A Flowchart Language Converter

A GRADUATE PROJECT

Submitted To the Faculty of
The Department of Computing and Mathematical Sciences
Texas A&M University-Corpus Christi

In Partial Fulfillment of the Requirements for the Degree of
MASTER of SCIENCE in COMPUTER SCIENCE

By

Catherine Chou Ha Chao

Summer, 2001

COMMITTEE MEMBERS

Dr. Holly Patterson-McNeill, Chairperson
Dr. R. Stephen Dannelly, Member
Dr. Michelle Moore, Member
Abstract

This project is the design and the implementation of a flowchart language converter. It translates the elements of a flowchart into C++ statements. The converter will be used to teach introductory programming courses in the computer science academic program at Texas A&M University -- Corpus Christi.
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>ii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>iii</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>C+ Flowchart Converter</td>
<td>3</td>
</tr>
<tr>
<td>Environment</td>
<td>12</td>
</tr>
<tr>
<td>System Design</td>
<td>13</td>
</tr>
<tr>
<td>The C+ Graphical User Interface</td>
<td>24</td>
</tr>
<tr>
<td>Implementation of the Syntax Checking on Source Information</td>
<td>29</td>
</tr>
<tr>
<td>Implementation of the Flowchart Display</td>
<td>32</td>
</tr>
<tr>
<td>Implementation of the Conversion to C++ Statements</td>
<td>34</td>
</tr>
<tr>
<td>Implementation of Miscellaneous Functions</td>
<td></td>
</tr>
<tr>
<td>Results</td>
<td>38</td>
</tr>
<tr>
<td>Bibliography</td>
<td>39</td>
</tr>
<tr>
<td>Appendices</td>
<td>40</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1  The Interface of the C+ Flowchart Converter.......................... 4
Figure 2  The Interface of the Help File........................................... 5
Figure 3  Error Message with No Variable Declared.......................... 6
Figure 4  Error Message with Undeclared Variable.......................... 6
Figure 5  START............................................................................. 6
Figure 6  END.............................................................................. 6
Figure 7  I/O................................................................................. 7
Figure 8  Process........................................................................... 7
Figure 9  IF-THEN-ELSE Structure............................................... 8
Figure 10 IF-THEN-ELSE Structure Example.................................... 8
Figure 11 WHILE-DO Structure..................................................... 10
Figure 12 WHILE-DO Structure Example.......................................... 10
Figure 13 C+ Flowchart Converter.................................................. 13
Figure 14 FILE Menu................................................................. 14
Figure 15 EDIT Menu...................................................................... 14
Figure 16 DEBUG Menu............................................................... 15
Figure 17 HELP Menu................................................................. 15
Figure 18 Graphical Toolbar.......................................................... 16
Figure 19 Declaration Toolbar........................................................ 17
Figure 20 Drawing Area and Step List Box...................................... 18
Figure 21 Process Form............................................................... 19
Figure 22  IO Form................................................................. 19
Figure 23  IfThenElseCombo Form........................................... 20
Figure 24  Add Form................................................................. 21
Figure 25  IfThenElse Form...................................................... 21
Figure 26  WhileDoCombo Form................................................ 22
Figure 27  WhileDo Form........................................................... 23
Figure 28  OutputWindow Form............................................... 24
INTRODUCTION

Flowcharts are often used as suggested teaching tools for introductory programming courses. It is one type of graphical representation that can help beginning programmers visualize the logic flow of a routine and/or show the interrelationship of processes within a program. According to Robert St. Amant, Henry Lieberman, Richard Potter, and Luke Zettlemoyer, “visual information can supplement the information available from other sources, suggesting new kinds of generalizations not possible from application data alone. In addition, these generalizations can map more closely to user intentions, especially beginning users, who rely on the same visual information when making selections.” [9]

In the past, professors would simply draw a flowchart on the board to help students understand the semantics of a process or a program. However, this process is time consuming. For instance, the professors had to spend too much time drawing the symbols, writing the information inside the symbols, and then explaining the process to the students. Consequently, professors tend to use software to create flowcharts, because of the ease and the speed of producing a meaningful flowchart instead of drawing the same flowchart on the board. For example, software can actually generate a flowchart from a particular syntax-corrected programming language. However, further development of the software is needed to fine-tune a flowchart [7].

There are several commercial software programs that are based on a graphical programming environment. AUTHORWARE provides the tools for creating multimedia pieces that use digital movies, sound, animations, text, and graphics to engage the user in
the learning process. In AUTHORWARE, the user selects special icons similar to flowchart symbols to build the application [8]. Although it is powerful, it is an expensive way to build application.

LABVIEW is a graphical programming development environment based on the G programming language for data acquisition and control, data analysis, and data presentation produced by National Instruments [2]. Here again, applications are created using icons instead of conventional, text-based code.

Although these software packages are powerful tools and are good teaching tools, they are still expensive to purchase. Also, they do not perform exactly as the faculty need for teaching C++ programming at Texas A&M University -- Corpus Christi. The primary goal of this graduate project is to design and implement an iconized program that converts a flowchart to the C++ programming language.
C+ FLOWCHART CONVERTER

This project developed a converter that translates flowchart symbols into equivalent statements from a subset of the C++ programming language. Since the converter does not translate statements to the full C++ programming language, this procedural converter is called C+. The reason for choosing the C++ programming language is that it is being taught in the beginning programming courses at Texas A&M University – Corpus Christi.

The C+ flowchart converter provides an introduction to flowchart and programming concepts. The C+ converter is implemented in the windows environment. It displays pop-up dialog boxes for the user to enter source information and draw the flowchart, with drop-down menus for converting the flowchart to the C++ statements.

The implementation of the C+ flowchart converter is in Visual Basic. It includes a menu toolbar, a graphical toolbar, a declaration toolbar, a drawing area, and a step list box. Figure 1 shows the interface of the C+ converter. It also includes a help file that provides information about how to use the converter and the C+ control structures. Figure 2 shows the interface of the C+ flowchart converter help file.
Figure 1. The Interface of the C+ Flowchart Converter
The C+ flowchart converter displays informative error messages along with constructing a flowchart. For example, before developing a flowchart, the user must declare the variables that will be used. Figure 3 shows the error message if the user did not declare a variable before developing a flowchart. During the construction of a flowchart, the user can use only the variables that have already been declared. Figure 4 shows the error message if the user tries to use a variable that has not been declared.
The C+ flowchart converter processes six types of flowchart symbols. They are START-TERMINATOR, END-TERMINATOR, I/O, PROCESS, IF-THEN-ELSE, and WHILE-DO.

The START and END flowchart symbols are the delimiters, indicating the beginning and end of a flowchart, and generate no statements. When the user activates either the START-TERMINATOR or the END-TERMINATOR symbol, it is drawn directly to the drawing area. Figure 5 and Figure 6 show the START and END symbols.
The I/O flowchart symbol accepts input data and output data. Figure 7 shows the I/O symbol.

![Figure 7. I/O](image)

The PROCESS flowchart symbol accepts one or more assignment-statements and most of the C++ library functions (Appendix D). Figure 8 shows the PROCESS symbol.

![Figure 8. PROCESS](image)

There are two forms of IF-STATEMENT in C++: the IF-THEN form and the IF-THEN-ELSE form. The C+ flowchart converter has only the IF-THEN-ELSE structure form. If the value of the expression in the decision flowchart symbol is true, the expression in the THEN process flowchart symbol will be executed. If the value of the expression is false, the expression in the ELSE process flowchart symbol will be executed. The user may simulate an IF-THEN structure by leaving the ELSE process flowchart symbol box empty. Figure 9 shows the IF-THEN-ELSE structure.
Figure 9. IF-THEN-ELSE Structure

Figure 10 is an example that shows the flow of control of the IF-THEN-ELSE structure “If the hours are less than or equal to 40.00, compute the regular pay. But if the hours are greater than 40, compute the regular pay and the overtime pay.”

Figure 10. IF-THEN-ELSE Structure Example
The following code fragment will be generated when the user activates the RUN button.

```java
if (Hour <= 40.0)
    Pay = Rate * Hour;
else
    Pay = Rate * 40.0 + Rate * (Hour - 40.0) * 1.5;
```

In C++, the WHILE-STATEMENT is a looping control structure. If the expression is true, the entire sequence of statements is executed, and then the expression is checked again. If the expression is still true, the statements are executed again. The cycle continues until the expression becomes false. The C+ flowchart converter has the same structure as C++. A sequence of statements executes repeatedly as long as the condition is met.

The WHILE-DO structure flowchart symbol accepts one or more comparisons in the WHILE-TEST symbol. If the value of the comparisons in the WHILE-TEST is true, the expressions in DO-ACTION will be executed. The comparisons will be checked again, and if the value of the comparison in the WHILE-TEST is still true, the expressions in DO-ACTION will be executed again. If the value of the comparisons in WHILE-TEST is false, the expressions in EXIT-ACTION will be executed. Figure 11 shows the WHILE-DO structure.
Figure 11. WHILE-DO Structure

Figure 12 is an example that shows the flow of control of the WHILE-DO structure "Sum the data set. A value > 10 is the sentinel."

Figure 12. WHILE-DO Structure Example
The following code fragment will be generated when the user activates the RUN button.

```
Sum = 0;
While (Number < 10)
{
    Sum = Sum + Number;
}
```

cout << Sum;

Notice that this is an infinite loop. Since C+ is just a converter, it will not check for logic errors.

An IF-THEN-ELSE structure flowchart symbol or a WHILE-DO structure flowchart symbol may be nested once inside the process flowchart symbol for either the IF-THEN-ELSE structure flowchart symbol or the WHILE-DO structure flowchart symbol.
ENVIRONMENT

The C+ flowchart converter is designed to operate on a computer with a Pentium processor operating at 300MHz or faster, 64MB RAM, and 2GB of hard disk. The operating system is Windows 98/2000 with Visual Basic installed.

The C+ interface and the C+ flowchart converter are implemented in Microsoft Visual Basic 6.0 for 32-bit Windows Development. The help file is implemented in Microsoft Help Workshop.
SYSTEM DESIGN

The development of the C+ flowchart converter had five steps. They were the Graphical User Interface (GUI) that accepts source information, the implementation of the syntax checking on source information, the implementation of the flowchart display using the Visual Basic graphics methods, the implementation of the conversion to C++ statements and the implementation of the miscellaneous functions.

![Diagram of C+ Flowchart Converter]

Figure 13. C+ Flowchart Converter

The C+ Graphical User Interface

The C+ flowchart converter contains a ConverterModule and nine forms. The nine forms are Converter, Process, IO, IfThenElseCombo, Add, IfThenElse, WhileDoCombo, WhileDo, and OutputWindow.

Converter form

The Converter form is the form that is displayed when the user uses the C+ flowchart converter. The Converter form contains five objects. They are the menu toolbar, the graphical toolbar, the declaration toolbar, the drawing area, and the step list box.
On the menu bar, there is a drop-down FILE menu that contains the options: NEW, OPEN, SAVE, SAVE AS, PRINT, and EXIT. Figure 14 shows the appearance of the FILE menu.

Figure 14. FILE Menu

There is a drop-down EDIT menu that contains the options to CUT, COPY, or PASTE text. Figure 15 shows the appearance of the EDIT menu.

Figure 15. EDIT Menu
There is a drop-down DEBUG menu that contains a RUN button that runs the program. Figure 16 shows the appearance of the DEBUG menu.

![Figure 16. DEBUG Menu](image)

There is a drop-down HELP menu that contains an INDEX button. When the INDEX button is activated, the help file displays instructions on how to use the converter and some of the C++ terminology that are included in the C+ flowchart converter. Figure 17 shows the appearance of the HELP menu.

![Figure 17. HELP Menu](image)
Beneath the menu toolbar, the graphical toolbar contains graphical interface buttons that corresponds to some items in the menu toolbar. The toolbar is divided into three units—the file operations, editing operations, and flowchart symbols. The file operations toolbar contains a shortcut NEW BLANK DOCUMENT button, a shortcut OPEN DOCUMENT button, and a shortcut SAVE DOCUMENT button. The editing operations toolbar contains a shortcut CUT button, a shortcut COPY button, and a shortcut PASTE button. The flowchart symbols toolbar contains a START-TERMINATOR flowchart button, a PROCESS flowchart button, an I/O flowchart button, an IF-THEN-ELSE structure flowchart button, a WHILE-DO structure flowchart button, and an END-TERMINATOR flowchart button. Figure 18 shows the appearance of the graphical toolbar.

![Graphical Toolbar](image)

**Figure 18. Graphical Toolbar**

Beneath the graphical toolbar, there is a declaration toolbar consisting of a text box that accepts the entry of a variable name that user wants to declare, a data type list that allows the user to choose the data type of the declared variable (Appendix A), and an “Add” pushbutton that adds the declared variable to the variable array and the step list box. Figure 19 shows the appearance of the declaration toolbar.
Beneath the declaration box, there is a drawing area and a step list box. The drawing area is where the flowchart is drawn when the C+ flowchart symbol is activated and the form is submitted. The name of the flowchart symbol is then added to the step list box. Figure 20 shows the appearance of the drawing area and step list box with a sample flowchart.
Figure 20. Drawing Area and Step List Box

Process form

The process form contains a text box titled “Action” that accepts one or more statements, a pushbutton labeled “Submit” that submits the information in the text box, and a pushbutton labeled “Clear” that clears the text box. Figure 21 shows the appearance of the Process form.
I/O form

The I/O form contains a text box titled “Output” that accepts one or more variable names for an output statement, a text box titled “Input” that accepts one or more variable names for input statement, a “Submit” pushbutton and a “Clear” pushbutton. Figure 21 shows the appearance of the IO form.

IfThenElseCombo form

The IfThenElseCombo form contains a text box titled “IF-TEST” that accepts condition to be tested, a text box titled “THEN-ACTION” that accepts one or more
actions that will execute if the condition is true and an “Add” pushbutton that displays the Add form for the user to add another structure inside the THEN structure, a text box titled “ELSE-ACTION” that accepts one or more actions that will execute if the condition is false and an “Add” pushbutton that displays the Add form for the user to add another structure inside the ELSE structure. Figure 23 shows the appearance of the IfThenElseCombo form.

![Figure 23. IfThenElseCombo form](image)

Add form

The Add form contains three pushbuttons named “Add I/O structure”, “Add If-Then-Else structure”, and “Add While-Do structure”, respectively. The I/O form is displayed if the “Add I/O structure” pushbutton is activated. The IfThenElse form is displayed if the “Add If-Then-Else structure” pushbutton is activated. The WhileDo form is displayed if the “Add While-Do structure” pushbutton is activated. The actions associated with the Add form are the THEN-ACTION and the ELSE-ACTION in the IfThenElseCombo.
form, and the DO-ACTION and the EXIT-ACTION in the WhileDoCombo form. Figure 24 shows the appearance of the Add form.

![Add Form](image)

**Figure 24. Add Form**

**IfThenElse form**

When the user tries to add a nested If-Then-Else structure by activating the “Add If-Then-Else structure” pushbutton on the Add form, the IfThenElse form is displayed. The IfThenElse form is similar to the IfThenElseCombo form without the “Add” pushbuttons since the user may not add another structure. Figure 25 shows the appearance of the IfThenElse form.

![IfThenElse form](image)

**Figure 25. IfThenElse form**
WhileDoCombo form

The WhileDoCombo form has a similar interface as the IfThenElseCombo form. The text boxes have different titles. The titles are “WHILE-TEST”, “DO-ACTION”, and “EXIT-ACTION”. Figure 26 shows the appearance of the WhileDoCombo form. Again the “Add” button indicated that the user may nest a structure within the DO-ACTION or EXIT-ACTION clauses.

Figure 26. WhileDoCombo form

WhileDo form

The WhileDo form has a similar interface as the IfThenElse form with different title, just as the WhileDoCombo form. Figure 27 shows the appearance of the WhileDo form.
OutputWindow form

The OutputWindow form contains a text box that displays the converted C++ statements and a "Close" pushbutton that closes the OutputWindow. Figure 28 shows the appearance of the OutputWindow form from the flowchart of Figure 20.
Implementation of the Syntax Checking on Source Information

Syntax checking on source information entered by the user is implemented in the ConverterModule of the C+ flowchart converter. The ConverterModule contains arrays to aid in syntax checking and code generation. The DTList array stores all the data types that are included in the C+ flowchart converter, a set of arrays for the reserved words
(Appendix B), four arrays that store all the relational operators, the logical operators, the additional operators, and the multiplication operators according to the C++ operators precedence (Appendix C), two arrays that store both the increment and the decrement operators and the compound assignment operators (Appendix D), and the four library function arrays that are included in the ctype.h header file, the math.h header file, the stdlib.h header file, and the string.h header file.

In the C+ flowchart converter, all actions and the I/O statements can accept one or more statements. For all statements, no semicolon is required at the end of the client data; however, a semicolon is required between multiple statements. The following code fragment shows how the converter decodes multiple statements and stores them to an array.

\[
\text{If \text{InStr}(1, \text{text}, \text{";"}, 1) > 0 \text{ Then}} \\
\text{newtext = Split(text, \text{";"})}
\]

All client data with leading and/or trailing spaces is stripped by using the built-in \text{Trim} function.
\[
\text{text = Trim(text)}
\]

There are five basic types of checking in the modules. They are the \text{Declaration\_Check}, the \text{VarName\_Check}, the \text{Test\_Check}, \text{Action\_Check}, and the \text{I/O\_Check}. 

25
Declaration_Choose

Whenever the user declares a variable name and type by activating the Add pushbutton, the converter increments the index VarItemsCnt by one. This function checks if the VarItemsCnt is equal to zero; if the VarItemsCnt is equal to zero, a message box is displayed with the “No Variable is declared” error message.

VarName_Check

In C+ flowchart converter, the user cannot declare the same variable name twice even though it has a different data type. This function checks if the variable has already been declared.

Test_Check

Test_Check includes five types of checks and four library function checks. Checks start with Test0_Check, which checks if the condition starts with an open parenthesis and ends with a close parenthesis. The following code fragment shows how Test0_Check is implemented.

```vbnet
If Left(text, 1) = "(" Then
    If Right(text, 1) = ")" Then
        newtext = Trim(Mid(text, 2, Len(text)). 2)
        If Test1_Check(newtext) = True Then
            Test0_Check = True
            Exit Function
        End If
    Else
        MsgBox ("Missing right parenthesis")
        Exit Function
    End If
ElseIf Test1_Check(text) = True Then
    Test0_Check = True
    Exit Function
End If
```
Test0_Check calls the Test1_Check to check if the condition contains the logical
operator in Test1 array. The following code fragment shows how to loop through array
Test1; if the same operator is found in Test1, found becomes true (found = True).
The split function then splits the client data using the operator as the delimiter and puts
the data into an array.

For i = 0 To UBound(Test1)
    If InStr(1, text, Test1(i), 1) > 0 Then
        op = Test1(i)
        found = True
        Exit For
    End If
Next i

If found = True Then
    newtext = Split(text, op, 1)
    For j = 0 To UBound(newtext)
        If Test0_Check(newtext(j)) = False Then
            Test1_Check = False
            Exit Function
        End If
    Next j
    Test1_Check = True
    Exit Function
ElseIf Test2_Check(text) = True Then
    Test1_Check = True
    Exit Function
End If

Test1_Check calls the Test2_Check to determine if the condition contains the
relational operator in Test2 array. Test2_Check calls the Test3_Check to determine if the
condition contains the multiplication operator in Test3 array. Test3_Check calls the
Test4_Check to determine if the condition contains the additional operator in the Test4
array. Test4_Check calls the Test5_Check to determine whether the condition is a
variable name, a number, or a library function. If the condition is not a variable name or
a number, it calls the Lib_Check function. The Lib_Check function contains the Lib_dtype_Check function, the Lib_math_Check function, the Lib_stdio_Check function, and the Lib_string_Check function.

**Action_Check**

This function checks if the user put in multiple actions and calls the Act_Check for the syntax checking. The Act_Check checks if the action is an increment or decrement statement, or a compound assignment statement, or a simple assignment statement with declared variable name through the VarName_Check function. An error message will be displayed if any of the checks is false.

**I/O_Check**

The I/O statement of C+ flowchart converter only accepts the input and the output of one or more variables. This function checks if the client data has multiple statements, then calls the VarName_Check to determine if all variables from the client data have been declared. In the following code fragment, the I/O_Check function checks for the semicolon and if it is found, splits it and puts it in an array. The VarName_Check function determines if the variables have been declared.

```pascal
If InStr(1, text, ",", 1) > 0 Then
  newtext = Split(text, ",")
  For i = 0 To UBound(newtext)
    If VarName_Check(Trim(newtext(i))) = False Then
      IO_Check = False
      Exit Function
  End If
  Next i
ElseIf VarName_Check(text) = False Then
  IO_Check = False
  Exit Function
```
End If

In addition to this module, the Converter form checks the variable name at declaration. The InvalidChar_Check determines if the variable name contains an invalid character. The InvalidChar_Check function checks for the invalid character using the Asc built-in function.

For i = 1 To Len(VN)
   InvalidChar = Mid$(VN, i, 1)
   Select Case Asc(InvalidChar)
      Case 32
         MsgBox ("Invalid character: space")
      Exit Function
      Case 33 To 47, 58 To 64, 91 To 94, 96, 123 To 126
         MsgBox ("Invalid character: " & InvalidChar)
      Exit Function
   End Select
Next i

The Number_Check determines if the variable name begins with a number, the DuplicateName_Check determines if the variable name is a duplicate variable name, and the RW_Check determines if the variable name is a C++ reserved word.

Implementation of the Flowchart Display

After the user submits all the source information and all checks return true, each form stores the client data into different arrays for future access. Since both the START-TERMINATOR and the END-TERMINATOR flowchart symbols are the delimiters, they are drawn in the drawing area without displaying a form. The START-TERMINATOR flowchart symbol is set to draw in the middle of the drawing area. The first four statements in the code below draw the START-TERMINATOR flowchart symbol. PicX
and PicY are the global variables that store the end point of each flowchart symbol. The 
CurrentX and CurrentY determine where to put the label of the TERMINATOR symbol.
The Print statement puts the word “START” inside the START-TERMINATOR symbol.

DrawingArea.Line (DrawingArea.ScaleWidth / 2. 300, 200)-
               (DrawingArea.ScaleWidth / 2 + 300, 200)
DrawingArea.Circle (DrawingArea.ScaleWidth / 2. 300, 350),
               (500. 200) / 2, , 1.6, 4.8
DrawingArea.Circle (DrawingArea.ScaleWidth / 2 + 300, 350),
               (500. 200) / 2, , 4.8, 1.8
DrawingArea.Line (DrawingArea.ScaleWidth / 2. 300, 500)-
               (DrawingArea.ScaleWidth / 2 + 300, 500)
DrawingArea.Line (DrawingArea.ScaleWidth / 2, 500)-
               (DrawingArea.ScaleWidth / 2, 700)
PicX = DrawingArea.ScaleWidth / 2
PicY = 700
DrawingArea.CurrentX = PicX. 300
DrawingArea.CurrentY = 260
DrawingArea.Print "START"

The IfThenElse and the WhileDo forms are both the child form of the
IfThenElseCombo and the WhileDoCombo forms, respectively. They only accept and
store the client data into the arrays without drawing anything to the drawing area. They
are drawn with the IfThenElseCombo form and the WhileDoCombo form when the user
submits the combo form. The following statements are selected from the Submit_Click
function from the IfThenElse form. The first statement calls the Test0_Click to check if
the client data has corrected syntax. The If statement beneath that checks which Add
button is activated; for example, if the ITEC_Eadd_ITE is true, the client data in this
IfThenElse form is added to the Else structure of the IfThenElseCombo form. The three
statements inside the If block re-declare the ITEC_Eadd_IfTest array, put the client data
in the IfTestBox into the ITEC_Eadd_ITEC array, and increment the
ITEC_Eadd_IfTestCnt by one to keep track of how many times the
ITEC_Eadd_IfTest is being called.

ElseIf Test0_Check(iFTestBox.text) = True Then
    If ITEC_Eadd_ITE = True Then
        ReDim ITEC_Eadd_IfTest(ITEC_Eadd_IfTestCnt)
        ITEC_Eadd_IfTest(ITEC_Eadd_IfTestCnt) =
            IfTestBox.text
        ITEC_Eadd_IfTestCnt = ITEC_Eadd_IfTestCnt + 1
    ElseIf WDEC_Eadd_ITE = True Then
        ReDim WDEC_Eadd_IfTest(WDEC_Eadd_IfTestCnt)
        WDEC_Eadd_IfTest(WDEC_Eadd_IfTestCnt) =
            IfTestBox.text
        WDEC_Eadd_IfTestCnt = WDEC_Eadd_IfTestCnt + 1
    ElseIf ITEC_Tadd_ITE = True Then
        ReDim ITEC_Tadd_IfTest(ITEC_Tadd_IfTestCnt)
        ITEC_Tadd_IfTest(ITEC_Tadd_IfTestCnt) =
            IfTestBox.text
        ITEC_Tadd_IfTestCnt = ITEC_Tadd_IfTestCnt + 1
    ElseIf WDEC_Dadd_ITE = True Then
        ReDim WDEC_Dadd_IfTest(WDEC_Dadd_IfTestCnt)
        WDEC_Dadd_IfTest(WDEC_Dadd_IfTestCnt) =
            IfTestBox.text
        WDEC_Dadd_IfTestCnt = WDEC_Dadd_IfTestCnt + 1
    End If

The following statements are selected from the IfThenElseCombo form. These also
checks if the Else structure of the IfThenElseCombo form has added another If-Then-Else
structure (ITEC_Eadd_ITE = True). When the check returns true, the code
calculates the length of the IfTest, sets the coordinates and prints the IfTest from the
array, and then draws the diamond for the IfTest symbol.

If ITEC_Eadd_ITE = True Then
    eadd_iflength = (Len(Trim(ITEC_Eadd_IfTest
        (ITEC_Eadd_IfTestCnt - 1))) + 2) * 80
    Converter.DrawingArea.CurrentX = PicX. eadd_iflength +
        eadd_iflength / 2
    Converter.DrawingArea.CurrentY = PicY + 50
    x = Converter.DrawingArea.CurrentX
    y = Converter.DrawingArea.CurrentY
Converter.DrawingArea.Print Trim(ITEC_Eadd_IfTest
        (ITEC_Eadd_IfTestCnt. 1))
Converter.DrawingArea.Line (PicX. eadd_iflength.
        eadd_iflength / 3, y + 100)-(PicX,
        PicY)
Converter.DrawingArea.Line.(PicX + eadd_iflength +
        eadd_iflength / 3, y + 100)
Converter.DrawingArea.Line.(PicX, y + 300)
        eadd_iflength / 3, y + 100)
        eadd_iflength, y + 100)
        eadd_iflength, y + 400)

Since both the Process form and I/O form do not have any child form, they process
the syntax checking on the client data and draw the actions or input or output statements
without submitting any other forms.

    If InStr(1, Process_Action(Process_ActionCnt. 1), ",;", 1)
        > 0 Then
        text = Split(Process_Action(Process_ActionCnt. 1),
                    ",;")
        For i = 0 To UBound(text)
            If length < Len(Trim(text(i))) Then
                length = Len(Trim(text(i)))
        End If
        Next i
    length = length * 80
    Converter.DrawingArea.CurrentY = PicY + 50
    For j = 0 To UBound(text)
        Converter.DrawingArea.CurrentX = PicX. length +
        length / 2
        Converter.DrawingArea.CurrentY =
        Converter.DrawingArea.CurrentY + 50
        Converter.DrawingArea.Print Trim(text(j)) & ",;
    Next j
Else
    length = Len(Trim(Process_Action(Process_ActionCnt.
                        1))) * 80
    Converter.DrawingArea.CurrentX = PicX. length + length
    / 2
    Converter.DrawingArea.CurrentY = PicY + 50
    Converter.DrawingArea.Print Trim(Process_Action
End If
Converter.DrawingArea.Line (PicX, 2 * length / 3, PicY) -
(PicX + 2 * length / 3,
Converter.DrawingArea.CurrentY
+ 50), , B
Converter.DrawingArea.Line (PicX,
Converter.DrawingArea.CurrentY) - (PicX,
Converter.DrawingArea.CurrentY + 200)
PicX = Converter.DrawingArea.CurrentX
PicY = Converter.DrawingArea.CurrentY
End If

**Implementation of the Conversion to C++ Statements**

After the flowchart is drawn to the drawing area including the END-TERMINATOR symbol indicating the end of the flowchart, the user can activate the Run pushbutton under the Debug menu to convert the flowchart to the C++ statements to be displayed on the OutputWindow form. The two header files iostream.h and the stddef.h are always included in the C++ output.

```
OutputWindow.Show
OutputWindow.Txt.text = "#include <iostream.h>" & vbCrLf
OutputWindow.Txt.text = OutputWindow.Txt.text & "#include <stdlib.h>" & vbCrLf
```

The header files ctype.h, the math.h, the stdlib.h and the string.h may be included if the user uses any of the library functions in those files that are allowed in the C+ flowchart converter.

```
If LCTYPE = True Then
    OutputWindow.Txt.text = "#include <ctype.h>"
ElseIf LMATH = True Then
    OutputWindow.Txt.text = "#include <math.h>"
ElseIf LSTDLIB = True Then
    OutputWindow.Txt.text = "#include <stdlib.h>"
ElseIf LSTRING = True Then
```
The main function is also included all the time; it indicates the beginning of the program execution. Although there are multiple forms of the main statement in C++, the C+ flowchart converter uses the following form (that is, no command line arguments and an integer value returned to the operating system).

A for loop is used to process all the items in the ListBoxItem array that stores all the items in the step list box. All variable name that user declared before constructing the flowchart will be added to the report. The OtherList array stores the statements that are not variable declarations. The case statement and the for loop and used to process the OtherList and detects which button was activated by the user during the construction of the flowchart. Another for loop is used to process the array that stores all client data of different form corresponding to the button activated by the user. The following code statements are selected from the case statement of the "IF-THEN-ELSE button activated" in the OtherList of the Run_Click function.

Case "If"
OutputWindow.Txt.Text = OutputWindow.Txt.Text & Space(3) & "if (" & Trim(itectext(1)) & ")" & vbCrLf

Case "Then"
OutputWindow.Txt.Text = OutputWindow.Txt.Text & Space(3) & ";" & vbCrLf
If itectext(1) <> "NoThenAction" Then
  If InStr(1, itectext(1), ";", 1) > 0 Then
    thentext = Split(itectext(1), ";")
    For e = 0 To UBound(thentext)
      OutputWindow.Txt.Text = OutputWindow.Txt.Text & Space(6) & Trim(thentext(e)) & ";" & vbCrLf
    Next e
Else

34


```
OutputWindow.Txt.text = OutputWindow.Txt.text &
    Space(6) &
    Trim(itectext(1))
    & ";" & vbCrLf
End If
End If

Case "Else"
    OutputWindow.Txt.text = OutputWindow.Txt.text &
    Space(3) & "]" & vbCrLf
    OutputWindow.Txt.text = OutputWindow.Txt.text &
    Space(3) & "else" & vbCrLf
    OutputWindow.Txt.text = OutputWindow.Txt.text &
    Space(3) & "{" & vbCrLf
If itectext(1) <> "NoElseAction" Then
    If InStr(1, itectext(1), ";", 1) > 0 Then
    elsetext = Split(itectext(1), ";")
    For e = 0 To UBound(elsetext)
        OutputWindow.Txt.text = OutputWindow.Txt.text &
        Space(6) &
        Trim(elsetext(e))
        & ";" & vbCrLf
    Next e
Else
    OutputWindow.Txt.text = OutputWindow.Txt.text &
    Space(6) &
    Trim(itectext(1))
    & ";" & vbCrLf
End If
End If
```

### Implementation of Miscellaneous Functions

The miscellaneous functions are those corresponding functions to the File New pushbutton, the File Open pushbutton, the File Save pushbutton, the Print pushbutton, the Exit pushbutton, the Cut pushbutton, the Copy pushbutton, and the Paste pushbutton.

The File New command, whether it is from the shortcut pushbutton on the graphical toolbar or under the File menu, calls the mNew_Click function. The mNew_Click function resets the data in the Variable text box and the DataTypeList and clears all data in the DrawingArea, StepListBox, OutputWindow, and all arrays.
The File Open command, whether it is from the shortcut pushbutton on the graphical toolbar or under the File menu, calls the mOpen_Click function. The mOpen_Click function loads the output to the OutputWindow from the selected text file and the flowchart to the DrawingArea from the selected flowchart file with BMP or JPEG format.

The File Save command, whether it is the shortcut pushbutton on the graphical toolbar or under the File menu, calls the mSave_Click function. The mSave_Click function checks if the file has been saved before. If it has, it saves it again. If not, it calls the ShowDialogThenSaveFile function. The ShowDialogThenSaveFile function displays a dialog box that prompts the user to put in a filename for the OutputWindow and the drawing area.

The Print command, whether it is the shortcut pushbutton on the graphical toolbar or under the File menu, calls the mPrint_Click function. The mPrint_Click function sends the output C++ statements from the OutputWindow to the printer.

The Exit pushbutton in the File menu calls the QuerySaveFile function to prompt the user about saving changes before exiting the program.

The Cut command, whether it is from the shortcut pushbutton on the graphical toolbar or under the Debug menu, calls the mCut_Click function. The mCut_Click function clears the Clipboard, selects all the text from the ActiveControl text field, copies it to the Clipboard, and then clears the ActiveControl text field.

```c
Clipboard.Clear
Clipboard.SetText ActiveControl.SelText
ActiveControl.SelText = ""
```

The Copy command, whether it is the shortcut pushbutton on the graphical toolbar or under the Debug menu, calls the mCopy_Click function. The mCopy_Click function
clears the Clipboard, selects all the text from the ActiveControl text field, and copies it to
the Clipboard.

Clipbaord.Clear
Clipboard.SetText ActiveControl.SelText

The Paste command, whether it is the shortcut pushbutton on the graphical toolbar on
under the Debug menu, calls the mPaste_Click function. The mPaste_Click function
pastes the text in the Clipboard into the selected text field.

ActiveControl.SelText = Clipboard.GetText
RESULTS

The C+ flowchart converter provides a user-friendly, visualization tool for the beginning programmer to build a flowchart and convert it to C++ statements. The whole package is contained in an executable file. The user has the options to create a new program or open an existing C+ converter file. The user is able to input a simple assignment statement, a compound assignment statement, an increment statement or a decrement statement as action, an input statement or an output statement for I/O, one level of nested If-Then-Else statement, and one level nested While-Do statement. When the user submits the form with syntactically correct statement, the corresponding flowchart symbol and data is drawn in the drawing area. After constructing the flowchart, user can activate the Run button under the Debug menu to convert the flowchart to C++ statements that are output to the OutputWindow.

The C+ flowchart converter also provides a help file that instructs the user how to use the converter and some of the C++ terminology that are included in the C+ flowchart converter.

Implementation remains for future work in the area of semantic adding more data structure such as array, pointer, and structs, adding checking such as dead code, type problem, and infinite loop, as well as adding control structure such as switch, for, do while, and literals to output.


DATA TYPE

char
short
int
long
float
double
double float
Appendix B

RESERVED WORDS

Just as in the C++ programming language, the following identifiers are reserved words. The programmer cannot declare them for other uses in the C+ converter.

<table>
<thead>
<tr>
<th>asm</th>
<th>auto</th>
<th>break</th>
</tr>
</thead>
<tbody>
<tr>
<td>case</td>
<td>catch</td>
<td>char</td>
</tr>
<tr>
<td>class</td>
<td>const</td>
<td>continue</td>
</tr>
<tr>
<td>default</td>
<td>delete</td>
<td>do</td>
</tr>
<tr>
<td>double</td>
<td>else</td>
<td>enum</td>
</tr>
<tr>
<td>extern</td>
<td>float</td>
<td>for</td>
</tr>
<tr>
<td>friend</td>
<td>goto</td>
<td>if</td>
</tr>
<tr>
<td>inline</td>
<td>int</td>
<td>long</td>
</tr>
<tr>
<td>new</td>
<td>operator</td>
<td>private</td>
</tr>
<tr>
<td>protected</td>
<td>public</td>
<td>register</td>
</tr>
<tr>
<td>return</td>
<td>short</td>
<td>signed</td>
</tr>
<tr>
<td>sizeof</td>
<td>static</td>
<td>struct</td>
</tr>
<tr>
<td>switch</td>
<td>template</td>
<td>this</td>
</tr>
<tr>
<td>throw</td>
<td>try</td>
<td>typedef</td>
</tr>
<tr>
<td>union</td>
<td>unsigned</td>
<td>virtual</td>
</tr>
<tr>
<td>void</td>
<td>volatile</td>
<td>while</td>
</tr>
<tr>
<td>Symbol</td>
<td>Type of Operation</td>
<td>Associativity</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>postfix ++</td>
<td>Expression</td>
<td>Left to right</td>
</tr>
<tr>
<td>postfix --</td>
<td></td>
<td></td>
</tr>
<tr>
<td>prefix ++</td>
<td>Unary</td>
<td>Right to left</td>
</tr>
<tr>
<td>prefix --</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* / %</td>
<td>Multiplicative</td>
<td>Left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>Additive</td>
<td>Left to right</td>
</tr>
<tr>
<td>&lt; &gt; &lt;= &gt;=</td>
<td>Relational</td>
<td>Left to right</td>
</tr>
<tr>
<td>== !=</td>
<td>Equality</td>
<td>Left to right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical-AND</td>
<td>Left to right</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= *= /= %=</td>
<td>Simple and Compound</td>
<td>Left to right</td>
</tr>
<tr>
<td>+= -=</td>
<td>Assignment</td>
<td></td>
</tr>
</tbody>
</table>

41
Appendix D

C++ LIBRARY FUNCTION

The Header File cctype.h

isalnum(ch)

Parameter: A char value ch

Function return value: An int value that is

- True: nonzero if ch is a letter or a digit character
- False: zero

isalpha(ch)

Parameter: A char value ch

Function return value: An int value that is

- True: nonzero if ch is a letter
- False: zero

iscntrl(ch)

Parameter: A char value ch

Function return value: An int value that is

- True: nonzero if ch is a control character
- False: zero
isdigit(ch)

Parameter: A char value ch
Function return value: A int value that is

- True: nonzero if ch is a digit character
- False: zero

isgraph(ch)

Parameter: A char value ch
Function return value: An int value that is

- True: nonzero if ch is a nonblank printable character
- False: zero

islower(ch)

Parameter: A char value ch
Function return value: An int value that is

- True: nonzero if ch is a lowercase letter
- False: zero
isprint(ch)

Parameter: A char value ch

Function return value: An int value that is

- True: nonzero if ch is a printable character, including the blank
- Zero

ispunct(ch)

Parameter: A char value ch

Function return value: An int value that is

- True: nonzero if ch is a punctuation character
- False: zero

isspace(ch)

Parameter: A char value ch

Function return value: An int value that is

- True: nonzero if ch a whitespace character
- False: zero
isupper(ch)

Parameter: A char value ch

Function return value: An int value that is

- True: nonzero if ch is an uppercase letter
- False: zero

isdigit(ch)

Parameter: A char value ch

Function return value: An int value that is

- True: nonzero if ch is a hexadecimal digit
- False: zero

tolower(ch)

Parameter: A char value ch

Function return value: A character that is

- the lowercase equivalent of ch, if ch is an uppercase letter
- ch, otherwise
toupper(ch)

Parameter: A char value ch
Function return value: A character that is

- the uppercase equivalent of ch, if ch is a lowercase letter
- ch, otherwise

The Header File math.h

In the following math routines,

1. Error handling for incalculable or out-of-range results is system-dependent.

2. All parameters and function return values are technically of type double.

   However, single-precision(float) values may be passed to the function.

acos(x)

Parameter: A floating point expression x, where $-1.0 \leq x \leq 1.0$
Function return value: Arc cosine of x, in the range $0.0$ through $\pi$

asin(x)

Parameter: A floating point expression x, where $-1.0 \leq x \leq 1.0$
Function return value: Arc sine of x, in the range $-\pi/2$ through $\pi/2$

atan(x)

Parameter: A floating point expression x
Function return value: Arc tangent of x, in the range $-\pi/2$ through $\pi/2$
ceil(x)

Parameter: A floating point expression x
Function return value: “Ceiling” of x

cos(angle)

Parameter: A floating point expression angle, measured in radians
Function return value: Trigonometric cosine of angle

cosh(x)

Parameter: A floating point expression x
Function return value: Hyperbolic cosine of x

exp(x)

Parameter: A floating point expression x
Function return value: The value e raised to the power x

fabs(x)

Parameter: A floating point expression x
Function return value: Absolute value of x

floor(x)

Parameter: A floating point expression x
Function return value: “Floor” of x
\( \log(x) \)

Parameter: A floating point expression \( x \), where \( x > 0.0 \)

Function return value: Natural logarithm (base \( e \)) of \( x \)

\( \log_{10}(x) \)

Parameter: A floating point expression \( x \), where \( x > 0.0 \)

Function return value: Common logarithm (base 10) of \( x \)

\( \text{pow}(x, y) \)

Parameter: Floating point expressions \( x \) and \( y \). If \( x = 0.0 \), \( y \) must be positive; if \( x \leq 0.0 \), \( y \) must be a whole number

Function return value: \( x \) raised to the power \( y \)

\( \sin(\text{angle}) \)

Parameter: A floating point expression \( \text{angle} \), measured in radians

Function return value: Trigonometric sine of \( \text{angle} \)

\( \sinh(x) \)

Parameter: A floating point expression \( x \)

Function return value: Hyperbolic sine of \( x \)
sqrt(x)

Parameter: A floating point expression x, where x \geq 0.0
Function return value: Square root of x

tan(angle)

Parameter: A floating point expression angle, measured in radians
Function return value: Trigonometric tangent of angle

tanh(x)

Parameter: A floating point expression x
Function return value: Hyperbolic tangent of x

The Header File stdlib.h

abc(i)

Parameter: An int expression i
Function return value: An int value that is the absolute value of i

atof(str)

Parameter: A string (null-terminated char array) str representing a floating point number, possibly preceded by whitespace character and a '+' or '-'
Function return value: A double value that is the floating point equivalent of the characters in str
Note: Conversion stops at the first character in str that is inappropriate for a floating point number. If no appropriate characters were found, the return value is system-dependent

atoi(str)

Parameter: A string (null-terminated char array) str representing an integer number, possibly preceded by whitespace characters and a '+' or '-'

Function return value: An int value that is the integer equivalent of the characters in str

Note: Conversion stops at the first character in str that is inappropriate for an integer number. If no appropriate characters were found, the return value is system-dependent

 atol(str)

Parameter: A string (null-terminated char array) str representing a long integer, possibly preceded by whitespace character and a '+' or '-'

Function return value: A long value that is the long integer equivalent of the characters in str

Note: Conversion stops at the first character in str that is inappropriate for a long integer number. If no appropriate characters were found, the return value is system-dependent
Exit(exitStatus)

Parameter: An int expression exitStatus

Effect: Program execution terminates immediately with all files properly closed

Function return value: None (a void function)

Note: By conversion, exitStatus is 0 to indicate normal program completion and is nonzero to indicate an abnormal termination

labs(i)

Parameter: A long expression I

Function return value: A long value that is the absolute value of I

rand()

Parameter: None

Function return value: A random int value in the range 0 through RAND_MAX, a constant defined in stdlib.h (RAND_MAX is usually the same as INT_MAX)

srand(seed)

Parameter: An int expression seed, where seed ≥ 0

Effect: Using seed, the random number generator is initialized in preparation for subsequent calls to the rand function
Function return value: None (a void function)

Note: if srand is not called before the first call to rand, a seed
value of 1 is assumed

system(str)

Parameter: A string (null-terminated char array) str representing an
operating system command, exactly as it would be typed by
a user on the operating system command line

Effect: The operating system command represented by str is
executed

Function return value: An int value that is system-dependent

The Header File string.h

strcat(tostr, fromstr)

Parameter: String (null-terminated char arrays) tostr and fromstr,
where tostr must be large enough to hold the result

Effect: fromstr, including the null character ‘\0’, is concatenated
(joined) to the end of tostr

Function return value: The base address of tostr

strcmp(str1, str2)

Parameters: Strings (null-terminated char arrays) str1 and str2

Function return value: An int value < 0, if str1 < str2 lexicographically
The int value = 0, if str1 = str2 lexicographically

An int value > 0, if str1 > str2 lexicographically

strcpy(tostr, fromstr)

Parameters: tostr is a char arrat and fromstr is a string (null-terminated char array), and tostr must be large enough to hold the result

Effect: fromstr, including the null character ‘\0’, is copied to tostr, overwriting what was there

Function return value: The base address of str

strlen(str)

Parameters: A string (null-terminated char array) str

Function return value: An int value ≥ 0 that is the length of str (excluding the ‘\0’)

Appendix E

BNF for the C+ Converter

<statement-sequence> ::= <Process-statement>  
| <IO-statement>  
| <If-Then-Else-statement>  
| <While-Do-statement>

<Process-statement> ::= <statement-sequence>

<IO-statement> ::= <input-statement>  
| <output-statement>

<If-Then-Else-statement> ::= if (<condition>)  
{  
<statement-sequence>  
}  
| if (<condition>)  
{  
<statement-sequence>  
}  
else  
{  
<statement-sequence>  
}

<While-Do-statement> ::= while(<condition>)  
{  
<statement-sequence>  
}

<statement-sequence> ::= <statement>  
| <statement><statement-sequence>

<input-statement> ::= cin >> <identifier>

<output-statement> ::= cout << <identifier>

<condition> ::= <expression><relational-op><expression>  
| <condition><logical-op><condition>

<statement> ::= <identifier>++  
| ++)<identifier>  
| <identifier>--  
| --<identifier>
<expression> ::= <expression><additional-op><term>

<additional-op> ::= <expression><relational-op><term>

<relational-op> ::= < identifier > < logical-op > < identifier >

<compound-assignment-statement> ::= <identifier><compound-op><identifier>

<simple-assignment-statement> ::= <identifier><equal-op><expression>

<term> ::= <term><multiplication-op><factor>

<factor> ::= <identifier> (expression)