        if ((c >= 65 && c <= 91) || (c >= 97 && c <= 123) || (c >= 48 && c <= 57) || (c >= 44 && c <=
46)
            || c == 93 || c == 125 || c == 32 || c == 9 || c == 10 || c == 95 || c == -1) {
            //kabul edilen karakterler A-Z a-z 0-9 , - . [] {} space \t \n _ karakterleri

            if (c == '[') {
                tokens[token_count].type = "parenthesis";
                tokens[token_count].value = "[";
                tokens[token_count].line = line_count;
```

```
    tokens[token_count].column = start_column;
    token_count++;
    continue;
} else if (c == ']') {
    tokens[token_count].type = "parenthesis";
    tokens[token_count].value = "]";
    tokens[token_count].line = line_count;
    tokens[token_count].column = start_column;
    token_count++;
    continue;
}

if (c != ' ' && c != '.' && c != '\t' && c != '\n' && c != ',' && c != EOF) {
    if ((c < 48 && c != 45) || c > 57) // 45='-'
        is_integer = false;

    lexeme[i++] = c; //lexeme okunmaya devam eder
} else {
    if (i != 0) { //lexeme'in boş olmadığndan emin olur
        lexeme[i] = '\0';
        bool is_keyword = false;
        for (int j = 0; j < 10; j++) { //lexeme anahtar kelime mi kontrolünü yapan döngü
            if (strcmp(lexeme, keywords[j]) == 0) { //string eşitliği
                tokens[token_count].type = "keyword";
                tokens[token_count].value = strdup(lexeme);
                tokens[token_count].line = line_count;
                tokens[token_count].column = start_column;
```

```
            token_count++;
            is_keyword = true;
            break;
        }
    }

    if (!is_keyword) {
        if (is_integer) {
            tokens[token_count].type = "integer";
            tokens[token_count].value = strdup(lexeme);
            tokens[token_count].line = line_count;
            tokens[token_count].column = start_column;
            token_count++;
        } else {
            tokens[token_count].type = "identifier";
            tokens[token_count].value = strdup(lexeme);
            tokens[token_count].line = line_count;
            tokens[token_count].column = start_column;
            token_count++;
        }
    }
}
if (c == '.') {
    tokens[token_count].type = "eol";
    tokens[token_count].value = "."; //çıktı amaçlıdır
    tokens[token_count].line = line_count;
    tokens[token_count].column = start_column;
```

```
                token_count++;
            } else if (c == ',') {
                tokens[token_count].type = "comma";
                tokens[token_count].value = ","; //çıktı amaçlıdır
                tokens[token_count].line = line_count;
                tokens[token_count].column = start_column;
                token_count++;
            }
            i = 0;
            is_integer = true; //reset

        }
      } else {
        printf("Unexpected character: %c in line %d, column %d", c, line_count, col);
        return stop();
      }
    }
  }
  if (c == EOF)
    break;
}
fclose(source_code); //kaynak kod ile işimiz bitti

tokens[token_count].type = "end of file"; //hata yakalama için
tokens[token_count].value = "EOF";
tokens[token_count].line = line_count;
tokens[token_count].column = eof_col;
```

```
//lex-analyzer sayesinde token arrayimiz hazır


struct stack p_stack;
p_stack.max = INITIAL_MAX;
p_stack.elements = NULL;
p_stack.top = -1;
int open_count = 0, close_count = 0, last_open = 0;


//kodun öndoğrulaması
//yanlış girilmiş integerları yakalar
//yanlış identifierları yakalar,  parentez eşler
for (int l = 0; l < token_count; l++) {
   if (strcmp(tokens[l].type, "identifier") == 0) {
      if (strlen(tokens[l].value) > 20)
         return error(7, NULL, tokens[l]); //max identifier uzunluğunu geçmiş mi kontrolü


      if (strstr(tokens[l].value, "-") != NULL) //identifierda "-" var mı kontrolü
         return error(5, NULL, tokens[l]);//geçersiz identifier


      if (tokens[l].value[0] >= 48 && tokens[l].value[0] <= 57 ) //ilk karakter sayı mı kontrolü
         return error(9, NULL, tokens[l]);//geçersiz identifier


   } else if (strcmp(tokens[l].type, "integer") == 0) {
      int digit_limit;
      if (strstr(tokens[l].value, "-") != NULL) //negatif bir int ise
         digit_limit = MAX_DIGIT + 1;
```

```
        else
            digit_limit = MAX_DIGIT;

        if (strlen(tokens[l].value) > digit_limit)
            return error(4, NULL, tokens[l]); //max basamak sayısı geçildi

        int dash_count = 0;
        char *temp = tokens[l].value;
        while (strstr(temp, "-") != NULL) {
            dash_count++;
            temp++;
        }
        if (dash_count > 1)
            return error(6, NULL, tokens[l]); //geçersiz integer

    } else if (strcmp(tokens[l].type, "parenthesis") == 0) {
        if (strcmp(tokens[l].value, "[") == 0) {
            push(&p_stack, '[');
            open_count++;
            last_open = l;
        } else if (strcmp(tokens[l].value, "]") == 0) {
            char temp = pop(&p_stack);
            if (temp != '[')
                return error(1, "Expected open parenthesis before using a close parenthesis ",
tokens[l]);
            close_count++;
        }
    }
```

```
    }
    if (open_count != close_count) {
        printf("Error: Expected a close parenthesis before end of file. Last open parenthesis is on line
%d",
            tokens[last_open].line);
        return stop();
    }

    struct token l_vars[100];
    int l_starts[100] = {0}; //loop başlangıç noktası
    int l_level = -1; //loop seviyesi, -1 loopta değiliz demek
    bool l_block[100] = {false}; //'true' code block var ise, 'false' tek satır kod ise
    //<stdbool.h> bunun için eklendi ^
    i = 0;

    //token array içinde loopa girer, tüm döngü parser mantığındadır
    //her tekrarda bir satırı değerlendirir
    //syntax kontrolü için tokens[i + 1] ya da tokens[i + 2] kullandık
    //örnek olarak 'move 5' okuduktan sonra 'to' var mı diye kontrol eder
    //sonra i'yi 2 artırır çünkü to bir işleme girmez
    while (i < token_count) {
        //döngü ve ilk satır başlar
        //döngü her başladığında ne tip bir satıra başladığımızı kontrol etmeliyiz
        if (strcmp(tokens[i].type, "keyword") == 0 || strcmp(tokens[i].value, "]") == 0) {
            if (strcmp(tokens[i].value, "int") == 0) { //yeni integer tanımlanır -> int x.
                i++;
                if (strcmp(tokens[i].type, "identifier") != 0)
                    return error(1, "Expected an identifier.", tokens[i]);
```

```
            if (strcmp(tokens[i + 1].type, "eol") != 0)
                return error(1, "Expected an end of line character", tokens[i + 1]);

            //declaration syntax'ı doğru
            //get() 'not declared' döndürebilir
            //declare edilmemiş ve 'not declared' döndürmemiş ise hata vardır
            if (strcmp(get(tokens[i].value), "not declared") != 0)
                return error(3, NULL, tokens[i]);

            symbol_table[symbol_count].name = tokens[i].value;
            symbol_table[symbol_count].value = "0";
            symbol_count++;

            i += 2; //satır sonunda yapılacak bir şey yoktur
        } else if (strcmp(tokens[i].value, "move") == 0) { //atama -> y'yi x'e kaydır ya da x'e doğru 5 karakter kaydır
            i++;
            if (strcmp(tokens[i].type, "identifier") != 0 && strcmp(tokens[i].type, "integer") != 0)
                return error(1, "Expected identifier or integer", tokens[i]);

            if (strcmp(tokens[i + 1].value, "to") != 0)
                return error(1, "Expected keyword 'to'", tokens[i + 1]);

            if (strcmp(tokens[i + 2].type, "identifier") != 0)
                return error(1, "Expected an identifier", tokens[i]);
                //yalnızca identifierlara değer verebiliriz
```

```
    if (strcmp(tokens[i + 3].type, "eol") != 0)
        return error(1, "Expected an end of line character", tokens[i + 3]);

    //assignment syntax'ı doğru
    char *new_val = valueof(tokens[i]);
    if (strcmp(new_val, "not declared") == 0)
        return error(2, NULL, tokens[i]);

    int found = set(tokens[i + 2].value, new_val); //sembol bulunamazsa 0 döndürür
    if (!found)
        return error(2, NULL, tokens[i + 2]);

    i += 4; //x'i y'ye kaydır, x'teydik,  "to" atlandı, "y" ve "."
  } else if (strcmp(tokens[i].value, "add") == 0) { //ekleme
    i++;
    if (strcmp(tokens[i].type, "identifier") != 0 && strcmp(tokens[i].type, "integer") != 0)
        return error(1, "Expected identifier or integer", tokens[i]);

    if (strcmp(tokens[i + 1].value, "to") != 0)
        return error(1, "Expected keyword 'to'", tokens[i + 1]);

    if (strcmp(tokens[i + 2].type, "identifier") != 0)
        return error(1, "Expected an identifier", tokens[i + 2]);
        //bir değişkene atamalıyız

    if (strcmp(tokens[i + 3].type, "eol") != 0)
        return error(1, "Expected an end of line character", tokens[i + 3]);
```

```
    //syntax'a ekleme doğru

    char *new_val = valueof(tokens[i]);
    if (strcmp(new_val, "not declared") == 0)
        return error(2, NULL, tokens[i]);

    //hedef kabul edildi, tokens[i + 2] ekleme yapılacak yer
    char *old_val = get(tokens[i + 2].value);
    if (strcmp(old_val, "not declared") == 0)
        return error(2, NULL, tokens[i + 2]);

    char *answer = add(old_val, new_val);
    if (strcmp(answer, "digit limit exceeded") == 0)
        return error(4, NULL, tokens[i + 2]);

    set(tokens[i + 2].value, answer);

    i += 4; //x y'ye eklenir, x'teydik, "to" atlandı, "y" ve "."
  } else if (strcmp(tokens[i].value, "sub") == 0) { //çıkarma
    i++;
    if (strcmp(tokens[i].type, "identifier") != 0 && strcmp(tokens[i].type, "integer") != 0)
        return error(1, "Expected identifier or integer", tokens[i]);

    if (strcmp(tokens[i + 1].value, "from") != 0)
        return error(1, "Expected keyword 'from'", tokens[i + 1]);

    if (strcmp(tokens[i + 2].type, "identifier") != 0)
```

```
        return error(1, "Expected an identifier", tokens[i + 2]);
        //bir değişkene atamamız gerekli

    if (strcmp(tokens[i + 3].type, "eol") != 0)
        return error(1, "Expected an end of line character", tokens[i + 3]);

    char *new_val = valueof(tokens[i]);
    if (strcmp(new_val, "not declared") == 0)
        return error(2, NULL, tokens[i]);

    //hedef kabul edildi, tokens[i + 2] ekleme yapılacak yer
    char *old_val = get(tokens[i + 2].value);
    if (strcmp(old_val, "not declared") == 0)
        return error(2, NULL, tokens[i + 2]);

    char *answer = sub(old_val, new_val);
    if (strcmp(answer, "digit limit exceeded") == 0)
        return error(4, NULL, tokens[i + 2]);

    set(tokens[i + 2].value, answer);

    i += 4; //x y'ye eklenir, x'teydik, "to" atlandı, "y" ve "."
  } else if (strcmp(tokens[i].value, "out") == 0) { //output
    i++;
    while (i < token_count) { //satır sonuna kadar her şeyi yazdır
        if (strcmp(tokens[i].type, "string") == 0) {
            printf(tokens[i].value);
```

```
    } else if (strcmp(tokens[i].type, "identifier") == 0) {
        char *value = valueof(tokens[i]);
        if (strcmp(value, "not declared") == 0)
            return error(2, NULL, tokens[i]);

        printf(value);
    } else if (strcmp(tokens[i].value, "newline") == 0) {
        printf("\n");
    } else //yazdırılabilir değil ise
        return error(1, "Expected string, identifier or 'newline' keyword", tokens[i]);

    i++;

    if (strcmp(tokens[i].type, "eol") == 0)
        break;

    //buraya ulaştıysa yazdırmaya devam eder, vilgül kontrolü yapılır
    if (strcmp(tokens[i].type, "comma") != 0)
        return error(1, "Expected comma or end of line character", tokens[i]);

    i++; //virgül atlanır
  }
  i++; //satır sonunda olduğumuz için loop koşulu kontrolü
} else if (strcmp(tokens[i].value, "loop") == 0) {
  i++;
  if (strcmp(tokens[i].type, "identifier") != 0 && strcmp(tokens[i].type, "integer") != 0)
    return error(1, "Expected identifier or integer", tokens[i]);
```

```
    if (strcmp(tokens[i + 1].value, "times") != 0)
        return error(1, "Expected keyword 'times'", tokens[i + 1]);

    char *loop_count = valueof(tokens[i]);
    if (compare(loop_count, "1") == -1)
        return error(8, NULL, tokens[i]);

    l_level++;
    l_vars[l_level] = tokens[i];

    i += 2; //'tmes anahtar kelimesini geçirir
    if (strcmp(tokens[i].value, "[") == 0) {
        i++; //'[' işleme girmez
        l_block[l_level] = true;
    } else if (strcmp(tokens[i].type, "keyword") != 0)
        return error(1, "Expected open paranthesis or a keyword", tokens[i]);

    l_starts[l_level] = i;
    continue;
}

//bir kod satırı değerlendirildi, döngüye girmiş mi kontrol edilir
if (l_level >= 0) {
    if (l_block[l_level]) { //loop bloğundayız
        if (strcmp(tokens[i].value, "]") == 0) {
            i++; //loopun son tekrarındaysak sonraki satırdan devam eder
```

```
            //this is not used if we keep looping
        } else {
            //şuanki döngüde en az bir satırı değerlendirmiş olucaz
            continue; //o yüzden sonraki satıra geçiyoruz
        }
    }

    char *old_val = valueof(l_vars[l_level]);
    char *new_val = sub(old_val, "1");

    if (strcmp(l_vars[l_level].type, "identifier") == 0) {
        set(l_vars[l_level].value, new_val);
    } else {
        l_vars[l_level].value = new_val;
    }
    if (strcmp(new_val, "0") == 0) {
        l_starts[l_level] = 0;
        l_block[l_level] = false;
        l_level--;
    } else {
        i = l_starts[l_level]; //loopun başına döner
    }
  }
} else {
  //her satır bir anahtar kelimeyle başlamalıdır
  return error(1, "Expected keyword", tokens[i]);
}
```

```
    }


    printf("\n\nInterpreted succesfully! Press a key to exit...");
    fseek(stdin, 0, SEEK_END); //clear input buffer https://stackoverflow.com/a/9750394'a teşekkürler
    getchar();
    return 0;
}


char *add(char *a, char *b) {
    if (strcmp(a, "0") == 0)
        return b;
    else if (strcmp(b, "0") == 0)
        return a;


    bool negative = false;
    if (a[0] == '-' && b[0] != '-') { // -3 + 5 == 5 - 3
        a = shiftstr(a, -1); //bir karakter sola kaydırarak negatiften kurtulunur
        return sub(b, a); //eğer a negatifse, b'den a çıkartılır
    } else if (a[0] != '-' && b[0] == '-') { //3 + (-5) == 3 - 5
        b = shiftstr(b, -1);
        return sub(a, b); //eğer b negatifse, a'dan b çıkartılır
    } else if (a[0] == '-' && b[0] == '-')
        negative = true; //eğer ikisi de negatif ise sonuç da negatiftir


    char result[MAX_DIGIT + 2], x[MAX_DIGIT + 2], y[MAX_DIGIT + 2];  // '-' ve '\0' için max basamak sayısı birer artırılır
```

```
for (int k = strlen(a) - 1; k >= 0 ; k--) {
   x[strlen(a) -1 - k] = *(a + k);
}
x[strlen(a)] = '\0';

for (int k = strlen(b) - 1; k >= 0; k--) {
   y[strlen(b) -1 - k] = *(b + k);
}
y[strlen(b)] = '\0';

char carry = '0';
int i = 0;
bool x_ended = false, y_ended = false;
for (;;) { // while(true)

   if (x[i] > 57 || x[i] < 48) { //sayı olup olmadığı kontrol edilir. x="123", x[3], x[4] sayı değillerdir
      x[i] = 48; //0 atanır
      x[i + 1] = '\0';
      x_ended = true;
   }

   if (y[i] > 57 || y[i] < 48) { //sayı olup olmadığı kontrol edilir
      y[i] = 48;
      y[i + 1] = '\0';
      y_ended = true;
   }
   if (x_ended && y_ended)
```

```
      break;


   // http://www.asciitable.com/
   //  3  + 5 +  1  <= 9
   //  51  + 53  +  49  <= 153
   if (x[i] + y[i] + carry <= 153) {  //9'u geçip geçmediğine bakılır
      result[i] = (char) (x[i] + y[i] + carry - 96); //i'ninci karakter alınır
      carry = '0';
   } else {    //9'u geçerse carry eklenir
      result[i] = (char) (x[i] + y[i] + carry - 106);
      carry = '1';
   }
   i++;
   if (i > MAX_DIGIT)
      return "digit limit exceeded";
}


if (carry == '1') {
   result[i++] = '1';
   if (i > MAX_DIGIT)
      return "digit limit exceeded";
}


result[i] = '\0';
strrev(result);
if (negative) { //sonucun başına '-' eklenir
   for (int j = strlen(result); j > 0; j--) { //sağa doğru bir karakter kaydırır
```

```
            result[j] = result[j - 1];
        }
        result[0] = '-';
        result[i + 1] = '\0'; //stringin sonunu yeniler
    }
    return strdup(result);
}

char *sub(char *a, char *b) {
    if (strcmp(a, b) == 0)
        return "0";

    bool negative = false;
    if (a[0] != '-' && b[0] == '-') { // +x - (-y) == x + y
        b = shiftstr(b, -1); //bir karakter sola kaydırılır, '-''e ihtiyaç yok
        return add(a, b); //"sub - from +" == "add + to +"
    } else if (a[0] == '-' && b[0] != '-') { // -x - x = -x + (-x)
        b = shiftstr(b, 1); // 54 -> 054, '-' için kaydırma
        b[0] = '-';
        return add(a, b); //b negatif ise a'dan b çıkartılır
    } else if (a[0] == '-' && b[0] == '-') {
        if (strlen(a) > strlen(b)) {
            negative = true;
        } else if (strlen(a) == strlen(b) && strcmp(a, b) > 0) { //-5 - (-3) = -2
            negative = true;
        }
    } else if (strlen(a) < strlen(b)) { //3 - 55 = -52
```

```
    negative = true;
} else if (strlen(b) == strlen(a) && strcmp(b, a) > 0) { // 3 - 5 = -2
    //5 - 3 yapılır, sonra '-' eklenir
    negative = true;
    char *temp = b;
    b = a;
    a = temp;
}

char result[MAX_DIGIT + 2], x[MAX_DIGIT + 2], y[MAX_DIGIT + 2];



for (int k = strlen(a) - 1; k >= 0 ; k--) {
    x[strlen(a) -1 - k] = *(a + k);
}
x[strlen(a)] = '\0';
for (int k = strlen(b) - 1; k >= 0; k--) {
    y[strlen(b) -1 - k] = *(b + k);
}
y[strlen(b)] = '\0';



//i şuan bulunduğumuz basamağı temsil eder
int i = 0;
bool x_ended = false, y_ended = false;
while (true) {
    if (x[i] > 57 || x[i] < 48) {  //rakam olup olmadığı kontrol edilir
```

```
    x[i] = '0'; //0 atanır hesaplamaları etkilemez
    x[i + 1] = '\0';
    x_ended = true;
}
if (y[i] > 57 || y[i] < 48) { //rakam olup olmadığı kontrol edilir
    y[i] = '0';
    y[i + 1] = '\0';
    y_ended = true;
}
if (x_ended && y_ended)
    break;

// http://www.asciitable.com/
//   5   -  3   > 0
//  53   - 51   > 0
if (x[i] - y[i] >= 0) {  //9'u geçip geçmediği kontrol edilir
    result[i] = (char) (x[i] - y[i] + 48); // 48 = ascii of '0'
} else {    // 3 - 5?
    int j = i + 1;
    while (x[j] == '0') { //tüm 0'lara 9 atanır, 500000 - 1?
        x[j] = '9';
        j++;
    }
    x[j] = (char) (x[j] - 1);

    //13        - 5       = 8
    //51   + 10 - 53   + 48 = 56
```

```
            result[i] = (char) (x[i] + 10 - y[i] + 48);
        }
        i++;
        if (i > MAX_DIGIT)
            return "digit limit exceeded";
    }

    result[i] = '\0';
    strrev(result);

    if (result[0] == '0' && strlen(result) > 1) { // 0002, 0'ları atıyoruz
        int zero_count = 0;
        for (int j = 0; j < strlen(result); j++) { //gereksiz 0'ları sayar
            if (result[j] == '0')
                zero_count++;
            else
                break;
        }
        for (int j = 0; j < strlen(result); j++) { //kaydırarak 0'lardan kurtulur
            result[j] = result[j + zero_count];
        }
    }
    if (negative) { //sonucun başına '-' ekler
        for (int j = strlen(result); j > 0; j--) { //sağa bir kaydır
            result[j] = result[j - 1];
        }
        result[0] = '-';
```

```
        result[i + 1] = '\0'; //str'nin sonunu yeniler
    }

    return strdup(result);
}

char *shiftstr(char *str, int n) {
    if (n < 0) { // 05 -> 5 if n=-1 | 3456 -> 56 if n=-2
        str = str + abs(n);
        return str;
    } else if (n > 0) { // 4 -> 04 if n=1 | 23 -> 00023 if n=3
        char *new_str = malloc(sizeof(char) * (strlen(str) + n));
        for (int i = 0; i < strlen(str) + n; ++i) {
            if (i < n)
                *(new_str + i) = '0';
            else
                *(new_str + i) = str[i - n];
        }
        *(new_str + strlen(str) + n) = '\0';
        free(str);
        return new_str;
    } else
        return str;
}

int error(int type, char *info, struct token t) {
    system("cls");
```

```
   printf("Error on line %d column %d: ", t.line, t.column);
   if (type == 1) // expected ...
      printf("Unexpected %s '%s'. %s.", t.type, t.value, info);
   else if (type == 2)
      printf("'%s' is not declared before.", t.value);
   else if (type == 3)
      printf("'%s' is already declared before.", t.value);
   else if (type == 4)
      printf("Maximum value of an integer is exceeded.");
   else if (type == 5)
      printf("'%s' is not valid variable name. Only alphanumeric characters and underscores
accepted.", t.value);
   else if (type == 6)
      printf("'%s' is not valid integer.", t.value); //--23
   else if (type == 7)
      printf("Maximum length of an identifier is exceeded.");
   else if (type == 8)
      printf("Loop variable '%s' must be greater than zero.", t.value);
   else if (type == 9)
      printf("'%s' is not valid variable name. It must start with an alphabetic character.", t.value);

   printf("\nPress enter to exit...");
   fseek(stdin, 0, SEEK_END);
   getchar();
   return -1;
}

int stop() {
```

```
    fseek(stdin, 0, SEEK_END);
    getchar();
    return -1;
}

char *valueof(struct token target) { //tokenin değerini bulur, int ya da identifier olabilir
    //eğer identifier ise değerini zaten biliyoruz
    if (strcmp(target.type, "integer") == 0)
        return target.value;
    else
        return get(target.value);
}

char *get(char *target_name) { //identifier'ın değerini çağırır
    //sembol tablosunda arayıp hedefin değerini döndürür
    for (int j = 0; j < symbol_count; ++j) {
        if (strcmp(symbol_table[j].name, target_name) == 0) {
            return symbol_table[j].value;
        }
    }
    return "not declared";
}

int set(char *target_name, char *new_val) { //identifier'ın değerini yerleştirir
    for (int j = 0; j < symbol_count; ++j) {
        if (strcmp(symbol_table[j].name, target_name) == 0) {
            symbol_table[j].value = new_val;
```

```
            return 1; //güncelleme başarılı
        }
    }
    return 0; //hata
}

void push(struct stack *st, char c) {
    if (st->elements == NULL) { //yer açılabilmesını sağlar
        st->elements = malloc(INITIAL_MAX * sizeof(char));
    } else if (st->top == st->max - 1) { //dolu ise yer açmaya devam edilir
        st->max *= 2; //kapasiteyi ikiye katlar
        char *more_elements = realloc(st->elements, st->max * sizeof(char));
        st->elements = more_elements;
    }
    st->top++;
    st->elements[st->top] = c;
}

char pop(struct stack *st) {
    if ((st->max / 2) > (st->top + 10)) { //yarısından fazlası boş
        st->max /= 2;
        char *less_elements = malloc(st->max * sizeof(char));
        free(st->elements);
        st->elements = less_elements;
    }
    if (st->top == -1) {
        //stack boş, null döndürülür
```

```
        return '\0';
    } else {
        char c = st->elements[st->top];
        st->elements[st->top] = '\0'; //debugging amaçlı
        st->top--;
        return c;
    }
}

/* compare(): iki decimal'i karşılaştırmaya yarar
 * returns -1 if a<b
 * returns  0 if a=b
 * returns  1 if a>b
 * */
int compare(char *a, char *b) {
    if (strcmp(a, b) == 0)
        return 0;

    if (a[0] == '-' && b[0] != '-') // -a  +b
        return -1;

    if (a[0] != '-' && b[0] == '-') // +a  -b
        return 1;

    if (a[0] == '-' && b[0] == '-') { // -a  -b
        if (strlen(a) == strlen(b)) { // -aaa -bbb
            if (strcmp(a, b) > 0)   //-5  -4
```

```
            return -1;
        else                //-4  -5
            return 1;
    }

    if (strlen(a) > strlen(b))  // -10  -5
        return -1;
    else
        return 1;

}

//buraya ulaşıldıysa ikisinin de pozitif olduğu doğrulanmıştır
if (strlen(a) == strlen(b)) { //+aaa  +bbb
    if (strcmp(a, b) < 0) // +4  +5
        return -1;
    else              // +5  +4
        return 1;
}

if (strlen(a) < strlen(b))  //+5  +10
    return -1;
else
    return 1;

}
```

3.Programcı Kataloğu

Projedeki Kişi Sayısı: 3 Kişi

Analiz: 1.5 gün

Tasarım: 7 gün

Gerçekleştirme: 4 gün

Test Ve Raporlama: 3 saat

4. Test Verileri Ve Ekran Görüntüleri

Test Verileri:

```
int size.
int sum.
move 5 to size.
loop size times {ignore me , I am a comment}
[ out size , newline.
add size to sum.
]
out newline , "Sum:" , sum.
```

```
int my_Var.
move 25 to my_Var.
int sum.
move 0 to sum.

int t.
move 5 to t.
sub t from my_Var.

out my_Var, newline.

out sum, newline.

loop 10 times [add 5 to my_Var.    ]
loop 10 times [add 1 to my_Var.    ]
out my_Var.
```

Ekran Görüntüleri:

```
Enter a file name: myscript.ba
20
0
80

Interpreted succesfully! Press a key to exit...
```

```
Enter a file name: yourscript.ba
5
4
3
2
1

Sum:15

Interpreted succesfully! Press a key to exit...
```