Compiler Construction Assignment

Done By :
Anushka S - 2019A7PS0208U
Dhakshina Priya I - 2019A7PS0003U
Mukta R - 2019A7PS0134U

Introduction

Ivory is a user-friendly programming language which has simplified syntax and is easy to learn for newbee coders. It also makes use of additional features like predefined functions to shorten and simplify code. It is an amalgamation of general pseudocode of popular languages like C and Python. Our aim is to build a translator that translates Ivory to C programming language.

The language syntax and implementation is briefly discussed and is compared with the syntax of the C programming language in the following format:

1.1 Constructs in Ivory language

IVORY Syntax	**Syntax Change**	**Description**	**C Syntax**

include “stdio.h”
#Z main()
{
stmts..
}
	No change in syntax	Was implemented using PGM start grammar that allowed for the following structure with modifications such as additional header files that can be specified in between “ ”	#include <stdio.h>
			int main()
			{
			// stmts;
			}

1.2 Separator

Two dots (..) are used to separate statements in this language.

#Z a..	No change in syntax	Was implemented by swapping the semicolon “;” for “..”	int a;

1.3 Comments

Only single line comments are allowed in this language which can be done using the symbol (^^)

ex : ^^Hello World	No syntax change	Successfully implemented single line comments	// Hello World

1.4 Declaration statements

The # symbol represents the declaration of a data type. The # is followed by one of the following datatypes. Variable Names - defined same as done in C Programming Language.

Basic Data Types: #Z #D #C	Change in basic data types: Int - Z Char - C Double - D	Removal of Float data type and substitution of it with Double datatype due to simplicity and similarity between the two.	int double char
Ex of **Declaration** : #Z a, b, c..	No change in syntax	Implemented same as C, variable preceded by data type and variables of same datatype separated by comma.	int a,b,c;
Ex of **initialization** : #Z a:6, b:7, c:8..	No change in syntax	Implemented same as C, variable preceded by data type and variables	int a=6, b=7, c=8;

		of same datatype separated by comma and the variable assigned a constant value using the assignment operator.	
Ex of **Arrays:** #Z a[5].. a[3] : 4.. #Z b[3][4].. #D c[3] : {1.0, 2.2, 3.7}.. #D d[2][2] : {{2.5, 3.3},{1.0, 7.8}}..	No change in syntax	Implemented multidimensional arrays and declaration, initialisation, and assignment operations for them.	int a[5]; a[3] = 4; int b[3][4]; double c[3] = {1.0, 2.2, 3.7}; double d[2][2] = {{2.5, 3.3},{1.0, 7.8}};

1.5 Operators

Arithmetic operators Addition + Subtraction - Multiplication * Division / Remainder % Exponent ^	No syntax change	The arithmetic operators were implemented the same as in C to maintain standard (except that exponent has a direct implementation unlike in C).	Addition + Subtraction - Multiplication * Division / Remainder % Exponent : math.pow() function
Scope Operator ::	No syntax change	The scope operator is the same as in C.	Scope Operator ::
Logical Operators And ‘& Or ‘| Not ~	No syntax change	Logical AND and OR operator implemented by preceding “&” and “|” with “ ‘ ”, and NOT is implemented using ‘~’	And && Or || Not !

Bitwise Operators And & Or |	No syntax change	The bitwise operators were implemented the same as in C to maintain standard	And & Or |
Relational Operators Assignment : Equivalence = Less than < Less than or equal to <: Greater than > Greater than or equal to >: Does not equal !=	No syntax change	Implemented	Assignment = Equivalence == Less than < Less than or equal to <= Greater than > Greater than or equal to >= Does not equal !=

1.6 Conditional statements

<condition> ifT{}
 	
<condition> ifT{}
ifF{}
 	
<condition> ifT{}
nxtT{}
ifF{}
	No syntax change
	
	(The <> just act as a placeholder and do not form the actual syntax)
		Successful implementation of if-else , where if is represented by ifT (if True), else if is represented by nxtT (next True), and else is represented by ifF (if False).	if (condition) {
			}
			
			if (condition) {}
			else {}
			
			if (condition) {}
			else if (condition) {}
			else{}

Break - ~b~ Continue - ~c~ Goto - goto	No syntax change	Break and Continue implemented with the consideration that they can only be used inside loops.	break; continue; goto <label>;

1.7 Loop structures

For/While Loop
loop <init>;<cond>;<incr/decr>
{
stmts..
}
	No syntax change
	
	(The <> just act as a placeholder and do not form the actual syntax)
		Our language implements a loop in a format similar to C's for loop format. It does not differentiate between for loop and while loop.	for (init; cond; expn)
			{
			stmts;
			}

Do-While loop do { stmts.. }loop<cond>..	No syntax change (The <> just act as a placeholder and do not form the actual syntax)	The do-while loop is implemented much like that in C, beginning with the "do" keyword and ending with a "loop" statement with the condition for the loop to keep executing.	do { stmts; } while (cond);

1.8 Input and Output

We modified the syntax for the read / write functions for the following reasons:

- '1' as a token would clash with integer token values causing problems in the rest of the grammar. Hence, we replaced '1' by 'std' token to denote standard input/output.
- Having a list of placeholders with data types as one argument and a list of variables as the next argument made type checking and count checking in the grammar difficult. Therefore, we made it of the format placeholder+datatype: variable, to have corresponding pairs making type checking easier.

rd(1; "@Z, @Z,.." var1,var2,...).. rd stands for read. The first argument indicates where the input is being read from, in this case 1 denotes standard	Syntax changed: rd(**std**; **@#Z:var1, @#Z:var2,.....**).. rd stands for read. The first argument indicates where the input is being read from, in this case	After modifying the syntax, we were able to implement the read operation with type checking and syntax constraints.	-For standard ip (std) scanf("%d%d", var1, var2); -For file-read fscanf(filename, "%d%d", var1, var2);

input. If the input needs to be read from a file the file name can be put within double quotes. The second argument represents the place holder in this case @Z is for integer data type and the third argument represents variable names where the value read will be stored.	**std** denotes standard input. If the input needs to be read from a file the file name can be put within double quotes. **The second argument represents a placeholder (with datatype) followed by a variable appended with ':' operator for each corresponding placeholder.** Eg: rd(std;@#Z:var1)..		
wr(1; " The answer is @C";var1).. wr stands for write. The first argument indicates where the result is being displayed, in this case 1 denotes standard output. If the output needs to be written into a file the file name can be put within double quotes. The second argument defines the content or print	Syntax changed: wr(**std**; "String" **@#Z:var1, @#Z:var2,..... "**Cont. String").. wr stands for write. The first argument indicates where the input is being written to, in this case **std** denotes standard output. If the output needs to be written to a file the file name can be put within double quotes. **The second argument**	After modifying the syntax, we were able to implement the read operation with type checking and syntax constraints.	-For standard op (std) printf("Hi %d Hope you are good", var1); -For file-write fprintf(filename, "Hi %d Hope you are good", var1);

statement. The place holder (for the variable to be printed) is represented using @ symbol followed by the variable data type. The third argument represents variable names whose value will be displayed.	**represents a placeholder (with datatype) followed by a variable appended with ':' operator for each corresponding placeholder. If any string needs to be output as well, it can be written in quotes around the variable.** Eg: wr(std; "Hi" @#Z:var1 "Hope you are good.")..		

1.9 Functions

Function Definition
fnc func_name (return_type; arg1, arg2,...)
{
stmts..
}
	No syntax change	Implemented function definition in the following format. Here each argument acts either as a variable declaration or as a variable initialisation.	return_type func_name (arg1, arg2, arg3,...)
			{
			stmts;
			}

Function Call func_name(arg1,arg2,arg3,...)..	No syntax change	Implemented function call to be able to account for factors such as if function exists, if sufficient number of arguments being passed as well as type checking requirements.	func_name(arg1, arg2, arg3,...);

1.10 Range

This inline function produces a random positive or negative number within a specified range given as input by the user.

eg :

```
#Z main()
{
   #Z a: Range(0,100,‘n’)..
   wr(1; “The number is : @Z”;a)..
   return 0..
}
```

The above feature’s grammar and action was not implemented.

1.11 def

def variableName variableValue eg: def Pi 3.14.. This keyword is used to include constants into the code.	No syntax change	Implemented a feature to define constants.	#define NAME CONSTANT Eg: #define PI 3.14

1.12 Array Functions

fnc push(void; arrayname, elementToPush):
 	
This inline function adds
	No syntax change	Implemented the following as an inline function which gets translated to C code	for (int i = size - 1, flag = 0; i >= 0; i--){
			if (array[i]!=0){
			i++;
			flag = 1;
			if (i >= size){
			printf(/ "Error, array

an element to the end of the array.		loop.	full/out of bounds/");
			break;
			}
			array[i] = element;
			break;
			}
			}

fnc pop(void; arrayname):
 	
This inline function removes the element from the end of the array.
	No syntax change	Implemented the following as an inline function which gets translated to C code.	for (int i =size - 1, flag = 0; i >= 0; i--){
			if (array[i]!=0){
			flag = 1;
			array[i] = 0;
			break;
			}
			}
			if (flag == 0)
			printf(/ "Array empty/");

fnc insert(void; arrayname, elementToInsert, Index) :
 	
This inline function inserts the given element at the specified index.
	No syntax change	Implemented the following as an inline function which gets translated to C code loop.	for (int i = 0,temp=0; i < size; i++){
			if (i==index){
			temp = array[i];
			array[i] = element;
			i++;
			break;
			}
			}
			for (int temp1 = 0; i < size; i++)
			{
			temp1 = array[i];
			array[i] = temp;
			}

fnc remove(void; arrayname, Index) :
 	
This inline function removes the element from the specified index.
	No syntax change	Implemented the following as an inline function which gets translated to C code loop.	for (int i = 0,temp=0; i <size; i++){
			if (i==%d){
			array[i] = 0;
			i++;
			break;
			}
			}
			for (int temp1 = 0; i < size; i++)
			{
			temp1 = array[i];
			array[i - 1] = temp;
			}

fnc deleteEle(void;	No syntax	Implemented the	for (int i = 0,temp=0, flag

arrayname, elementToDelete):
 	
Removes the first occurrence of the element from the list
	change	following as an inline function which gets translated to C code loop.	= 0; i < size; i++){
			if (array[i]==element){
			array[i] = 0;
			i++;
			flag = 1;
			break;
			}
			}
			if (flag)
			{
			for (int temp1 = 0; i < size; i++)
			{
			temp1 = array[i];
			array[i - 1] = temp;
			}
			printf(/ "Element deleted/");
			}
			else
			printf(/ "Element not in array/");

1.13 Record

A record can store multiple items in a single variable. Unlike arrays it can hold multiple elements of various data types. Its size cannot be changed after initialization (i.e. insertion and deletion of elements is not permitted) but values of the elements may be changed anytime after initialization.
The record will be implemented in the form of a structure in C.

eg: Record A:[1,”hello”,2.4 ,”world”]	No syntax change	**Failed to implement the feature.**
		Provided grammar and action for the implementation of a Record structure but the grammar failed testing due to an unidentifiable
			struct A{
			int var0;
			char* var1;
			double var2;
			char* var3;
			};

		segmentation fault.	
fnc changeVal(void; recordName, index, newValue).. It comes with an inline function to change the value of an element.	No syntax change	**Failed to implement the feature.** Provided grammar and action for the implementation of the function but failed to test it due to error in creating a Record.	A.varIndex = newValue;