Bilkent University

Department of Computer Engineering

CS 315 Programming Languages

Designing a Graph Definition and Querying Language

Tutorial for SAPL

Section: 01 Group Number: 13

Uğurcan AYTAR, 21200487
Umut Cem SOYULMAZ, 21201744
Yusuf AVCİ, 21202512

1) Entry point:

SAPL supports an entry point which is named as main. Programmer should write "beginmain" at the beginning of the code section that is to be run. In addition to this, programmer have to write "endmain" at the end of code section that is to be run.

Example code is given below:

```
beginmain

endmain
```

These words are reserved words and cannot bu used for other purposes.

2) Variable names:

In SAPL variable names are being used in order to access the created variable later on. The variable name should start with a lowercase character and and should not include any character other than alphanumeric characters.

3) White space and new lines:

In SAPL white space is used for seperation of tokens. In order for the compiler understand the different tokens, there should be at least one space or tab character be exist between them. Also, white space is used for seperation of code lines. The compiler looks for new line characters in order to understand the end of the code line. Code lines are important in SAPL as the parser creates the parse tree for each code line.

4) Defining directed graphs:

In order to define a directed graph programmer should write "Directed" word which defines the type and after leaving a space programmer should provide a variable name for the graph. The variable name of the graph is used for accessing modifying graph and running queries on the graph later on.

An example code is given below:

```
beginmain
    Directed myGraph
endmain
```

5) Defining undirected graphs:

In order to define an undirected graph programmer should write "Undirected" word which defines the type and after leaving a space programmer should provide a variable name for the graph. The variable name of the graph is used for accessing modifying graph and running queries on the graph later on.

An example code is given below:

```
beginmain
    Undirected myGraph
Endmain
```

6) Properties:

In SAPL properties are name value pairs. In order to define a property programmer should write "Property" in the begining of the line. After defining the type as property, programmer should write a variable name for the property by leaving a space between the type and the name. After this point, in order to assign a value to the property programmer has to use "=" sign after the name of the property. There are two possible types of values that can be assigned to properties. Each of them are name, value pairs.

1. The first type consists of a string as name and an integer as value.

   An example is given below:

   ("id", 1234)

2. The second type consists of a string as name and a string as value.

   An example is given below:

   ("Ali", "Veli")

7) Adding vertices and defining vertex properties:

In order to add a vertex, programmer has to define a vertex first. For defining a vertex, programmer should specify the type as "Vertex" and specify a name for the vertex by leaving a space between the type and the name. In addition to this, programmer has to specify at least one property for the vertex. In order to specify a property for the vertex, programmer can write the name of a predefined property in the code block inside paranthesis or can directly write name value pair in paranthesis.

In order to add a vertex to a graph, programmer should first write the name of the graph and then should write ".AddVertex" string. Also, programmer has to write the name of the vertex inside paranthesis.

Example codes are given below:

```
Vertex myVertex("ali, veli")
myGraph.AddVertex(myVertex)
Property myProperty = ("id", 1234)
Vertex myVertex2(myProperty)
myGraph.AddVertex(myProperty)
```

In addition to those above, programmer can attach multiple properties to a vertex. This can be done by simply writing ".AddProperty" after the name of the graph, writing the and specifying the property inside paranthesis.

An example code is given below:

```
Vertex myVertex("ali, veli")
myVertex.AddProperty("id", 1234)
myVertex.AddProperty("cID", 4321)
Property myProperty = ("examplename", "examplevalue")
myVertex.AddProperty(myProperty)
```

8) Defining edges between vertices and attaching properties to the edges.

In order to define an edge, programmer should specify the type at the begining of a line by writing "Edge". After that, programmer should specify the name of the edge by leaving a space between the type and the name. At that point, programmers are free to make a choice between attaching properties on the edge or not. If programmer does not want to attach a property, then the edge creation is finished at this point.

An example code is given below:

```
Edge myEdge
```

On the other hand, if programmer wants to add properties to edges, then, they might first create a Property and attach it by simply using ".AttachProperty" function or he/she can directly attach it by using ".AttachProperty" function and defining the name value pair inside paranthesis.

An example code is given below:

```
Edge myEdge
Property myProperty = ("id", 1234)
myEdge.AttachProperty(myProperty)
myEdge.AttachProperty("ali", "veli")
```

In this case, it is a fact that more than one property can be assigned to an edge.

Programmer can connect to vertices by simply writing the name of the edge at the begining of the line and then writing ".ConnectTo". Then, programmer should specify the vertices inside paranthesis with a comma seperation.

An example is given below:

```
myEdge.ConnectTo(vertex1, vertex2)
```

In this case, when vertices are parts of a directed graph, the direction of the edge is from the left vertex to the right one with respect to the order of the writing inside paranthesis. In this example, the direction is from vertex1 to vertex2. When the graph is an undirected graph the order does not matter.

9) Strings

In SAPL, programmer should write strings inside "" marks. In addition to this, programmer can also define string values in order to use tham later. This simply can be done by writing the type as "String" in the beginning of the line and writing the name of the variable by leaving a space between the type and the name. After definig the string variable, programmer can initialize the value or just leave it uninitialized.

An example code is given below:

```
myString2 = "mesela"
```

In addition to these, a string value can be assigned to another string after defining a string in the code block.

An example code is given below:

```
myString1 = “Ali”
myString2 = myString1
```

10) Integer

In SAPL, programmer can write integers directly without using any special characters around them. In addition to this, programmer can also define integer values in order to use tham later. This simply can be done by writing the type as "Integer" in the beginning of the line and writing the name of the variable by leaving a space between the type and the name. After definig the integer variable, programmer can initialize the value or just leave it uninitialized.

An example code is given below:

```
myInteger2 = 1234
```

In addition to these, an integer value can be assigned to another integer after defining a integer in the code block.

An example code is given below:

```
myInteger1 = 123
myInteger2 = myInteger1
```

11) Float

In SAPL, programmer can write floating numbers directly without using any special characters around them. In addition to this, programmer can also define float values in order to use tham later. This simply can be done by writing the type as "Float" in the beginning of the line and writing the name of the variable by leaving a space between the type and the name. After definig the float variable, programmer can initialize the value or just leave it uninitialized.

An example code is given below:

```
myFloat2 = 1234.1234
```

In addition to these, a float value can be assigned to another floating number after defining a float in the code block.

An example code is given below:

```
myFloat1 = 123.123
myFloat2 = myFloat1
```

12) Lists

In SAPL, programmer can create lists of values by directly writing “List” at the begining of the line. Then programmer should write a name for the variable by leaving one space between the type and the name of the value. Programmer can initialize the list directly by writing equals sign after stating the name and then ‘{‘ character should be written. This character states the begining of the list and ‘}’ character states the ending of the list. Also, in lists, values should be seperated from each other by putting a comma between them. In lists any type that is supported by the SAPL can be hold.

A sample code is given below;

```
List alist = {123, 1234, “example list string value”}
```

Programmer can access the values in the list directly by writing the list's name and stating the index of the value that he/she wants to reach inside '[' and ']' characters.

An example code is given below;

```
alist[0]
```

Programmer can use the values directly or can assign the value to another instance. Also, programmer can change the value directly putting equals to sign after accessing the value.

An example code is given below:

```
alist[0]=12
```

Programmer can add new items to the list by simply writing ".Add" after the name of the list and new value should be specified between paranthesis.

An example code is given below:

```
alist.Add(13)
```

Programmer can delete values from list by simply writing ".Delete" after the name of the list and the value of the item to be deleted should be specified between paranthesis.

An example code is given below:

```
alist.Delete(13)
```

13) Maps

In SAPL, programmer can create maps of value-key pairs by directly writing "Map" at the begining of the line. The programmer should write a name for the variable by leaving one space between the type and the name of the value. Programmer can initialize the map directly by writing equals sign after stating the name and then '{' character should be written. This character states the begining of the list and '}' character states the ending of the

list. In order to add a key-value pair, programmer should write them in between paranthesis with a comma seperation between them. Also, items in the map should be seperated by commas. In maps any type of key value pair can be hold which means keys can be any type that is supported by the SAPL and for values the same is valid.

A sample code is given below:

```
Map amap = {(2, "two"), ("five", 5)}
```

Programmer can access the values in the map directly by writing the map's name and stating the key of the value that he/she wants to reach inside '[' and ']' characters.

An example code is given below;

```
amap[2]
```

Programmer can use the values directly or can assign the value to another instance. Also, programmer can change the value directly putting equals to sign after accessing the value.

An example code is given below:

```
amap["five"]=12
```

Programmer can add new items to the map by simply writing ".Add" after the name of the list and new item should be specified between paranthesis.

An example code is given below:

```
amap.Add(13, "thirteen")
```

Programmer can delete values from list by simply writing ".Delete" after the name of the list and the value of the item to be deleted should be specified between paranthesis.

An example code is given below:

```
amap.Delete(13)
```

14) Sets

In SAPL, programmer can create sets of values by directly writing "Set" at the begining of the line. The programmer should write a name for the variable by leaving one space between the type and the name of the value. Programmer can initialize the set directly by writing equals sign after stating the name and then '{' character should be written. This character states the begining of the list and '}' character states the ending of the list. In order to add a value, programmer should write values with a comma seperation between them.

A sample code is given below:

```
Set aset = {12, 13, 14, 15}
```

Programmer can add new values to sets by simply writing ".Add" after the name of the list and new item should be specified between paranthesis.

An example code is given below:

```
aset.Add(16)
```

Programmer can delete values from set by simply writing ".Delete" after the name of the set and the value of the item to be deleted should be specified between paranthesis.

An example code is given below:

```
aset.Delete(13)
```

15) Regular Path Queries:

In SAPL, programmer define regular path queries by directly writing "Path" at the begining of the line. The programmer should write the name for the variable by leaving one space between the type and the name of the value. Programmer can initialize the path directly by writing equals sign after stating the name. Then programmer should specify each edges property on the way within paranthesis as a name value pair. If not specified,

paranthesis still must be written in order to tell the SAPL to move one step forward and every possibility is accepted on the road. The direction of the path should be specified with "->" characters.

An example code is given below:

```
Path apath = ()->("edge1", 1245)>()->()->()->("edge2", "value of edge2")
```

Programmer can concatinate two paths by simply writing a path value after the equals sign.

An example code is given below:

```
Path bpath = apath->()->("edge3", 3)
```

Programmer can alternate the way of the path by simply putting the parts of the path in between paranthesis and putting "|" character between them.

An example code is given below:

```
Path cpath = (apath|bpath)->()->("edge4", 123)
```

Programmer define a repetition in the path definition by simply putting "*" character after the part that ment to be repeated. This sign means repetation of the part before it is allowed.

An example code is given below:

```
Path dpath = ((apath*|bpath*)->()->("edge5",1234))*
```

Programmer can use boolean expressions in the path definitions. Programmer can check the existance of a property name in the edge by writing "Contains:" and writing the name to be search in the edge properties. Also, programmer can check the non-existance of the property name by writing "!Contains: " and writing the name to be searched in the edge properties.

An example code is given below:

```
Path epath = (Contains: "edge1")->(!Contains: "edge1")
```

Programmer can compare the values in the edges and specify paths with respect to the result of the comparisons. These comparions are made with comparator operators which are ">"(for greater than), "<"(for less than), "<="(for less than or equals to), "<="(for less than or equals to), "=="(for equals to) and "!="(for not equals to). In order to compare name and values of properties seperately, programmer should write "name" or "value" stating the comparison is made for names of properties or values of properties.

An example code is given below:

```
Path fpath = (name=="edge1")->(value>13)*->()->("edge3", 123)
```

Also, characted comparison in strings are supported by SAPL language. This is done by simply adding "[" and "]" character after "Contains" function. Starting from the 0, programmer can compare a character inside a name of a property in an edge.

An example code is given below:

```
Path gpath = (Contains[0]: "e")*
```

Programmer can, define new variables on queries. This is simply done by creating values inside paranthesis.

An example code is given below:

```
Path jpath = (myname=name)->(name==myname)
```

In this case, myname is assigned to name of the first vertex and it is used on the second vertex on the path.

Programmer can use edges' properties in definition of path queries. This is basically done by writing conditions in between "-" and ">" characters.

An example is given below:

```
Path hpath = (name=="ali")-("alitoayşe", 1)>("ayşe", 2)
```

16) Comments

In SAPL, comments are specified with “//” characters. When the compiler sees these characters it assumes that the rest of the line is comment.