FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

System Programming and Compiler Construction

VI Semester (Computer) **Academic Year: 23 - 24**

Experiment No 1

- *Ivan Dsouza*
- *9601*
- *T.E. Comps A (Batch C)*

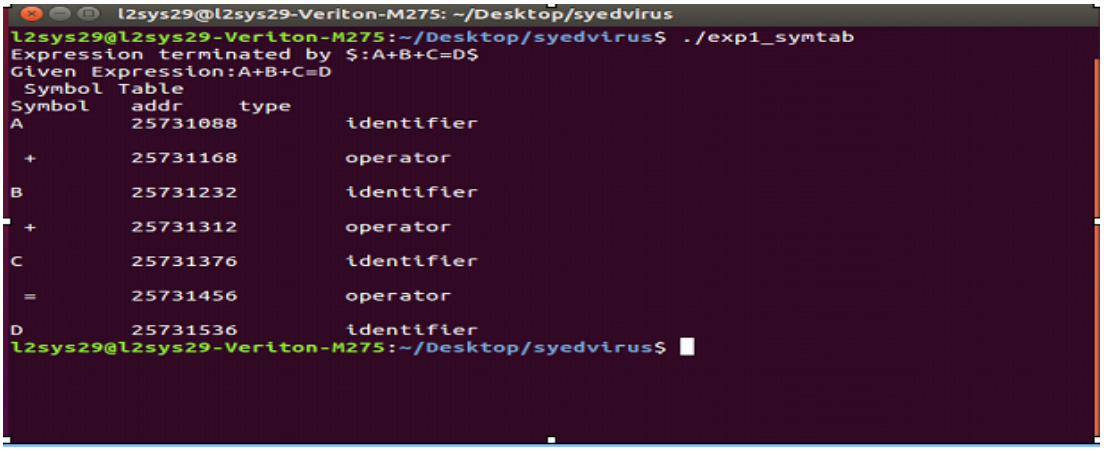
AIM:

To write a program for implementing Symbol Table.

ALGORITHM

Step1: Start the program for performing insert, display, delete, search and modify option in symbol table
Step2: Define the structure of the Symbol Table
Step3: Enter the choice for performing the operations in the symbol Table
Step4: If the entered choice is 1, search the symbol table for the symbol to be inserted. If the symbol is
already present, it displays “Duplicate Symbol”. Else, insert the symbol and the corresponding address in
the symbol table.
Step5: If the entered choice is 2, the symbols present in the symbol table are displayed.
Step6: If the entered choice is 3, the symbol to be deleted is searched in the symbol table.
Step7: If it is not found in the symbol table it displays “Label Not found”. Else, the symbol is deleted.
Step8: If the entered choice is 5, the symbol to be modified is searched in the symbol table.

Sample Input and Output:



FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

System Programming and Compiler Construction

VI Semester (Computer) **Academic Year: 23 - 24**

Postlab Questions:

1. Explain different phases of compiler. Illustrate all the output after each phase for the following statement
   a= b +c -d *5

Code:

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

struct SymbolEntry
{
   char symbol;
   void *address;
   char type[40];
};

void insert(struct SymbolEntry symbolTable[], int *symbolCount, char newSymbol)
{
   for (int i = 0; i < *symbolCount; i++)
   {
      if (symbolTable[i].symbol == newSymbol)
      {
         printf("Duplicate Symbol. Cannot insert.\n");
         return;
      }
   }

   symbolTable[*symbolCount].address = malloc(sizeof(int));
   strcpy(symbolTable[*symbolCount].type, isalpha(newSymbol) ? "identifier" : "operator");
   symbolTable[*symbolCount].symbol = newSymbol;
   (*symbolCount)++;
   printf("Symbol inserted successfully.\n");
}

void display(const struct SymbolEntry symbolTable[], int symbolCount)
{
   printf("\nSymbol Table\nSymbol \t Address \t Type\n");
   for (int i = 0; i < symbolCount; i++)
      printf("%c \t %p \t %s\n", symbolTable[i].symbol, symbolTable[i].address, symbolTable[i].type);
```

```
}

void delete(struct SymbolEntry symbolTable[], int *symbolCount, char symbolToDelete)
{
   int foundIndex = -1;
   for (int i = 0; i < *symbolCount; i++)
   {
      if (symbolTable[i].symbol == symbolToDelete)
      {
         foundIndex = i;
         break;
      }
   }

   if (foundIndex == -1)
   {
      printf("Symbol not found. Cannot delete.\n");
      return;
   }

   free(symbolTable[foundIndex].address);

   for (int i = foundIndex; i < *symbolCount - 1; i++)
      symbolTable[i] = symbolTable[i + 1];

   (*symbolCount)--;
   printf("Symbol deleted successfully.\n");
}

void search(const struct SymbolEntry symbolTable[], int symbolCount, char symbolToSearch)
{
   int foundIndex = -1;
   for (int i = 0; i < symbolCount; i++)
   {
      if (symbolTable[i].symbol == symbolToSearch)
      {
         foundIndex = i;
         break;
      }
   }

   if (foundIndex == -1)
   {
```

```
    printf("Symbol not found. Cannot display.\n");
    return;
  }

  printf("\nSymbol Information\nSymbol: %c\nAddress: %p\nType: %s\n",
      symbolTable[foundIndex].symbol, symbolTable[foundIndex].address,
symbolTable[foundIndex].type);
}

void modify(struct SymbolEntry symbolTable[], int symbolCount, char symbolToModify)
{
  int foundIndex = -1;
  for (int i = 0; i < symbolCount; i++)
  {
    if (symbolTable[i].symbol == symbolToModify)
    {
      foundIndex = i;
      break;
    }
  }

  if (foundIndex == -1)
  {
    printf("Symbol not found. Cannot modify.\n");
    return;
  }

  printf("Symbol found. Modifying symbol type.\n");
  strcpy(symbolTable[foundIndex].type, "modified");
  printf("Symbol modified. New type: %s\n", symbolTable[foundIndex].type);
}

int main()
{
  struct SymbolEntry symbolTable[50];
  int symbolCount = 0, choice;
  char newSymbol;

  do
  {
    printf("\nSymbol Table Operations\n1. Insert Symbol\n2. Display Symbol Table\n3. Delete
Symbol\n4. Search Symbol\n5. Modify Symbol\n6. Exit\nEnter your choice: ");
    scanf("%d", &choice);
```

```
    switch (choice)
    {
    case 1:
      printf("Enter symbol to insert: ");
      scanf(" %c", &newSymbol);
      insert(symbolTable, &symbolCount, newSymbol);
      break;
    case 2:
      display(symbolTable, symbolCount);
      break;
    case 3:
      printf("Enter symbol to delete: ");
      scanf(" %c", &newSymbol);
      delete(symbolTable, &symbolCount, newSymbol);
      break;
    case 4:
      printf("Enter symbol to search: ");
      scanf(" %c", &newSymbol);
      search(symbolTable, symbolCount, newSymbol);
      break;
    case 5:
      printf("Enter symbol to modify: ");
      scanf(" %c", &newSymbol);
      modify(symbolTable, symbolCount, newSymbol);
      break;
    case 6:
      for (int i = 0; i < symbolCount; i++)
        free(symbolTable[i].address);

      printf("Exiting the program.\n");
      break;
    default:
      printf("Invalid choice. Please try again.\n");
    }
  } while (choice != 6);

  return 0;
}
```

Output:

```
PS C:\Users\ivana\Desktop\College\Third Year\SEM 6\SPCC Pracs> cd "c:\Users\ivana\Desktop\College\Third Year\SEM 6\SPCC Pracs\"

Symbol Table Operations
1. Insert Symbol
2. Display Symbol Table
3. Delete Symbol
4. Search Symbol
5. Modify Symbol
6. Exit
Enter your choice: 1
Enter symbol to insert: A+B+C=D$
Symbol inserted successfully.

Symbol Table Operations
1. Insert Symbol
2. Display Symbol Table
3. Delete Symbol
4. Search Symbol
5. Modify Symbol
6. Exit
Enter your choice: Enter symbol to insert: Symbol inserted successfully.

Symbol Table Operations
1. Insert Symbol
2. Display Symbol Table
3. Delete Symbol
4. Search Symbol
5. Modify Symbol
6. Exit
Enter your choice: Enter symbol to insert: Symbol inserted successfully.

Symbol Table Operations
1. Insert Symbol
2. Display Symbol Table
3. Delete Symbol
4. Search Symbol
5. Modify Symbol
6. Exit
Enter your choice: Enter symbol to insert: Symbol inserted successfully.

Symbol Table Operations
1. Insert Symbol
2. Display Symbol Table
3. Delete Symbol
4. Search Symbol
5. Modify Symbol
```

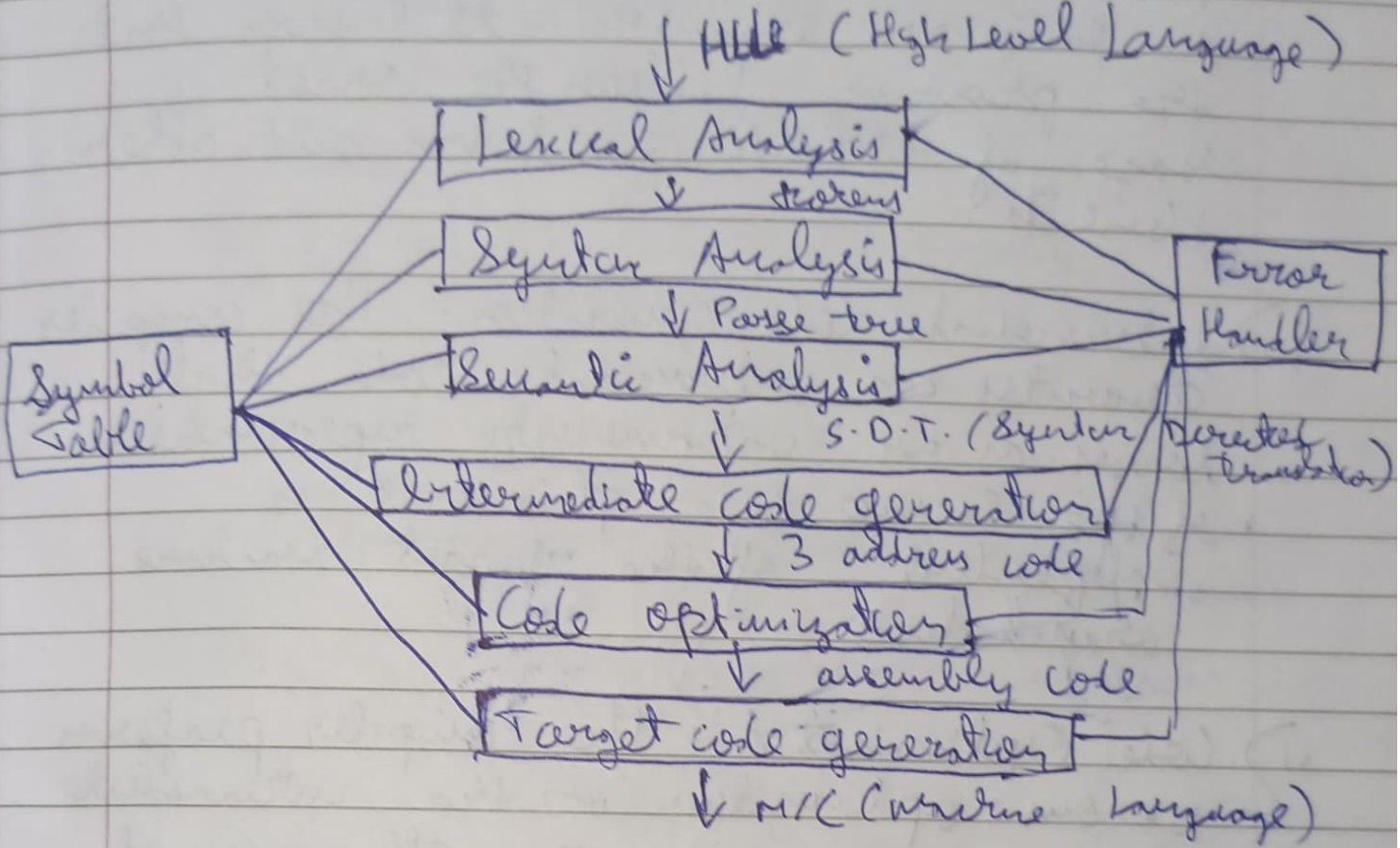
```
Symbol Table Operations
1. Insert Symbol
2. Display Symbol Table
3. Delete Symbol
4. Search Symbol
5. Modify Symbol
6. Exit
Enter your choice: Enter symbol to insert: Symbol inserted successfully.

Symbol Table Operations
1. Insert Symbol
2. Display Symbol Table
3. Delete Symbol
4. Search Symbol
5. Modify Symbol
6. Exit
Enter your choice: Enter symbol to insert: Symbol inserted successfully.

Symbol Table Operations
1. Insert Symbol
2. Display Symbol Table
3. Delete Symbol
4. Search Symbol
5. Modify Symbol
6. Exit
Enter your choice:
2

Symbol Table
Symbol   Address         Type
A        0000017b12b61450        identifier
B        0000017b12b61470        identifier
C        0000017b12b61490        identifier
=        0000017b12b614b0        operator
D        0000017b12b614d0        identifier
$        0000017b12b614f0        operator

Symbol Table Operations
1. Insert Symbol
2. Display Symbol Table
3. Delete Symbol
4. Search Symbol
5. Modify Symbol
6. Exit
Enter your choice: 6
Exiting the program.
PS C:\Users\ivana\Desktop\College\Third Year\SEM 6\SPCC Pracs>
```

Postlab:

IVAN DSOUZA
9601
T.E. COMPS A

Page No.
Date

a1) Different phases of a compiler are as follows:



i) Lexical Analysis: this phase involves breaking the source code into tokens which are the smallest units like Keywords, identifiers and operators.

ii) Syntax Analysis: In this phase the compiler checks the syntax of source code against the grammar rule of the programming language. It creates a parse tree representing syntactic structure of program.

iii) Semantic Analysis: The compiler checks the meaning of the program in terms of the language semantics. It ensures that the program follows the correct usage of variables, functions and other elements.

iv) Intermediate Code Generation: The compiler generates an intermediate code that serves as an intermediate representation of the source code. This code is independent of the target machine architecture.

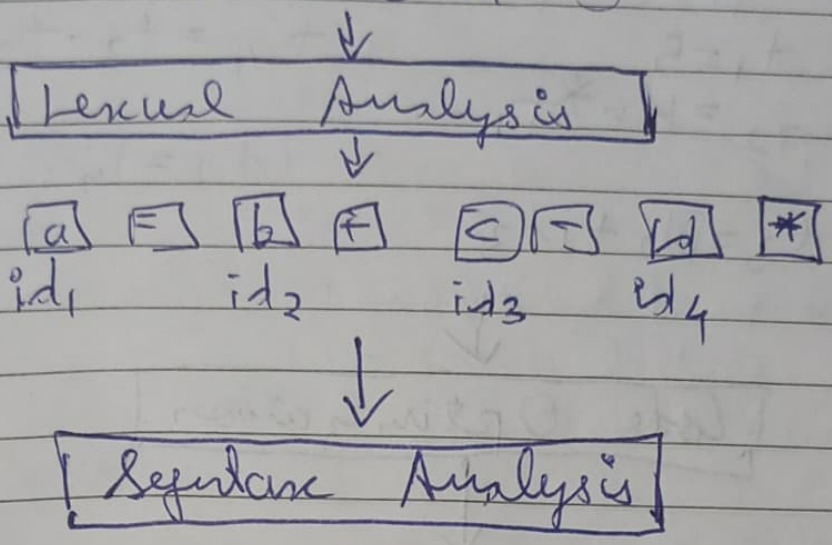
v) Code Optimization: The compiler performs various optimizations on the intermediate code to improve the efficiency of the generated machine code. This includes removing redundancies and improving it.

vi) Code Generation: This is final phase of compile this phase takes the optimized intermediate code and generates the actual machine code that can be executed by hardware.

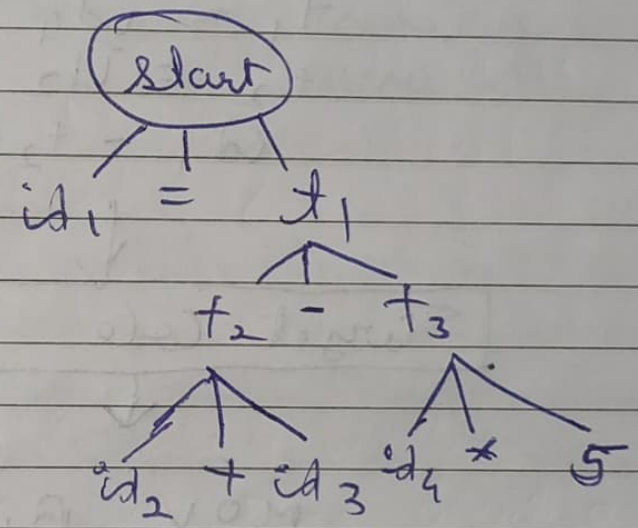
Page No.
Date

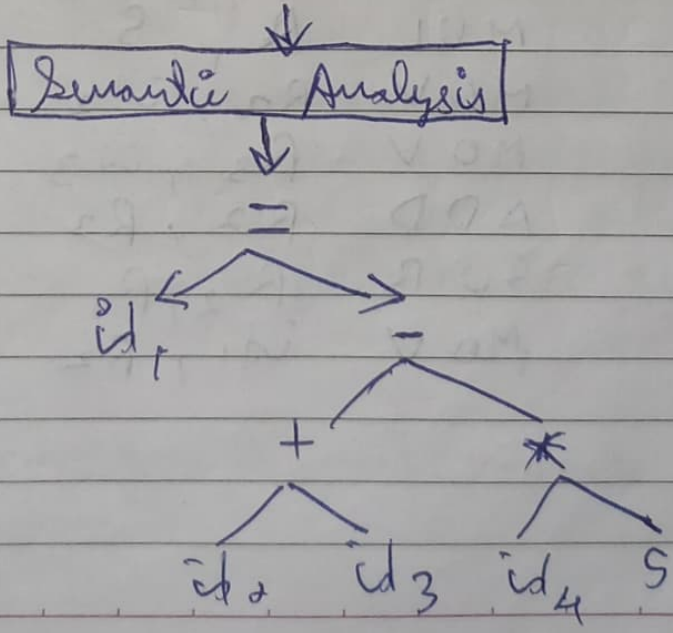
Example:

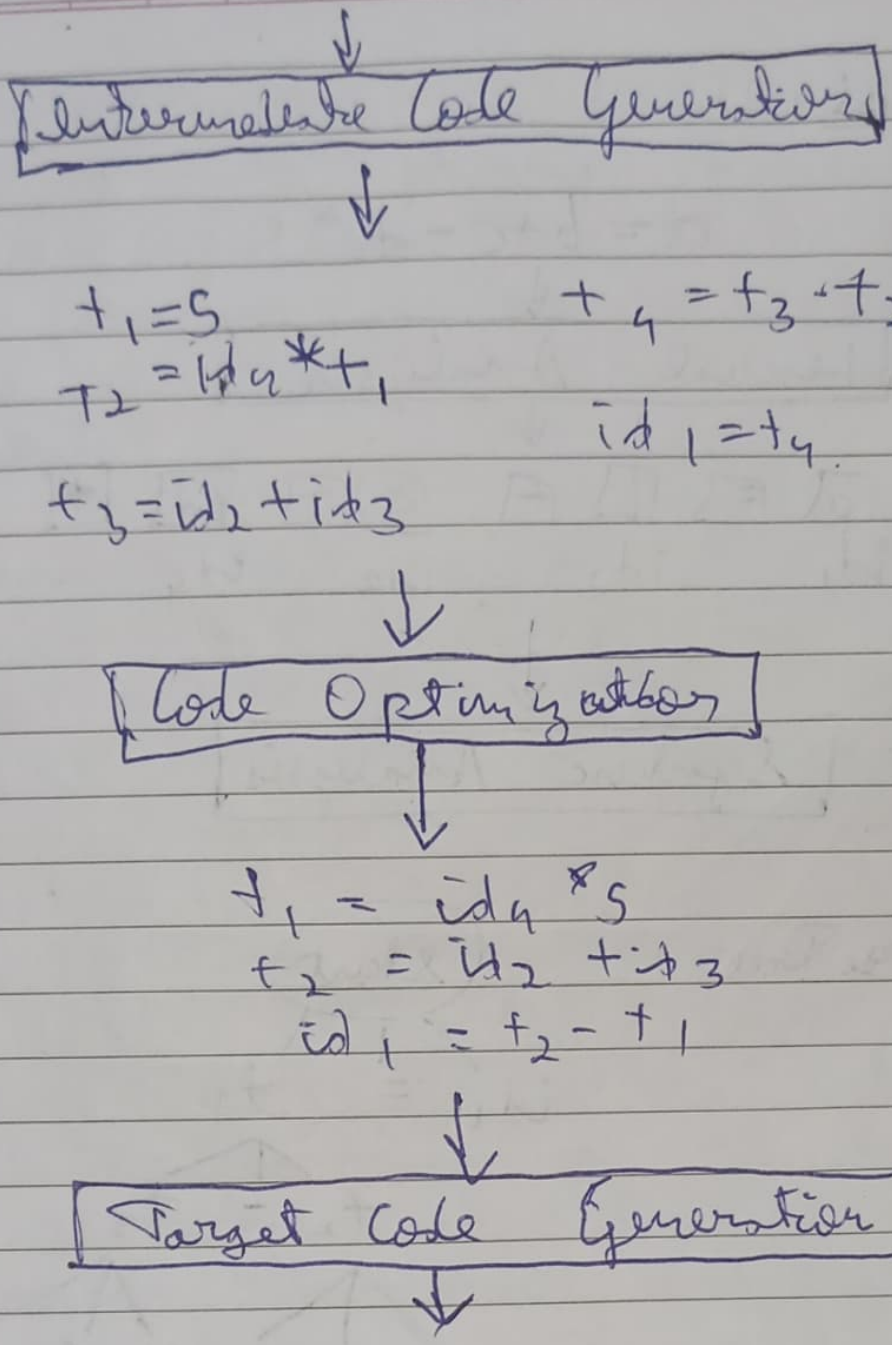
$a = b + c - d * 5$



Parse Tree






Intermediate Code Generation
t1=5
t2=id4*t1
t3=id2+id3
t4=t3*t2
id1=t4
Code Optimization
t1 = id4 * 5
t2 = id2 + t3
id1 = t2 - t1
Target Code Generation
MOV R1, id4
MUL R1, 5
MOV R2, id2
MOV R3, id3
ADD R2, R3
SUB R2, R1
MOV id1, R2