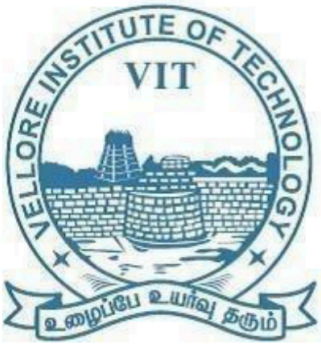

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

B.Tech. Winter Semester 2023-24
School Of Computer Science and Engineering (SCOPE)

Digital Assignment - I

Compiler Design Lab (L31+L32)

Apurva Mishra, 22BCE2791

6 August 2024

1. Question

Problem 1.1.
Write a C/C++ Program to simulate the lexical analysis phase of a compiler using files.

1.1. Code

lexer.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

enum type {
  Identifier,
  Delimiter,
  Operator,
  Keyword,
};

struct item {
  char name[3];
  enum type type;
  int id;
};

char keywords[5][4] = {"int", "else", "for", "fn", "if"};
char operators[2][2] = {"=", "+"};
char delimeters[3][2] = {",", "//", ";"};

int isize = 14;
char *input;
// char input[] = "int a, b = 10;";

char *read_file(const char *filename) {
  FILE *fp = fopen(filename, "rb");
  if (fp == NULL) {
    perror("Error opening file");
    return NULL;
  }

  fseek(fp, 0, SEEK_END);
```

```
  long size = ftell(fp);
  rewind(fp);

  char *buffer = (char *)malloc(size + 1);
  if (buffer == NULL) {
    perror("Memory allocation failed");
    fclose(fp);
    return NULL;
  }

  if (fread(buffer, 1, size, fp) != size) {
    perror("Error reading file");
    free(buffer);
    fclose(fp);
    return NULL;
  }
  isize = size;
  buffer[size] = '\0'; // Null-terminate the string
  fclose(fp);
  return buffer;
}

bool is_operator(char *operator) {
  for (int i = 0; i < 2; i++) {
    int result = strcmp(operator, operators[i]);
    if (result == 0) {
      return true;
    }
  }
  return false;
}

bool is_keyword(char *keyword) {
  for (int i = 0; i < 5; i++) {
    int result = strcmp(keyword, keywords[i]);
    if (result == 0) {
      return true;
    }
  }
  return false;
}

bool is_delimiter(char *delimeter) {
  for (int i = 0; i < 3; i++) {
    int result = strcmp(delimeter, delimeters[i]);
```

```
    if (result == 0) {
      return true;
    }
  }
  return false;
}

// print char in given range for array: input
void print_chars(int start, int end) {
  for (int i = start; i < end; i++) {
    printf("%c", input[i]);
  }
  printf("\n");
}

// remove given index from array: input
void r_index(int index) {
  if (index <= 0) {
    return;
  }
  for (int i = index; i < isize; i++) {
    input[i] = input[i + 1];
  }
}

// remove whitespace from global scopce array: input
// remove \n from global scopce array: input
// returns newsize
void r_wn() {
  for (int i = 0; i < isize; i++) {
    if (input[i] == ' ') {
      r_index(i);
      isize -= 1;
    } else if (input[i] == '\n') {
      r_index(i);
      isize -= 1;
    }
  }
}

void lexer() {
  int token_found = 0;
  int index_prev = 0;

  for (int j = 0; j < isize; j++) {
```

```
    int pointer = 0;
    char token[4] = {'\0'};

    // assume, max token size = 3
    for (int i = 0; i < 3; i++) {

      token[pointer] = input[j + i];
      pointer += 1;

      if (is_keyword(token)) {
        printf("  Keyword: ");
        print_chars(j, j + pointer);
        token_found = 1;
        break;
      } else if (is_delimiter(token)) {
        printf("  Delimiter: ");
        print_chars(j, j + pointer);
        token_found = 1;
        break;
      } else if (is_operator(token)) {
        printf("  Operator: ");
        print_chars(j, j + pointer);
        token_found = 1;
        break;
      }
    }

    if (token_found == 1) {
      token_found = 0;

      if (index_prev != 0) {
        // identifier
        printf("  Identifier: ");
        print_chars(index_prev, j);
      }
      index_prev = j + pointer;

      j += pointer;
    }

    pointer = 0;
    // reset token
    for (int m = 0; m < 3; m++) {
      token[m] = '\0';
    }
```

```
  }
}

int main() {
  input = read_file("input.txt");
  printf("Input:\n  %s\n", input);
  r_wn();
  printf("Sanitized input:\n  %s\n\n", input);
  printf("Tokens: \n");
  lexer();
  return 0;
}
```

1.2. Input/Output

input.txt

```
int a, b = 10;
```

Output

```
$ zig cc  ./src/lexer.c -std=c23 -o ./bin/lexer
$ ./bin/lexer
Input:
  int a, b = 10;

Sanitized input:
  inta,b=10;

Tokens:
  Keyword: int
  Delimiter: ,
  Identifier: a
  Operator: =
  Identifier: b
  Delimiter: ;
  Identifier: 10
```

1.3. Ouput

```
da/ass1/doc via t v0.11.1
❯ just lr
zig cc  ./src/lexer.c -std=c23 -o ./bin/lexer
./bin/lexer
Input:
  int a, b = 10;

Sanitized input:
  inta,b=10;

Tokens:
  Keyword: int
  Delimiter: ,
  Identifier: a
  Operator: =
  Identifier: b
  Delimiter: ;
  Identifier: 10
```

2. Question

Problem 2.1.
Write a C/C++ Program to create symbol table using files.

2.1. Code

symbol_table.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct item {
  int label;
  char symbol[3];
  int address;
  struct item *next;
};

struct item *createItem(int label, char *symbol, int address) {
  struct item *newItem = (struct item *)malloc(sizeof(struct item));
```

```
  newItem->label = label;
  strcpy(newItem->symbol, symbol);
  newItem->address = address;
  newItem->next = NULL;

  return newItem;
}

void insertAtBeginning(struct item **head, int label, char *symbol,
                       int address) {
  struct item *newItem = createItem(label, symbol, address);

  newItem->next = *head;
  *head = newItem;
}

// Function to insert an item at the end of the list
void insertAtEnd(struct item **head, int label, char *symbol, int
address) {

  struct item *newItem = createItem(label, symbol, address);
  struct item *temp = *head;

  if (*head == NULL) {
    *head = newItem;
    return;
  }

  while (temp->next != NULL) {
    temp = temp->next;
  }
  temp->next = newItem;
}

void deleteItem(struct item **head, int label) {
  struct item *temp = *head, *prev = NULL;

  if (temp != NULL && temp->label == label) {
    *head = temp->next;
    free(temp);
    return;
  }

  while (temp != NULL && temp->label != label) {
    prev = temp;
```

```
    temp = temp->next;
  }

  if (temp == NULL) {
    return; // Item not found
  }

  prev->next = temp->next;
  free(temp);
}

void modifyItem(struct item *head, int label, int newLabel, char
*newSymbol,
                int newAddress) {
  struct item *temp = head;

  while (temp != NULL && temp->label != label) {
    temp = temp->next;
  }

  if (temp == NULL) {
    return;
  }

  temp->label = newLabel;
  strcpy(temp->symbol, newSymbol);
  temp->address = newAddress;
}

struct item *searchItem(struct item *head, int label) {
  struct item *temp = head;

  while (temp != NULL && temp->label != label) {
    temp = temp->next;
  }

  return temp;
}

void displayList(struct item *head) {
  struct item *temp = head;

  while (temp != NULL) {
    printf("Label: %d, Symbol: %s, Address: %d\n", temp->label, temp-
>symbol,
```

```
            temp->address);
    temp = temp->next;
  }
}

int main() {
  // singly linked list
  struct item *head = NULL;

  insertAtEnd(&head, 1, "A", 100);
  insertAtBeginning(&head, 2, "B", 200);
  insertAtEnd(&head, 3, "C", 300);

  printf("Original list:\n");
  displayList(head);

  modifyItem(head, 2, 20, "BB", 220);

  deleteItem(&head, 1);

  struct item *foundItem = searchItem(head, 3);
  if (foundItem != NULL) {
    printf("\nItem found: Label: %d, Symbol: %s, Address: %d\n",
           foundItem->label, foundItem->symbol, foundItem->address);
  } else {
    printf("\nItem not found\n");
  }

  printf("\nModified list:\n");
  displayList(head);

  return 0;
}
```

2.2. Input/Output

Output

```
$ zig cc  ./src/table.c -std=c23 -o ./bin/table
$ ./bin/table
Original list:
Label: 2, Symbol: B, Address: 200
Label: 1, Symbol: A, Address: 100
Label: 3, Symbol: C, Address: 300
```

```
Item found: Label: 3, Symbol: C, Address: 300

Modified list:
Label: 20, Symbol: BB, Address: 220
Label: 3, Symbol: C, Address: 300
```

2.3. Output