BITS Pilani
Dubai Campus

II Semester

(2022-2023)

Compiler Construction Assignment

Design and Implementation of a user-friendly Programming Language

Done By:

Aryan Zutshi- 2019A7PS0246U

Himanshu Rathi- 2019A7PS0244U

Dhruv Maheshwari- 2019A7PS0020U

Acknowledgment

We are extremely grateful for the help and guidance provided to us by our faculty Dr. Santosh Kumar and Dr. Angel Arul Jothi who provided us with an opportunity to explore different types of programming languages to complete our assignment. This assignment helped us explore different approaches to analyzing and creating a programming language with the help of various books and online resources. It allowed for more clear visualization of the problems making them more coherent for better understanding. The continuous support and positive criticism of all our classmates, friends, seniors, and parents have made this project possible and we greatly appreciate their support. Thank you, again, for aiding us in the completion of this project.

Regards

Introduction

In this assignment, we were assigned the task to design a user-friendly programming language, further, a translator for the same that would translate a program written in our language to one of the target (preferably C) language of our choice.

We formulated a language called HinC, which would then be converted using a translator to the target language C. This language was formulated to provide a more legible language to the vast demographic of Hindi speakers who constitute up to 4.429 percent of the world's total population. Our syntax is inspired by the various prevailing words used in the Hindi language. This would allow even individuals who are unable to effectively understand English to be able to code in the language of Hinglish using HinC.

We will be implementing the translation by creating a new lexical analyzer that is functionally analogous to lex in certain aspects.

1 A Language Translator from Hin*C* to C

We design a language Hin*C* somewhat closer to the pseudocode template to have better readability. Our translator will translate the program written in Hin*C* to a C program.

1.1 Constructs in Hin*C* language

We assumed the language Hin*C* to not support function calls and libraries and hence the so generated C program would always follow the template:

```
#include <stdio.h>
int main()
{
        statements;
}
```

1.2 The symbol; separates statements as in C

1.3 Comments

Line comments $$

Block Comments are usually enclosed in $- and -$ (C equivalent for /* comments */).

1.4 Declaration statements

1. ~var_type var_name$_1$ var_name$_2$ var_name$_3$; where ~ is a keyword indicating these are variables with names *var_name$_i$*, each being of data type var_*type*, which can be *ank, akshr, dashm* or *dodashm*.
2. ~var_type var_name[size][size]; declares a 2D array where ~ is keywords and *type*, as usual, can be *ank, akshr, dashm* or *dodashm*.

The language doesn't support initialization during declaration for arrays.

1.5 Operators:

1. ":" is the assignment operator (*a* = 10; in C would be *a* : 10; in HinC).
2. Aur is equivalent to *&&* in C.
3. Ya is equivalent to || in C.
4. Ulta is equivalent to !(logical operator) in C.
5. Brbr is equivalent to == in C.
5. Everything else is the same as in C.

1.6 Conditional statements:

1. Agr (*condition*) Toh *statements* | $$ | is used for ending
2. Agr (*condition*) Toh *statements* Vrna *statements* |

3. Agr (*condition*) Toh *statements* VrnaAgr (*condition*) *. . .* Vrna *statements* | //nested if, need not end in a Vrna block.

In addition, our Hin*C* supports Ja(goto) operator (same as the goto in C) subsuming the actions of both *todh(break)* and *(continue)* operators in C. Correspondingly, we have labels in our program (LABEL followed by'=>' operator).

1.7 Loop structures:

1. Jabtak(*condition*) Tabtak *statements* |
2. Phirse *statements* Jabtak(*condition*) $$Corresponds to do-while(*condition*) in C
3. Dohrah(i:x,y,z) *statements* | //loop start from x, goes till y(y not included), step size is z
4. Dohrah(, ,) works similar to "for loop" in C.

1.8 Input and Output:

1. Padh($var_1, var_2, . . . , var_n$); $$In contrast to C, the function Read() doesn't bother about verifying the data types associated with variables.
2. Likh($var_1, var_2, . . . , var_n$); $$Instead of variables, we could even have interactive messages as static strings enclosed in single quotes (say, 'The value of x is') as arguments to function Likh

1.9 The main procedure:

mukhya()

shuru:

$statement_1$

$statement_2$

.

.

.

$statement_n$

khatm

2 Sample Translation

2.1 Hin*C* Program	2.2 Equivalent *C* Program

mukhya()
Shuru:
~int num ;
~int i, flag;
flag : 0;
Likh ('Enter the
value to be
checked :');
Padh(num);
Dohrah(i:2;num;1)
Agr (num%i Brbr 0)
Toh
flag : 1;
Ja Exit;
KhatmAgr
KhatmDohrah
Exit=>
Agr(flag Brbr 1)
Toh Likh ('The value' ,num, 'is not prime');
Vrna
Likh ('The value' ,num, 'is prime');
KhatmAgr
khatm
	#include <stdio.h>
	int main( )
	{
	int num;
	int i , flag ;
	flag =0;
	printf("Enter
	the value to
	be checked :"
	);
	scanf("%d",&
	num);
	for(i=2; i<num; i=i+1){
	if (num%i ==0){
	flag =1;
	goto Exit;
	}
	}
	Exit :
	if(flag ==1){
	printf("The value %d is not prime");
	} else {
	printf("The value %d is prime");
	}
	}

mukhya()
shuru:
~ank a[4];
a[0] : 5;
a[1] : 6;
a[2] : 7;
a[3] : 8;
~ank sum;
sum : 0;
Dohrah(i:0,4,1)
sum : sum + a[i];
KhatmDohrahddss
Likh ('Sum of elements of array', sum);
Khatm
	int main()
	{
	int a[4];
	a[0] =5;
	a[1] =6;
	a[2] =7;
	a[3] =8;
	int sum;
	sum=0;
	for (int i=0; i<4;i=i+1)
	{
	sum=sum+a[i];
	}
	
	printf("Sum of elements of array %d",sum);
	return 0;
	}

3. Functionalities Implemented

1. Data Type Implementation and Validation of Assignment
    i. Basic Data Type Implementation and Validation (int, char, float, double) => (ank, akshr, dashm, dodashm)
    ii. Derived Data Type Implementation (Array, Struct) => (stru)

2. Control Statements Implementation
    i. Conditional Statements (if, else-if, else) => (Agr, Vrna Agr, Vrna) including implementation of nested if-else statements
    ii. Loops (for, while, do-while) => (Dohrah, Jabtak, Phirse)
    iii. Goto, Break and Continue => (Ja, Todh, Chlnedo)

3. Arithmetic and Logical Operators
    i. And (&&) => (Aur)
    ii. Or (||) => (Ya)
    iii. Not (!) => (Ulta)
    iv. Equal to (==) => (Brbr)
    v. Other Operators remain same as C

4. Only #include<stdio.h> as header file is available as of now

5. Facilitating Input and Output using Likh() and Padh() functions// printf() and scanf() respectively

6. Block Comments implemented using $- 'comment' -$ and line comments using $ 'comment' $

7. In our program we are required to declare main function first and then user-defined functions are optional. The user-defined functions are of type void and do not allow the user to pass parameters into the defined function.

4. Difficulties Faced

1. Struct is limited to only valid declaration, and cannot access the variables declared in the struct, as it would require the implementation of nested tables or stack of symbol table as struct can contain any type of variable including struct itself. It'll go into infinite loop if we define same struct inside itself therefore pointers are required, which proved too time-consuming and complex at our level.
2. Our program is restricted to single array & struct variable declaration in one.

3. Array index out of bounds will not be handled on our end but by the C compiler after translation. Also, we cannot read and write an array. This could've been done but due to restriction of time this was left.

4. Difficulty in implementing other derived datatypes such as union

5. Due to a lack of time, knowledge & complexity, we were unable to implement the ideas we formulated for user-defined functions as we'll have to handle input parameters, return type etc.

6. Due to a lack of time, we were also unable to implement multidimensional arrays. We have formulated the theoretical approach to achieving it but were unable to implement it.

5. Ideas not implemented but theorized

1. The implementation of pointers is required to implement other features such as all functionalities of a struct.
2. We planned to implement multi-dimensional arrays with a fixed number of dimensions, and we would employ an array to store the values of the additional dimensions along with the number of dimensions defined by the user and the values of each element.
3. We didn't implement variable scope checking as it would require us to use multiple symbol table (one way is by using a struct which will contain a symbol table for normal variable and array table for arrays and struct table for struct and we can have multiple instances of them and as we move inside a nesting we'll increase the struct index but since we were unable to implement struct successfully we couldn't do it).