Tracing & Trace Tables

Topics:

+ Trace Table Header
+ Populating a Trace Table
+ Writing a Trace Table as a List

Resources:

- `prog1.cpp`
- `prog2.cpp`
- `prog3.cpp`

Introduction

Tracing, which is the process of determining what a program is doing, is one of the most essential abilities every programmer needs to know how to do. To assist you with tracing, you can use a *trace table*. It is a table that keeps track of the changes to important components in the program for a partical instance of the program (a call). Last, it is important to note that an inidividual trace table should be constructed for each significant function.

Trace Table Header

When you are constructing a trace table, you start with the header row. It consists components of the program that are the *mutable* (changeable) structures such as variables, arrays, conditions of control structures and so on. Furthermore, it includes an output column whenever the program is displaying something, a return column whenever the program (function) is returning a value [this is be discussed more when we cover functions in more details], and the optional step column whenever you need to keep track of your steps. To practice, let us determine the header for the code below.

```
int main()
{
 int x, y = 4;
 cin >> x;
 cout << "result:  "<< x + y;
 return 0;
}
```

If your header consists of the columns *step*, *x*, *y*, *output* and *return*, you are correct. From the code, you see that there are two variables `x` and `y`; hence, there should be a column for each of them in the trace table header. Furthermore, the code is displaying a message (using a cout statement) and contains a

return statement, which means there should be a output column and return column as well. Notice that we do not include an input column. The reason for this is because inputs are stored directly into variables. Hence, when you come across an cin statement, you are just modifying the value of a variable.

Sometimes identifiers and/or conditions of control structures are long. Instead of the writing them in the header, you can use substitutions and a *legend*, which is a list that explains what substituted symbols stand for. For instance, consider the following code and trace table header.

```
int main()
{
 int numerator, denominator, remainder, whole;

 cout << "Enter numerator: ";
 cin >> numerator;
 cout << "Enter denominator: ";
 cin >> denominator;

 whole = numerator / denominator;
 remainder = numerator % denominator;
 cout << whole << " + "<< remainder << "/"<< denominator << '\n';
 return 0;
}
```

D = denominator
N = numerator
R = remainder
W = whole

step	N	D	W	R	output	return

The above header would be no different from a header with the names of the variables written out except it saves alot of space. Thus, the advantage (or need) of using a legend is to avoid the header exceeding a single line.

Construct the header of the following exercises.

Exercise 1.

```
int main()
{
 double real, imaginary;

 cout << "Enter real component: ";
 cin >> real;
 cout << "Enter imaginary component: ";
 cin >> imaginary;

 cout << real << " + "<< imaginary << "i\n";
 return 0;
}
```

Exercise 2.

```
int main()
{
 cout << "My name is John Doe\n";
 cout << "Hello World\n";
 return 0;
}
```

Exercise 3.

```
int main()
{
 int n = 7;
 n = n * n * n * n;
 cout << n << "\n";
 return 0;
}
```

Populating a Trace Table

After you construct the header of a trace table, you are ready to populate it. Whenever you are populating a trace table, there are a few things you must know. First, programs are sequential. Hence, you must begin at the beginning of the program and either work your way to the end of the body of the program or to an executable return statement. It is important to note that once a return statement is executed, the program terminates regardless of any code that follows it. Therefore, if your trace table have a return column and you populate it, your trace is over; otherwise, you continue until you make it to the end of the body of the program.

Second, only one thing changes during a single step unless either the header of the program has a list of parameters, which means, the first step will list all the values of the parameters; or, the program is executing a function call that takes reference parameters. Besides those cases, you will only be populating the step column and one other column for every row even if multiple variables are initialized on a single line. Remember initialization occurs from left to right. Furthermore, the step column does not correspond with the lines in the code; but rather, the changes in the code. Therefore, some lines do not contribute to the trace table; whereas, other lines contribute multiple times to the trace table.

Thirld, some column such as the output column, will require you to delegate its value to a separate table. Whenever, you are populating these columns, you will place an identification tag, which would be the name of the separate table, instead and you would modify the separate table. We use the separate tables for these special columns because adding the content to these columns will make the trace table harder to construct and redundant.

Last, whenever a program receives inputs, a list of inputs will be provided to perform the trace. The each input in the list corresponds to the cin statement in the same order; that is, the first input is for the first cin statement, the second input for the second cin statement and so on.

Let us generate the trace table for the code below

```
int main()
{
 int x = 6, y = 4, r;
 r = x * x - y * y;
 cout << "result:  "<< r << "\n";
 return 0;
}
```

The trace table should look something like this

step	x	y	r	output	return
1	6				
2		4			
3			20		
4				out	
5					0

out
result: 20

Notice that the values of populated columns are not repeated in future steps. This is because it is understood that the value of the column remains the same until it is reassigned. Hence, to generate the value of the `r` column, which is dependent on the values of the `x` and `y` columns, we only needed to look

at the last changes to the `x` and `y` columns.

Let us work with another example that takes inputs

```
int main()
{
 int sum, i;

 cout << "Enter an integer: ";
 cin >> i;

 sum = i + (i + 1) + (i + 2);
 cout << sum << '\n';
 return 0;
}
```

Now, the trace table given the input (`9`) is

step	i	sum	output	return
1			out	
2	9			
3		30		
4			out	
5				0

```
        out
Enter an integer:
30
```

Complete the following exercises.

Exercise 4. Generate the trace table for Exercise 1 given the inputs (`-6`, `12`).

Exercise 5. Generate the trace table for Exercise 2.

Exercise 6. Generate the trace table for Exercise 3.