1

Tracing Programs

1.1 Introduction

There are several types of errors that can crop up in programming. Compiler errors are easier to handle because they are generally typos that can be corrected. A more difficult type of error to handle is that of errors occurring during execution of the program. These errors can occur because of arithmetic exception because of improper arguments such as division by zero which produces the side effect of stopping the program. The solution is to identify where the program stopped and to include “debugging” output statements to follow the state of the variables some place before the bad calculation takes place until the the error is found.

Unintentional errors that allow the program to complete but produce the wrong solution are called logical errors. It is often difficult to identify logical errors in programs. The solution is to follow the calculations of the variables one at a time looking for the error. This task is generally made easier if a region of the program is identified first as the likely source of the problem.

Many software environments include a “debugger” option which allow you to follow the program one line at a time. The C++ compiler as many other compilers allow you to compile with such an option which when coupled with a “debugger program” allows the user to follow the program one line at a time.

Here we illustrate this process by computing a trace of the whole computation. In practice a trace is restricted to a small portion of a code. But the same principles apply.

1.1.1 Tracing Programs

C++ was discussed. The structure of the program was examined, it was compiled and ran. In this section, a programmers technique for developing good programs will be discussed. This technique is known as tracing a program. By tracing we mean that assuming some input values, we follow the program line-by-line and determine the value of each variable as they are computed. By doing this each
statement of the program is exercised and its correctness evaluated.

Consider the trace1.cpp program:

```cpp
// File trace1.cpp

#include <iostream.h>

void main()
{
    float x, z;
    int j, n;
x = 3 / 5 * 10.0;
cout << " Value of x: " << x << endl;
n = 10. / 6 * 2;
cout << " Value of n: " << n << endl;
j = 3.0 + 8.0 / 2.0 + 2;
cout << " Value of j: " << j << endl;
z = 2 * 2 * 2 * 3.;
cout << " Value of z: " << z << endl;
}
```

This program computes and prints all the variables. The actual output from this program is:

- Value of x: 0.0
- Value of n: 3
- Value of j: 9
- Value of z: 24.0

Which we could have organized as a table:

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>n</th>
<th>j</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.0</td>
<td>3</td>
<td>9</td>
<td>24.0</td>
</tr>
</tbody>
</table>

Our objective is to follow the state of the variables of the program. Let us consider the following program trace2.cpp.

```cpp
// File trace2.cpp

#include <iostream.h>

void main()
{
    int i, j;
    float a, b;
cout << "Input the values of i and j" << endl;
cin >> i >> j;
cout << "Input the values of a" << endl;
cin >> a;
i = i + j;
j = j + 2;
cout << "Values of i and j" << i <<", " << j << endl;
b = a / i;
a = a - b;
cout << "Values of a and b" << a <<", " << b << endl;
b = a / j;
a = a + b;
```
Let us consider the following inputs:

1. I = 5, J = 15, A = 5.6
2. I = 15, J = 0, A = 50.6

The actual output from this program is:

Input the values of i and j
5, 15
Input the values of a
5.6
Values of i and j 20 17
Values of a and b 5.32 .28
Values of a and b 5.63294 .3129411

for the first set of inputs and

Input the values of i and j
15, 0
Input the values of a
50.6
Values of i and j 15 2
Values of a and b 47.22666 3.37333
Values of a and b 70.83999 23.61333

for the other set of inputs. In table form, we would have constructed one row at a time. Yielding the following tables:

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>15</td>
<td>5.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.32</td>
<td>.28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.63294</td>
<td>.3129411</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>47.50.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>47.22666</td>
<td>3.37333</td>
</tr>
<tr>
<td></td>
<td></td>
<td>70.83999</td>
<td>23.61333</td>
</tr>
</tbody>
</table>

1.1.2 Exercise

Please complete the trace of the following program. Place your results on the table given in the following below. Consider the inputs to be:

4, 3
5.6
void main()
{
    int hh, nn;
    float aa, bb, cc, gg;
    hh = 10 / 6. * 2;
    nn = 3.0 + 8/2.0 + 2;
    aa = 3 / 5 * 10.0;
    cc = 2 * 2* 3 * 3.;
    cout << "Input the values of hh, nn" << endl;
    cin >> hh, nn;
    hh = hh + nn;
    nn = nn + 2;
    cout << "Input the values of aa" << endl;
    cin >> aa;
    bb = aa / hh;
    aa = aa - bb;
    gg = (aa - bb) * nn;
    cc = (bb - gg )/hh;
}

The transformation of an int constant such as 9 into a float valued constant 9. is easy to do. In order to transform and int type variable into a float type variable, the procedure float() is invoked. In the above program, float(hh) returns a float type intermediate result.
1. Tracing Programs

<table>
<thead>
<tr>
<th>Program Statements</th>
<th>hh</th>
<th>nn</th>
<th>aa</th>
<th>bb</th>
<th>cc</th>
<th>gg</th>
</tr>
</thead>
<tbody>
<tr>
<td>hh = 10 / 6. * 2;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nn = 3.0 + 8/2.0 + 2;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aa = 3 / 5 * 10.0;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cc = 2 * 2 * 3 * 3.;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cin &gt;&gt; hh &gt;&gt; nn;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hh = hh + nn;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nn = nn + 2;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cin &gt;&gt; aa ;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bb = aa / hh ;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aa = aa - bb ;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gg = (aa - bb) * nn;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cc = (bb -gg)/hh;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.1.3 Char Type

Computers store and process information in three principal forms: numerical information, logical information, and character information. The use of format specifiers in I/O is best handled when it is presented as a string of characters. Thus we arrive at another type variable that C++ supports, namely the Character string. The C++ language can processes character data in composite form, processing strings of characters rather than its individual characters.

A C++ char string is a string of characters enclosed by quotation marks.

"The world is made smaller by the WWWeb"
"ES2503 Scientific Programming"

The type char string has a property, the length, which is inseparable from its type. Char string have fixed string length which must be specified when the variable is declared.

    char fnm[10];

It is also possible to have character named constants:

    const char decltb[8] = "<table>";
    const char begrw[9] = "<tr><td>";

It is important to note that the compiler will add a null character to the string which explains why the length must be one more that specified.
An alternative is to have the length determined implicitly from the definition of the constant string.

```
char enddata[] = "</td> <td>";
```

### Trace Tables in Html with Character Variables

A C++ program can be instrumented to output its own trace in a format that can be viewed by a Web browser. To do this, the program need to open a file name for the output, “trace2.html”. It also needs to be instrumented to output the html tags. In addition to decltb, begth and enddata defined above we need:

```
char endrw[10] = "</td></tr>";
char endtb[8] "</table>";
```

In order to make the table more readable, the value of the char string statement will be set to the actual C++ statement. The enclosed program implements these changes into trace2. Try it out. Note that the border around the table is specified within the table tag.

```c
// file trace2.cpp

#include <fstream.h>
ofstream fout("trace2.html");

void main()
{
  int i, j;
  float a, b;
  char decltb[] = "<table border =4>;"
  char enddata[] = "</td> <td>";
  char begrw[] = "<tr><td>"
  char endrw[] = "</td></tr>";
  char endtb[] = "</table>";

  fout << "<title>Trace of program trace2</title>";
  fout << decltbs; // start table
  fout << begrw << "Statements" << enddata ;
  fout << "i" << enddata << "j" << enddata << "a" << enddata << "b" << endrw;

  cout << "Input the values of i and j" << endl;
  cin >> i >> j;
  cout << begrw << "cin >> i << j;" << enddata << endl;
  fout << i << enddata << j << enddata << a << enddata << b << endrw << endl;
  cout << "Input the values of a" << endl;
  cin >> a;
  fout << begrw << "cin >> a;" << enddata << endl;
  fout << i << enddata << j << enddata << a << enddata << b << endrw << endl;
  i = i + j;
 (fout << begrw << "i = i + j;" << enddata << endl;
  fout << i << enddata << j << enddata << a << enddata << b << endrw << endl;
  j = j + 2;
  fout << begrw << "j = j + 2;" << enddata << endl;
  fout << i << enddata << j << enddata << a << enddata << b << endrw << endl;
```
cout << "Values of i and j" << i<<", " << j << endl;
b = a / i ;
fout << begrw << "b = a / i ;" << enddata << endl;
fout << i << enddata << j << enddata << a << enddata << b << endrw << endl;
a = a - b;
fout << begrw << "a = a - b;" << enddata << endl;
fout << i << enddata << j << enddata << a << enddata << b << endrw << endl;
cout << "Values of a and b" << a"<<", " << b << endl;
b = a / j;
fout << begrw << "b = a / j;" << enddata << endl;
fout << i << enddata << j << enddata << a << enddata << b << endrw << endl;
a = a + b;
fout << begrw << "a = a + b;" << enddata << endl;
fout << i << enddata << j << enddata << a << enddata << b << endrw << endl;
cout << "Values of a and b" << a"<<", " << b << endl;
fout << endtb;
}

1.1.4 Comma Separated Files

Spread sheets can also produce HTML tables. Nexs is a Unix based spread sheet. It can export tables in HTML. In this section we adapt the techniques learned in the last two sections to output Comma Separated Values in a file of type .csv. The only thing we need to do differently is to output a ‘’, ‘’, between the values of the variables. Hence,

    char Comma[2] = ",";

The program trace3.cpp implements these changes. Try it out. Note that there are no HTML directives in this file. The border around the table will have to be specified later.

// file trace3.cpp

#include <fstream.h>
ofstream fout("trace3.csv");

int main()
{
    int i, j;
    float a, b;

    char Comma[2] = ",";
    // first we output the headers for the table
    fout << "Statements" << Comma ;
    fout << "i" << Comma << "j" << Comma;
    fout << "a" << Comma << "b" << endl ;

    /* after each calculation, we output the resulting values and the state
    of all the variables. */
    cout << "Input the values of i and j" << endl;
    cin >> i >> j;
    fout << "cin >> i >> j;" << Comma ;
1. Tracing Programs

```cpp
fout << i << Comma << j << Comma << a << Comma << b << endl;
cout << "Input the values of a" << endl;
cin >> a;
    fout << "cin >> a;" << Comma;
    fout << i << Comma << j << Comma << a << Comma << b << endl;
i = i + j;
    fout << "i = i + j;" << Comma;
    fout << i << Comma << j << Comma << a << Comma << b << endl;
j = j + 2;
    fout << "j = j + 2;" << Comma;
    fout << i << Comma << j << Comma << a << Comma << b << endl;
cout << "Values of i and j" << i<<", " << j << endl;
b = a / i;
    fout << "b = a / i ;" << Comma;
    fout << i << Comma << j << Comma << a << Comma << b << endl;
a = a - b;
    fout << "a = a - b;" << Comma;
    fout << i << Comma << j << Comma << a << Comma << b << endl;
cout << "Values of a and b" << a<<", " << b << endl;
b = a / j;
    fout << "b = a / j;" << Comma;
    fout << i << Comma << j << Comma << a << Comma << b << endl;
a = a + b;
    fout << "a = a + b;" << Comma;
    fout << i << Comma << j << Comma << a << Comma << b << endl;
cout << "Values of a and b" << a<<", " << b << endl;
    return 1;
```

Note that the `endl` is used once per row.

```cpp
fout << "cin >> a;" << Comma;
fout << i << Comma << j << Comma << a << Comma << b << endl;
```

It is important. The spreadsheet expects all values of a single row of the table to appear on only one row in the data file. It assumes that a new row begins with the first column.

The file `trace3.csv` is imported into Nexs as a comma separated values file. On the Nexs window, select import under the File menu. On the dialog box select comma separated values on the the menu selection button under text. Then export it as `trace3.html` file.

The resulting `trace3.html` file does not have a title. It will need to be modified to include titles and headings. The border around the table will need to be specified at this time by modifying the `<table>` tag into `<table border=4>`.

1.2 Gas reduced Pressure and Temperature

The gas reduced pressure, \( P_{r} = P/P_{sc} \), and temperature, \( T_{r} = T/T_{sc} \), where \( P \) is the pressure (psia), \( T \) is the temperature (\(^\circ\)R), \( P_{sc} \) is the pseudo-critical pressure (psia), and \( T_{sc} \) is the pseudo-critical temperature (\(^\circ\)R). The pseudo-critical pressure and temperature are given by:

\[
P_{sc} = 756.8 - 131gas_{g} - 3.6gas_{g}^{2}
\]
and 
\[ T_{sc} = 169.2 + 349.5g - 74g^2 \]

where \( g \) is the gas gravity.

The program `ReducedPT.cpp` performs the following functions:

1. Reads the pressure and temperature from a file named `PTin.txt`.
2. Prompt the user for the gas gravity, and reads it from the keyboard.
3. Computes the gas reduced pressure and temperature using the formulas above.
4. Writes the gas reduced pressure and temperature to the file `RPTout.txt`.

```cpp
// Program ReducedPT.cpp
#include <fstream.h>

ifstream PTin("PTin.txt");
ofstream RPTout("RPTout.txt");

int main()
{
    float Press, critPress, gasReducePress;
    float Temp, critTemp, gasReduceTemp;
    float GasGrvty;

    PTin >> Press >> Temp;
    cout << "Please enter the Gas Gravity" << endl;
    cin >> GasGrvty;

    critPress = 756.8 - 131*GasGrvty - 3.6*GasGrvty*GasGrvty;
    critTemp = 169.2 + 349.5 * GasGrvty - 74 *GasGrvty*GasGrvty;

    gasReducePress = Press / critPress;
    gasReduceTemp = Temp / critTemp;

    RPTout << "Pressure =" << Press << endl;
    RPTout << "Temperature =" << Temp << endl;
    RPTout << "Gas Gravity =" << GasGrvty << endl;

    RPTout << "Gas Reduced Pressure =" << gasReducePress << endl;
    RPTout << "Gas Reduced Temperature =" << gasReduceTemp << endl;

    return 1;
}
// End of file ReducedPT.cpp
```

### 1.3 Tracing a Program Project

In this project we will practice tracing a C++ program and practice making tables in `html`.
1. Change directory into your `es2503Programs` directory.

2. Copy the files `ReducedPT.cpp` and `PTin.txt`. Save them in your directory `es2503Programs`. Compile and run the program to see how it works.

3. Modify the program `ReducedPT.cpp` so it completes a trace of the program and outputs to the files `RPTtraceResults.csv`.

4. Start Nexs and import the file `RPTtraceResults.csv`. Export the file `RPTtraceResults.html`.

5. Modify the file `RPTtraceResults.html` to include a title, a heading and a border. The results of the trace should be in the form on an `html` table.

6. Once the program is working correctly, handin the `trace.cpp` and the results files `RPTtraceResults.csv` and `RPTtraceResults.html`.

7. Copy the files `PTin.txt`, `RPTout.txt` and `RPTtraceResults.html` to your `public_html/es2503` directory.

8. Change directory to your `public_html/es2503Reports` directory, and create the report file `RPTtraceReport.html` which explains the project and includes relative links to `PTin.txt`, `RPTout.txt` and `RPTtraceResults.html`.

9. Modify your `es2503.html` file to place a link to `RPTtraceReport.html`.


11. Clean up your home directory and any of your public areas to make sure that no files containing your program or portion thereof are accessible by any user.

12. Submit a Journal relating your experiences with this assignment.