BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PRINCIPLES OF PROGRAMMING LANGUAGES

ASSIGNMENT-1

DDL(Dress Designing Language)

-Language for professional dress designers to showcase their creativity in simple and convenient way.

SUBMITTED BY

JASHANJOT KAUR
2012A7PS044P
VIVEK KUMAR SAHU
2012A7PS058P

1.DESCRIPTION OF APPLICATION DOMAIN

Dress Designing

Our language has been designed for the dress designers. It aims at increasing the creativity of the dress designer, by providing him/her with a language that can help experiment on new designs, and do minor or major changes to the existing design, without starting from scratch, and thus saving a lot of his/her time and effort.

By using DDL,the designer can come up with the accurate drawing of his/her design by writing just few lines of code,instead of having have to make sketches by hand .At the same time,it allows the designers to pre compute the cost of the dress before it is sent for production.One more functionality that DDL provides is that it allows the user to sort dresses in various ways.One can select all the dresses of a particular type or those sharing similar attributes.

2.LIST OF PROBLEMS IN OUR DOMAIN THAT CAN BE SOLVED BY USING OUR LANGUAGE

The DSL we have designed can be used to design a new dress,compare two dresses on the basis of their attributes, find dresses with a given attribute(eg: left pocket on the top),find the cost of a dress etc.We have described the problems that can be targeted using this language below:

1.**Designing a new dress**:

Our DSL allows us to design a wide variety of dresses,choose the colors of our choice,add various type of buttons at various locations of the dress. We also have the liberty of adding collars and sleeves(of different lengths).Various other objects such as frill,zipper,embroidery can also be added.In a nut shell, dress designer has complete freedom to come up with new and creative designs for respective dresses.

Finally after designing our dress,we can view the final image.

2.Editing all dresses of a given type

One can also feel the need to edit the dresses they have already designed. Our DSL provides operators and features to override the existing properties of attributes, or add/remove some object(say pockets,collars,etc.) to another object(say shirt),in a simple and convenient way

3. Search for dress(es) in a given list/array of dresses

Our DSL allows users to search for dresses from a given list of dresses in various ways:

a)Finding all the dresses having a particular object eg: exactly two pockets.
NOTE:Here the pockets can have different properties(eg:color)

b)Finding all the dresses of the same type,say jeans.

c)Finding all the dresses having identical objects.Eg: All dresses containing two buttons of green color and medium size.

4. Calculate cost of a given dress

Our DSL can also be used to calculate the cost of a dress with given attributes.We can add up the cost of individual attributes to compute the final cost.It only takes a few lines of codes and is therefore, convenient for the designer.

3.LIST OF PROGRAMMING FEATURES IN OUR LANGUAGE

We have provision for the following features in our language:-

1.Looping

For traversing a given list, or re-doing a particular snippet of code fixed number of times, we have provided looping.It has only one form.

For example, for each $x in (0,50) or for each $x in $Dress[$x]

2. Branching

For searching for a particular attribute, among many, we have provided conditional statements to make the process easier.

3. Polymorphism

Some of our operators exhibit different functionalities on different data types.

For example, '+', when operating on 2 integers, does a simple arithmetic addition(Commutative).But while operating on objects, adds one object on top of another(Not commutative).

4.Object Orientation(to an extent)

All our data types on instantiation, form an object.That is, they have properties and exhibit some behavior(i.e. only location can be specified by the object)

5.Strongly Typed

All our variables have to be assigned one of the data types while being declared.

6.No sub-programming capability

We have not provided provision for functions in our language for the following reasons:

a.Creating similar items can be done using '*' instead of calling subprogram(function) defined again and again.In fact, '*' is more writeable.

b.If we need to make items with certain properties different from original, we can instead edit the required attributes in 1 line using '->', '-', '+' operators.

That is, using subprograms in our language does not, in any way, make our language more efficient for the writer or reader, plus, we can do our tasks without using subprograms(functions).Hence, we do not have sub-programming capability.

Languages can be evaluated based on some criterion as mentioned below. Our language stands good overall in all the aspects, as can be seen below:

1.Readability:

We have tried to keep uniformity with operator definition(for example, all operators starting with '@' refer to location, etc.) plus, operator sizes are very small, and we also have kept data type names to be as intuitive as possible, to make readability easier for our language. Also, all our identifiers (variables) start with '$' which helps in distinguishing identifiers from other tokens. '*' also helps increase readability since multiple copies are created at once.

2.Write-ability:

Our small-sized and uniform operator definitions are a boost to write-ability. Starting every identifier with a '$' symbol, though makes the code more readable, it also makes the writing of code a little bit difficult.But we have '*' operator defined to make as many identical copies(objects) as we want, which undoubtedly increases readability and write-ability.

3.Reliability:

Our application domain does not require converting one data type to another, and hence we have not provided any such feature. This has lead to our language becoming very reliable.

4.Cost:

Our language has many operators and data types, which provide complexity to our language.but our scheme of uniformity in our operators, helps reduce some lexical analysis cost. Overall, our language can be termed a bit costly at the expense of providing more readable and write-able code.

5.Well defined:

Uniformity in operator declaration, enough data types to reduce work of programmer, and well defined data types have lead to a well defined language.

4.LIST OF ALL TOKEN TYPES

a.KEYWORDS

i)DATA TYPES

The set of data types is exhaustive, that is, we do not allow programmer to create their own data types, since all of our creating, designing, storing, searching can be done using the data types provided here.

Following are the data types provided by our language.
We have assumed the following default values on initialization, for the code snippets provided in Part 5.

DATA TYPES		*DEFAULT VALUES*
➢ Shirt	--	plain white shirt, size =S.
➢ Jeans	--	plain blue jeans,size=S.
➢ Kurta	--	plain white kurta, size =S.
➢ Frock	--	plain white fock, size=S.
➢ Button	--	size=S, color=white
➢ Pocket	--	size=S, color=white
➢ Collar	--	color=white
➢ Sleeve	--	color=white, size=S
➢ Zip	--	color=white
➢ Frill	--	color=white, size=S
➢ Emb	--	plain white image

- ➢ Size --denotes size of the object using letter symbols
- ➢ Color --denotes color of the object using ARGB form

- ➢ Univ --Univ is universal data type, i.e. all data types are of Univ data type.

Following data types have similar meaning as in Java or Python:-

- double
- bool
- List
- Array
- String

ii)***OTHERS***

- for each --used for traversing
- in --used along with 'for each'
- try --provided for exception handling(for example, when image is not available)
- catch --provided for exception handling
- if --if statement
- else --else statement
- INF --INFINITY (constant value)
- begin --start keyword for our program
- end --end keyword for our program
- break --keyword to go out of the loop

b.OPERATORS

+	On integers and doubles, it does simple arithmetic addition. On objects, it adds the object on the RHS to the object whose RHS is -> on the same line, preceding this statement.
-	On integers and doubles, it does simple arithmetic subtraction. On objects, it removes the object on the RHS from the object whose RHS is -> on the same line, preceding this statement.
*	creates as many objects identical to the object on the LHS, as is the integer on the RHS.

->	adds/removes to the object on the LHS, all the objects to the RHS according to the + or - operator before those objects. adds/overrides properties to the object on the LHS.The properties are on the RHS of the statement, and start with @ or %.
=	Assignment operator for integer, double and bool values.
==	Equality operator for integer, double and bool values.
{	Used with if, for and try,catch statements, denoting start of a block.
}	Used with if, for and try,catch statements, denoting end of a block.
(	With if, start symbol for the condition With for, start symbol for specifying range With %, to specify start of the ARGB value tuple. Otherwise to specify precedence order of statements.
)	With if, end symbol for the condition With for, end symbol for specifying range With %, to specify end of the ARGB value tuple. Otherwise to specify precedence order of statements.
[	With Arrays and List, to specify start of range as in Python or Java
]	With Arrays and List, to specify end of range as in Python or Java
;	End of statement
:	With begin, implies start of program. With end, implies end of program.
||	OR operator
&&	AND operator
!	NOT operator

%	gives the object on LHS, the size which is denoted on the RHS gives the object on LHS, the ARGB color denoted on the RHS gives the object on LHS, the embroidery image url given on RHS.
<>	returns true if object o LHS is of datatype given on RHS, else returns false
^^	object on its RHS is displayed(both back view and front view.
^^ \n	newline operator
|||	returns cost of the object on its RHS
~	returns true if object on its LHS and RHS are identical(i.e. have same properties, need not point to the same memory location.)
#	One line comments

The following operators specify the location at which the object will attach to, when added to the other object(using -> and +)

@b	at the back
@f	at the front
@l	at left half
@r	at right half
@u	at upper half
@d	at bottom half(i.e. down)
@mid	at middle, from the top
@cr	to fill the entire location completely
@bd	to fill along the borders of the give location

c.IDENTIFIERS

All our identifiers will start with a '$' symbol and can contain any letters from the english alphabet, any number, or underscore character in any order. Symbol after $ can also be a number.For example, $bu, $po, $x.

4. CODES IN OUR LANGUAGE

#1.Creation of a new dress.

Here, we are creating a red(alpha=1) colored collared kurta of medium size, with 1 pocket on its right side and 3 small sized buttons from top on its front. It has full length sleeves with embroidery type taken from a given url, at the bottom of the sleeves.

EXPLANATION: While approaching this problem,we will first of all select a blank template of kurta by writing the following snippet of code:

```
Kurta $kr;
```

Then we can specify various things such as colour,size etc.Eg: For chosing the colour,

```
$kr % (1,255,0,0);
```

Now,we need to add various objects to the kurta such as pocket.We will first of all declare the pocket object,then specify it's attributes(like size,color) as follows:

```
Pocket $po;
$po-> %m % (1,255,0,0);
```

And the add the location on which it will attach to, when added to another object.

```
$po->@f@u@r@cr;
```

After this,we need to add it to the kurta

```
$kr->$po ;
```

On the similar lines,we can add sleeves and embroidery to the sleeves.After we are done with adding objects to the kurta object,we need to view the final object,which we can we do by the help of following code snippet:

```
^^$kr;
```

```
begin:                          #indicates start of our program
        Kurta $kr;              #declare(and initialize to default values) kurta data
                                #type
        $kr % m;                #shirt size=M
        $kr % (1,255,0,0);      #red color

        Button $bu ;
        $bu-> % m % (1,0,0,0);  #button size = M, color=black
        $bu-> @f@mid@u;

        String $str="Embroidery image not found";
        bool $found=false;
        int $i;
        Emb $em;

for each $i in (0,INF){                 #if url is not found,the code will repeat
        if($found==false){
                try{
$em->% "http://embroworld.com/Files/free-embroidery-design-146.jpg";
                $found=true;
                break;
                }catch{
                        ^^$str;
                }
        }
}

        $em->  % s @ bd @ d ;
        Sleeve $sl;
        $sl->  % l % (1,255,0,0) ;
        $sl-> +$em;

        Collar $co;
        $co -> %(1,255,0,0) ;

        Pocket $po;
        $po-> %m % (1,255,0,0);
        $po->@f@u@r@cr;

$kr-> +$bu*3 + $co + $po ;      #add all the objects to the kurta
^^$kr;                          #view kurta
:end
```

#2.Edit all given dresses of a given type.

We are given a list of, say 50, collared shirts of may be different design.We need to remove collars from all of them.
We assume that we are given an array of 50 shirts, called $Shirts[50].
To solve the problem,we will iterate over the entire array of shirts and in a particular shirt,we will check the data type of all the objects using the '<>' operator as follows:

```
if($x<>Collar==true)
```

Now,if the data type turns out to be collar,we will remove it by using the following command:

```
$Shirts[$i] ->- $x;
```

```
begin:
        int $i;

        for each $i in (0,50){                          #iterate over the complete array
                Univ $x;
                for each $x in Shirts[$i]{      #iterate over all the objects in Shirts[$i]
                        if($x<>Collar==true){   #if collar is found
                        $Shirts[$i] ->- $x;     # remove collar
                        }
                }
        }
        :end
```

#3.Searching for dress(es) in a given list/array of dresses

#3.a) Search in a given list, which of the dresses have a given particular attribute.

We assume that we are given a List of 50 dresses, called $Dress[50].We will find out which of the given dresses have exactly 3 buttons on it.

```
begin:
        int i;
        bool $b[50];
                for each $i in (0,50){
                $b[$i]=false;                #initialize to false
                }
for each $i in (0,50){
                Univ $x;                        #since we will be iterating over objects of different
                                                    # data types,we have used universal data type
        for each $x in $Dress[50]{
                int $m;
                int $k=0;                       #counter to check the number of buttons
                for each m in (0,4){
                        if( x<>Button==true){
                                $k=$k+1;         #increment the counter if button is found
                        }
                }
        if($k==3){                              #if the number of buttons is equal to 3
                $b[$x]=true;
        }
     }
}


for each $i in (0,50){
        if($b[$i]==true){
                ^^$Dress[$i];
                ^^\n;                   #implies go to next line after showing $Dress[$i];
        }
}

:end
```

#3.b)Search for a particular type of dress in a given list of dresses.

We are given a list of 50 dresses denoted by $Dress[50].We will search for only jeans in the given collection.

```
begin:
	$Dress[50];		//Given, contains mix of 50 different dresses.

	int $i;

	bool $b[50];

	for each $i in (0,50){
		$b[$i]=false;
	}


	for each $i in (0,50){
		if($Dress[$i] <> Jeans){		#if the object type is jeans
			$b[$i]=true;
		}
	}

for each $i in (0,50){
	if($b[$i]==true){
		^^$Dress[$i];
		^^\n;
	}
}

:end
```

#3.c) From a given list of dresses, select dress having the exact given attribute.

We assume that we are given a List of 50 dresses, called $Dress[50].We will find out which of the given dresses have buttons of medium size and green color(alpha=1).

```
 begin:

        int i;
        bool $b[50];
                for each $i in (0,50){
                $b[$i]=false;
                }

Button $bu;

$bu->%m%(1,0,255,0);                    # create button object of medium size and green
color
        for each $x in (0,50){
                for each $i in $Dress[$x]{
                        if($i~$bu == true){     #compare the two objects
                                $b[$x]=true;
                        }
                }
        }
for each $i in (0,50){
        if($b[$i]==true){
                ^^$Dress[$i];
                ^^\n;                   #implies go to next line after showing $Dress[$i];
        }
}

:end
```

#4.Calculate cost of a given Dress

We will calculate the cost of kurta designed in sub task 1.That is we have $kr available to us.Also the basic cost of an object is already initialized(defined by the language designers itself).So,we need to add the cost of all the objects in the dress to the cost of the object(i.e. the dress without any objects added to it).

```
begin :

        double $cost=|||kurta;
        Univ $x;
        for each $x in $kr{
                if($x<>Button || $x<> Pocket || $x<>Sleeve || $x <> Emb){
                        $cost= $cost + |||x;
                }

^^$cost;
:end
```