భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

fns

finculate-not-speculate

Language Specification Document

Written by:

DIYA GOYAL - CS20BTECH11014
DONTHA AARTHI - CS20BTECH11015
KHARADI MONIKA - CS20BTECH11026
MD ADIL SALFI - CS20BTECH11031
NAMITA KUMARI - CS20BTECH11034
NYALAPOGULA MANASWINI - CS20BTECH11035
SUSHMA - CS20BTECH11051

INDEX

1.INTRODUCTION

1.1 DESCRIPTION:

FNS - A procedural programming language to ease financial calculations. It has inbuilt functions to calculate simple interest, compound interest, loan EMIs, SIP and step-up SIP maturity returns, CAGR and XIRR. If time permits, we will include functions for GST and income tax calculation as well. The goal is to do complicated calculations in the backend so that the user has an exact value for the amount of money involved in the picture.

All calculations involving bank loans, returns and investments have the formula of compound interest at its core. Doing mental calculations is non-intuitive because compound interest has an exponential graph, hence making estimates is very difficult. Our language can be a useful tool to automate calculations for the end-user so that they can make better decisions that affect their financial health.

1.2 FEATURES:

1. Easy to learn.
2. Functions like help() that describe the work of other functions.
3. Statically compiled.

2. DATA TYPES

2.1 PRIMITIVE DATA TYPES:

double: It is 64 bits signed, double precision floating point value and follows IEEE 754 specifications.

int: It is a 32 bits signed integer value.

2.2 NON-PRIMITIVE DATA TYPES:

array: sequence of values of a particular data type

date: date data type is used to represent dates in the DD_MM_YYYY format

month: month data type is used to represent a month of the year in MM_YYYY format

Below are examples of variable declarations in our language:

```
int a,b=-5;
a = 5;
double num;
num = 5.67;
date d = 19_10_2002;
month m;
m = 10_2022;
array<int> arr[3] = {1, 2, 3};
```

3.CONVENTIONS

3.1 COMMENTS:

Both single and multiline comments are supported.
All tokens after **$$** are treated as comments and are ignored by the compiler.
Multiline comments start with **$/** and end with **/$.**

```
$$ This is a single-line comment

$/ This is
a multi-line
comment /$
```

3.2 KEYWORDS:

Following are reserved keywords in our language **fns,** they cannot be used as identifiers.

int	double	date	month	array	else
continue	loop	fbreak	break	if	

3.3 IDENTIFIERS:

Identifiers must start with a letter, which can be followed by a sequence of letters, digits, and underscores. Other special characters cannot be used in the identifier. This language is case sensitive, so App, app and aPp are all considered different.

3.4 PUNCTUATORS:

Statement should be terminated by semicolons (;).

3.5 OPERATORS:

Operator	Description	Associativity
+	Addition	Left
-	Subtraction	Left
*	Multiplication	Left
/	Division	Left
%	Modulo	Left
>	Greater than	Left
<	Less than	Left
>=	Greater than or equal	Left
<=	Less than or equal	Left
==	Equal	Left
!=	Not Equal	Left
=	Assignment	Right
&	Logical And	Left
|	Logical Or	Left
!	Logical Not	Left

3.6 OPERATOR PRECEDENCE:

```
( )
!
^
*  /  %
+  -
>  <  >=  <=  ==  !=
&  |
=
```

3.7 OPERATOR OVERLOADING:

1. We are overloading the ‘ - ‘ operator. It is used for subtraction in numerical data types. When used with date data type, it gives the number of days between two dates, similarly when used with month data type, it gives the number of months.
2. We are also overloading ‘%’ operator. On one hand it is used as a modulus operator and on the other hand, it is used to specify the data type of a variable whose value needs to be printed.

3.8 STATEMENTS:

If-else constructs:

```
if(expression)
{
    $$ Statements to be executed
}
else
{
    $$ Else is optional.
    $$ Statements to be executed
}
```

Loop construct:

```
loop(expression)
{
    $$ Statements to be executed
}
```

4.FUNCTIONS:

4.1 FUNCTION DECLARATION:

```
func_name: (parameters) -> (return list)
{
    $$ Statements to be executed
}
```

4.2 INBUILT FUNCTION EXAMPLES:

```
compoundIntrst:(double principle, double rateY, double timeY, int
n)->(double amt)
{
    rateY = rateY/100;
    amt = principle*(1+rateY/n)^(n*timeY);
}
SIPmaturity:(double mnthlyInv, double growthRateY, int months)->(double
maturity)
{
    double i = growthRateY/12/100;
    maturity = mnthlyInv*((1+i)^months-1)*(1+i)/i;
}
SIPmaturityDeets:(double mnthlyInv, double growthRateY, int
months)->(double maturity, double inv, double intrst, double returnPerc)
{
    SIPmaturity(mnthlyInv,growthRateY,months)->(maturity);
    inv = mnthlyInv*months;
    intrst = maturity-inv;
    returnPerc = intrst/inv*100;
}
```

4.3 MAIN FUNCTION:

```
execute:()->(int x)
{
    double m,inv,intr,r;
    double mnthlyInv = 1000, growthRateY = 5.5, months = 17;
    SIPmaturity(mnthlyInv,growthRateY,months)->(m,inv,intr,r);
    display("Maturity = %d\nTotal Investment = %d\nTotal Profit =
%d\nProfit Percentage(%) = %d",m,inv,intr,r);
}
```

Expected output :

```
Maturity = 17718.64
Total Investment = 17000
Total Profit = 718.64
Profit Percentage(%) = 4.22
```

5. Grammar

program: declarations ;
declarations: declarations declaration
| %empty
;

declaration: function
| vardec_stmt STM_DELIM
;

function: IDENTIFIER COLON_OP L_PAREN paramdecls R_PAREN "->"
L_PAREN paramdecls R_PAREN compound_stmt R_BRACE
;

paramdecls: paramdecl

```
|                %empty
;

paramdecl:  paramdecl COMMA_OP typename IDENTIFIER
|                typename IDENTIFIER
;

typename: INT
|               CHAR
|               DOUBLE
|               STRING
|               DATE
|               MONTH
;

print_list: %empty
|                print_list COMMA_OP exprs
;

print_stmt: DISPLAY L_PAREN STRING_LITERAL print_list R_PAREN
STM_DELIM
;

stmt: compound_stmt R_BRACE
|       selection_stmt
|        jump_stmt
|       expression_stmt
|       empty_stmt
|       vardec_stmt STM_DELIM
|       iteration_stmt
|       print_stmt
;

expression_stmt: exprs STM_DELIM ;

jump_stmt: CONTINUE STM_DELIM
|                BREAK STM_DELIM
|                FUNCTION_BREAK STM_DELIM
```

```
;

empty_stmt: STM_DELIM
;

vardec_stmt: typename vardec1
|                   vardec_stmt COMMA_OP vardec1
;

vardec1: IDENTIFIER ASSIGN initializer
|            IDENTIFIER
;

initializer: expr
|              L_BRACE initializer_list R_BRACE
;
initializer_list: initializer
|                     initializer_list COMMA_OP initializer
;

compound_stmt:  L_BRACE
|                           compound_stmt stmt
;

selection_stmt: IF p_expr stmt %prec LOWER_THAN_ELSE
|                       IF p_expr stmt ELSE stmt
;

iteration_stmt: LOOP p_expr stmt
;

p_expr: L_PAREN expr R_PAREN
;

exprs: expr
|         exprs COMMA_OP expr
;
```

```
expr: NUMBER
|       DOUBLE_CONST
|       STRING_LITERAL
|       DATE
|       MONTH
|       IDENTIFIER
|       L_PAREN exprs R_PAREN
|       expr L_BRAKET exprs R_BRAKET
|       expr L_PAREN R_PAREN
|       expr L_PAREN exprs R_PAREN
|       expr ASSIGN expr
|       expr ADDITION_OP expr
|       expr SUBTRACT_OP expr %prec ADDITION_OP
|       expr MULTI_OP expr
|       expr DIV_OP expr %prec MULTI_OP
|       expr MOD_OP expr
|       expr "+=" expr
|       expr "-=" expr
|       "++" expr
|       "--" expr %prec INC_OP
|       expr "++"
|       expr "--" %prec INC_OP
|       expr LESSER_OP expr
|       expr GREATER_OP expr
|       expr OR_OP expr
|       expr AND_OP expr
|       expr EQ_OP expr
|       expr NE_OP expr %prec EQ_OP
|       expr POWER expr
|       MULTI_OP expr
|       SUBTRACT_OP expr
|       '!' expr
;
```