# Problem Description

Name: C++ Switch Object

Description: The following code, written in Smalltalk, emulates the features of a C++ switch statement. It is able to be populated with cases that call a sequence of statements that they have been matched to when evoked.

Output: The expression that is related to the case called or the default expression.

# Class Description

Name: CSwitch

Instance Variables: aSwitch, default

Description: Represents a C++ Switch statement.

Name: aSwitch

Type: Dictionary

Description: Represents the body of the switch statement, excluding the default statement. The keys of the dictionary are considered cases and their values are the statements they correspond to.

Name: default

Type: blockClosure

Description: This is the default statement that is called whenever a case is called for that does not exist. It can be made nil, but that is not advised.

# Message Descriptions

Name: new

Description: Creates a new instance of CSwitch, initializes its instance variables, and returns the object.

Name: init

Description: Initializes the instance variables.

Name: printOn

Arguments: stream

Description: Overrides Object's printOn message. This was overridden to make it easier to see the default of the switch.

Name: setDefault

Argument: aBlock

Description: Sets the default variable.

Name: add: block: break:

Arguments: case, aBlock, aBool

Description: Adds a case to aSwitch by creating a case object, initializing it with the given arguments, and then adding to the list.

Name: addAll: block: break:

Arguments: cases, aBlock, aBool

Description: Adds a list of cases to the aSwitch object as a list. Attached to that list is a single block and optional break.

Name: switch

Arguments: expression

Description: Evaluates the expression and iterates through the aSwitch object until the case equivalent to the expression is found. Otherwise, the default statement is returned. If the default statement was not initialized, then nothing is given as output.

Name: aSwitch

Description: Returns the aSwitch object. This is really only for debugging purposes or if you need to see the cases.

Name: default

Description: Returns the default object. This is really only for debugging purposes or if you need to see the default case.

# Class Description

Name: Case

Instance Variables: break, constants, block

Description: Represents a case or a series of cases.

Name: break

Type: Boolean

Description: Represents the break statement. If set to true, then it represents there being a break statement.

Name: constants

Type: OrderedCollection

Description: Represent the case or cases attached to the block.

Name: block

Type: BlockClosure

Description: The set of statements to be associated with the cases.

# Message Descriptions

Name: new

Description: Creates a new instance of Case, initializes its instance variables, and returns the object.

Name: init

Description: Initializes the instance variables.

Name: printOn

Arguments: stream

Description: Overrides Object's printOn message. This was overridden to make it easier to see the default of the switch.

Name: add: block: break:

Arguments: case, aBlock, aBool

Description: Basically initializes a case object. The case argument is the constant, while the aBlock argument and aBool arguments represent the code and break statement respectively.

Name: addAll: block: break:

Arguments: cases, aBlock, aBool

Description: Add a list of constants to create a case object containing them before adding the case to the end of the aSwitch object.

Name: hasConstant

Argument: expression

Description: Returns whether the case object has a constant that is equal to the given expression.

Name: hasBreak

Description: Returns whether this case has a break associated with it.

Name: block

Description: Returns the block associated with the case.

Name: constants

Description: Returns the list of constants. For debugging purposes only.

# Tests

### Test Plan

The switch statement is tested both inside and outside of loops to ensure that the correct value is outputted. Its persistence across the workspace life is checked next. Finally, tests to ensure that fall through is available are given.

### Settings

The code was written and tested using GNU Smalltalk with the Linux OS.

To use CSwitch, import the file using “FileStream fileIn:’MySwitch.st’”. Next, create an object by using the new message followed by the init message. After that, all other messages can be used.

### Input and Results

switch(grade)

{

case 'A' :

cout << "Excellent!" << endl;

case 'B' :

case 'C' :

cout << "Well done" << endl;

break;

case 'D' :

cout << "You passed" << endl;

break;

case 'F' :

cout << "Better try again" << endl;

default :

cout << "Invalid grade" << endl;

}

st> FileStream fileIn: 'MySwitch.st'

st> s := CSwitch new . s init .

st> s add:'A' block:['Excellent!' print] break:false .

st> s addAll:#('B' 'C') block:['Well done' print] break: true .

st> s add:'D' block:['You passed' print] break:true .

st> s add:'F' block:['Better luck next time!' print] break:false

st> s setDefault:'This is a fall through!' .

st> s switch:'A' .

'Excellent!''Well done'

st> s switch:'D' .

'You passed'

st> s switch:'F' .

'Better luck next time!''This is a fall through!'

for (int i=1;i<=4; i++){

switch(i^2) {

case 1:

cout << "1" << endl;

break;

case 4:

cout << "2 squared" << endl;

break;

case 9:

cout << "3 squared" << endl;

}

}

st> s init .

st> s add:1 block:['1' printNl] break:true .

st> s add:4 block:['2 squared' printNl] break:true .

st> s add:9 block:['3 squared' printNl] break:false .

st> s switch:10 .

st> <blank>

st> s switch:9 .

'3 squared'

st> (1 to: 4) do: [:i | s switch:(i squared)]

'1'

'2 squared'

'3 squared'