CS 7B - Fall 2017 - Final Exam Solutions (in-class portion).

Write your responses to following questions on this paper, or attach extra, as needed. Use complete sentences where appropriate and write out code using proper style and syntax.

1. Write the number of the definition on the right next to the term it defines.

   (a) **copy** 4

   (b) **The Basic Guarantee** 12

   (c) **overload** 6

   (d) **container** 17

   (e) **RAII** 5

   (f) **pointer** 1

   (g) **reference** 13

   (h) **class** 9

   (i) **generic programming** underline 8

   (j) **object oriented programming** 14

   (k) **invariant** 16

   (l) **type** 18

   (m) **byte** 15

   (n) **template** 3

   (o) **constructor** 11

   (p) **parametric polymorphism** 7

   (q) **move** 2

   (r) **scope** 10

   (1) (1) a value used to identify a typed object in memory; (2) a variable holding such a value.

   (2) an operation that transfers a value from one object to another, leaving behind a value representing "empty."

   (3) A mechanism that allows a programmer to use types as parameters for a class or a function.

   (4) An operation making two objects have values that compare equal..

   (5) A basic technique for resource management based on scopes.

   (6) Define two functions or operators with the same name but different argument (operand) types.

   (7) The form of generic programming relying on explicit template parameters.

   (8) Writing code that works with a variety of types presented as arguments, as long as those argument types meet specific syntactic and semantic requirements.

   (9) A user-defined type that may contain data members, function members, and member types.

   (10) The region of program text (source code) in which a name can be referred to.

   (11) An operation that initializes an object. Typically establishes an invariant and often acquires resources needed for an object to be used (which are then typically released by a destructor).

   (12) Code either succeeds or throws an exception without having leaked any resources.

   (13) (1) a value describing the location of a typed value in memory; (2) a variable holding such a value.

   (14) Polymorphism you get from using class hierarchies and virtual functions.

   (15) The basic unit of addressing in most computers.

   (16) Something that must be true at given point(s) of a program; typically used to describe the state (set of values) of an object or the state of a loop before entry into the repeated statement.

   (17) An object that holds elements (other objects).

   (18) Something that defines a set of possible values and a set of operations for an object.

2. Consider the following complete program:

```
1  #include <iostream>
   using namespace std;
3
   void increment_all (int* start, int* stop) {
5    int * current = start;
     while (current != stop) {
7      ++(*current);
       ++current;
9    }
   }
11
   void print_all (const int* start, const int* stop) {
13   /* write code for this */
   }
15
   int main () {
17   int numbers[] = {10,20,30};
     increment_all (numbers,numbers+3);
19   print_all (numbers,numbers+3);
   }
```

(a) How many bytes are needed (typically) for the variable declared on line 17?
ANS: 12 bytes. Each `int` requires (typically) 4 and there are three `int`s.

(b) What part of memory is used to store the array declared on line 17?
ANS: Local variables(declared and defined in functions) are stored in the "stack" section of memory.

(c) How would you change the declaration on line 17 to allocate memory for an equivalent array on the free store?
ANS: `int* numbers = new int[3]10,20,30;`

(d) What are the types of the variables passed to `increment_all()` on line 18? How are these interpretted by `increment_all()`?
ANS: When you declare an array normally, as we do on line 17, you get a pointer for free. The name of the array acts as a pointer to the first element of the array. When we pass the array in by its name, we are passing the address of the first array element. So, the expected parameter is a pointer. When we add 3 to that pointer, we get the address (since these are `int`s) 12 bytes further down in memory.

(e) How does `increment_all()` process the input?
ANS: It makes aliases "`start`" and "`stop`" for the addresses passed in, copies `start` to an `int` pointer named "`current`". Then, while the memory address of "`current`" is not equal to that of "`stop`" the thing pointed to by `current` is incremented and the value of `current` (the pointer) is also incremented.

(f) Assuming `print_all()` prints all the values between `start` and `stop` (inclusive), what does `main()` print to the console?
ANS: `11 21 31`

(g) Write code to define `print_all()`.

```
void print_all (const int* start, const int* stop) {
  while(start!=stop)
    cout << *(start++) << " ";
}
```

3. Consider the following code:

```
#include <iostream>
using namespace std;

void change(void* data, int psize) {
  if ( psize == sizeof(char) )
  { char* pchar; pchar=(char*)data; ++(*pchar); }
  else if (psize == sizeof(int) )
  { int* pint; pint=(int*)data; ++(*pint); }
}

int main () {
  char a = 'x';
  int b = 1602;
  change(&a,sizeof(a));
  change(&b,sizeof(b));
  cout << a << ", " << b << '\n';
  return 0;
}
```

(a) Give a detailed description of `change()` and how it works. What type of input does it take? How does it process that input?
ANS: It takes a variable named `data` of type `void*`, which is a pointer, but the type that it points to is not known. It then tries to suss-out what it's pointing to by looking at it's size. If the size is one byte (the size of a `char`) then it casts it as a pointer to `char` and increments what it points to, otherwise it checks if it's a 4-byte pointer and, if so, casts it as a pointer to `int` and increments what the supposed int it points to is.

(b) What is the output of `main()`?
Well, since students had access to computers, it was pretty easy to check! You get `y, 1603`

4. Define a `File_handle` class with a constructor that takes a string argument (the file name), opens the file in the constructor, and closes it in the destructor.

```
#include <fstream>
#include<iostream>
using namespace std;

//-------------------------------------------------------------------------

class File_handle {
    FILE* f;
    string fname;
public:
    File_handle(const string& fn, const string& m);
    ~File_handle();
    FILE* file() { return f;    }
};

//-------------------------------------------------------------------------

File_handle::File_handle(const string& fn, const string& m)
    : f(0), fname(fn)
{
    cout << "Opening file " << fname << "\n";
    f = fopen(fn.c_str(),m.c_str());
```

```
    if (!f) cerr<< "Problem opening " << fname;
}

//------------------------------------------------------------------------------

File_handle::~File_handle()
{
    cout << "Closing file " << fname << "\n";
    int i = fclose(f);
    if (i!=0) cerr<<"Problem with closing ",fname;
}

//------------------------------------------------------------------------------

void f(const string& s)
{
    File_handle fh(s,"w");
    cout << "Enter a string you would like to write into the file: ";
    string in;
    getline(cin,in);
    fwrite(in.c_str(),sizeof(char),in.size(),fh.file());
}

//------------------------------------------------------------------------------

int main()
{
    f("ex4.txt");
}
```

5. Write a function `cypher()` that will take two c-strings (`char*` type), `plainText` and `codeWord` as input and write the result of shifting the upper-case alphabetic characters of `plainText` by the successive characters of `codeWord` and writing the result into memory it allocates on the free store. For instance, if the plainText is "`MEETMEATNOON`" and the codeword is "`AXE`" Then the text written into memory is `M+A=M` (since '`A`' is a zero shift) and then `E+X=B` (since '`X`' is a shift of 23) then "`E+E=I`" since '`E`' is a shift of 4, to get "`MBI..`" and so on. Use wrap-around logic so that the next shift is again by `A=0` and when you get to the end of the alphabet you have `...XYZABC...`.
   Do not use any standard library functions. Do not use subscripting; use the dereference operator *

instead.

```
#include<iostream>
using namespace std;

char* cypher(char* pt, char* ct) {
    int n{0}, m{0};
    while(*pt) {++n; ++pt; }
    char* ptcopy = new char[n];
    pt -= n;
    while(*ct) {++m; ++ct; }
    char* ctcopy = new char[m];
    ct -= m;
    int i = 0;
    while(i < n) {
        pt[i] = char(65+ (pt[i] +  ct[i%m])%26);
        ++i;
        //cout << "\npt[" << i-1 << "]=" << pt[i-1];
    }
    return pt;

}

int main() {
    //cout << "\nEnter plaintext: ";
    //char c{'a'};
    //cout << c;
    //cin.get();
    char* plaintext = new char[256];
    char* codeword = new char[256];
    cout << "\nEnter the plain text: ";
    cin >> plaintext;
    cout << "\nEnter the code word: ";
    cin >> codeword;
    //char plaintext[] = "MEETMEATNOON";
    //char cyphertext[] = "AXE";
    char* encoded = cypher(plaintext, codeword);
    cout << encoded;
}
```

6. Define a program that counts the number of words in a Document. Provide two versions: one that defines word as "a whitespace-separated sequence of characters" and one that defines word as "a sequence of consecutive alphabetic characters." For example, with the former definition, `alpha.numeric` and `as12b` are both single words, whereas with the second definition they are both two words.
ANS: My apologies for this terrible code:

```
#include<iostream>
#include<string>
#include<fstream>
using namespace std;

int countWords1(const string& fn) {
    int c{0};
    string w;
    ifstream read(fn);
    while(read>>w) ++c;
    read.close();
    return c;
}

int countWords2(const string& fn) {
    int countr{0};
    string s;
    char c{'0'};
    bool newWrd{false};
    ifstream read(fn);
    while(read>>s) {
        ++countr;
        for(int i = 0; i < s.length(); ++i)
            if(!isalpha(s[i]) && !newWrd) {
                cout << "\ns[" << i << "]="<<s[i];;
                newWrd=true;
                ++countr;
                while(!isalpha(s[i]) && newWrd) {
                    ++i; //sin!
                    if(isalpha(s[i])) newWrd=false;
                }
                //read.unget();
            }
    }
    read.close();
    return countr-1;
}
int main() {
    string fn;
    cout << "\nEnter a filename to count its words: ";
    cin >> fn;
    cout << "\nBy one count, that file has " << countWords1(fn) << " words.";
    cout << "\nBy another count, that file has " << countWords2(fn) << " words.";
}
```