

This worksheet is for your use during and after lecture. It will not be collected or graded, but I think you will find it a useful tool as you learn C++ and study for the exams. Explain all false answers for the “True or False” questions; in general, show enough work and provide enough explanation so that this sheet is a useful pre-exam review. I will be happy to review your answers with you during office-hours, via Email, or instant messaging.

1. Write C++ statements to perform the following:

(a) Declare an uninitialized pointer to an int.

```
Solution: int* ptrInt;
```

(b) Use two C++ statements (ending with semicolons) that declare an integer `foo` and a pointer to `foo` named `fooPtr`.

```
Solution:  
int foo;  
int *fooPtr( &foo );
```

(c) Use one C++ statement (ending with a semicolon) that declares two integers `foo` and `bar`, as well as pointers to them `fooPtr` and `barPtr`.

```
Solution: int foo, bar, *fooPtr( &foo ), *barPtr( &foo );
```

2. Suppose that `double v;` is already defined. Write C++ statements that do the following.

(a) Prints the memory address of `v` to `cout` (assume `cout` is already defined).

```
Solution: cout << &v << endl; //endl is optional
```

(b) Declares a pointer to `v` named `vpPtr`.

```
Solution: double* vpPtr( &v );
```

(c) Prints the value of `v` to `cout`, do not use the variable `v` in your statements.

```
Solution: cout << *vpPtr << endl; //endl is optional
```

(d) Changes the value of `v` to  $33v - 10$ , do not use the variable `v` in your statements.

```
Solution: *vpPtr = 33*(*vpPtr) - 10;
```

3. The snippet of code at the right did not produce the programmer's anticipated output of

```
x = 3 = 3
```

State why, and speculate what the program actually did print out.

**Solution:** The programmer meant to write `*xptr += 1`, instead the `*` was forgotten and the programmer changed the address that `xptr` contained.

Now it `xptr` probably points to `y` or `w`, so either

```
x = 2 = 3.141...
```

or

```
x = 2 = 2.718...
```

might have been displayed. (Which one depends on whether the hardware architecture grows the stack "down" or "up.")

```

1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main()
6 {
7     double y( acos(-1) );
8     double x(2);
9     double w( exp(1) );
10
11     double *xptr( &x );
12
13     x *= *xptr;
14     *xptr /= 2;
15     xptr += 1;
16     cout << "x= " << x << " w= "
17         << *xptr << endl;
18     return 0;
19 }

```

4. The `const` keyword could have been used by the programmer in question 3 and time would not have been wasted finding the logic bug. That is to say, the programmer could have used `const` and had the compiler find the error automatically. How would you modify the program to accomplish this by adding just one `const`?

**Solution:** Change line 11 to  
`double* const xptr( &x );`

5. Write C++ statements to perform the following:

- (a) Allocates enough memory for an array of 30,000 doubles. The code should check for an allocation error, if there is one it should perform `exit(1)` after an error message is displayed.

**Solution:**

```

5 double* const d( new(nothrow) double[30000] );
6 if( d == NULL ) {
7     cout << "Error allocating double array" << endl;
8     exit(1);
9 }

```

- (b) Allocates a 10,000 element array of type `BigImage` (which has a default constructor).

**Solution:**  
`BigImage* const b( new(nothrow) BigImage[10000] );`

- (c) Frees the memory allocated in part a.

**Solution:**  
`delete[] d;`

(d) Frees the memory allocated in part b.

**Solution:**

```
delete[] b;
```

6. Suppose your disk has two files containing lists of numbers: `double.txt` and `integer.txt`. Write a program that opens each file, counts how many numbers are in each file, allocates the appropriate amount of memory for arrays to hold these numbers, checks for memory allocation failure (displaying an error message and using `exit(1)` if `new(nothrow)` fails), and then rereads the data files populating the appropriate arrays. After all of this, print the sum of all the numbers in both arrays, free the memory appropriately, and finish the program with `exit(0)`.

**Solution:**

```
1 #include <fstream>
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 double sum_double_file( const string& filename )
7 {
8     fstream infile( filename.c_str() );
9     if( !infile ) {
10         cout << "Error opening double file_" << filename << ".\n" << endl;
11         return 0.0;
12     }
13     // count how many
14     int n(0);
15     double d;
16     while( infile >> d ) {
17         n++;
18     }
19     // allocate memory
20     double* const new_array( new(nothrow) double[n] );
21     // reopen file (we could also use seekg(0)
22     infile.close();
23     fstream infile2( filename.c_str() );
24     /**
25      * No error checking, if it worked the first time, it should work
26      * again... This logic isn't immune to runtime errors, but it is good enough
27      * for a worksheet solution.
28      */
29
30     /**
31      * Since we have a fixed amount of room (n), we need to make sure we
32      * don't read more than n values. Can you think how we could read more than
33      * n values from the file, a real, plausible explanation?
34      */
35     for( int i(0); i<n && !infile2.fail() ; i++ ) {
36         infile2 >> new_array[i];
37     }
38     // don't need this anymore
39     infile2.close();
40     // now compute the sum
41     double sum(0);
42     for( int i(0); i<n; i++ ) {
```

```
43     sum += new_array[i];
44     }
45     // delete array memory
46     delete[] new_array;
47     // return the sum
48     return sum;
49 }
50
51
52
```

```
53  /***
54  * I wrote sum_int_file() by simply copying sum_double_file and then using
55  * my editors search and replace for double<->int.
56  *
57  * C++ has a more sophisticated means of writing functions that differ by just
58  * a variable type, they are called templates.
59  *
60  * I believe it is one of the first things covered in CSCI262 (Data Structures).
61  */
62  int sum_int_file( const string& filename )
63  {
64      fstream infile( filename.c_str() );
65      if( !infile ) {
66          cout << "Error opening int_file'" << filename << "'." << endl;
67          return 0.0;
68      }
69      // count how many
70      int n(0);
71      int d;
72      while( infile >> d ) {
73          n++;
74      }
75      // allocate memory
76      int* const new_array( new(nothrow) int[n] );
77      // reopen file (we could also use seekg(0)
78      infile.close();
79      fstream infile2( filename.c_str() );
80      // read into array
81      for( int i(0); i<n && !infile2.fail() ; i++ ) {
82          infile2 >> new_array[i];
83      }
84      // don't need this anymore
85      infile2.close();
86      // now compute the sum
87      int sum(0);
88      for( int i(0); i<n; i++ ) {
89          sum += new_array[i];
90      }
91      // delete array memory
92      delete[] new_array;
93      // return the sum
94      return sum;
95  }
96
97  int main()
98  {
99      double sum( sum_double_file( "doubles.txt" ) + sum_int_file( "integers.txt" ) );
100     cout << "Sum_of_'doubles.txt'_and_'integers.txt'_is_" << sum << endl;
101
102     return 0;
103 }
```