Chapter 2

Basics

2.1 Exercises

Objects, Types, and Values

2.1 For each of the declarations below, do the following. State whether the declaration is also a definition. If the declaration is not a definition, write a definition for the entity being declared. If the declaration is a definition, write a declaration for the entity being declared that is not also a definition.

(a) `int i = 5;`

(b) `int abs(int x);`

(c) `float sqr(float x) {return x * x;}`

(d) `extern const double pi;`

(e) `char s[] = "Hello";`

(f) `double x;`

(g) `int (*func)(int, int);`

(h) `template <typename T> T max(T x, T y);`

(i) `template <typename T> bool is_negative(T x) {return x < 0;}`

(j) `auto meaning_of_life = 42;`

2.2 For each of the declarations below, do the following. State whether the declaration is also an initialization. If it is not an initialization, modify it so that it is one.

(a) `int x;`

(b) `void (*f)();`

(c) `const double pi = 3.14;`

2.3 State whether each line of code below corresponds to a type declaration or definition.

(a) `struct Point {double x; double y;};`

(b) `struct Thing;`

(c) `enum Fruit : int {apple, orange, banana};`

(d) `enum Color : int;`

2.4 Write a declaration for each of the entities listed below. Initialize each one. Do not use the null pointer in any initializations.

(a) a pointer to a **char**

(b) a pointer to a constant **char**

(c) a constant pointer to a **char**

(d) a constant pointer to a constant **char**

(e) a pointer to a function taking a **double** parameter and returning an **int**

(f) a pointer to a pointer to an **int**

(g) an lvalue reference to an array of 16 **int**s

(h) a pointer to an array of 10 elements of type `std::string`

(i) an lvalue reference to an array of 8 **int**s

2.5 Explain what is potentially wrong with the line of code below.

```
char c = -1;
```

2.6 For each of the conditions below, state whether the condition must be true, must be false, or could be either true or false.

(a) `sizeof(char) == 1`

(b) `sizeof(int) == 2 || sizeof(int) == 4`

(c) `sizeof(short) < sizeof(int) && sizeof(int) < sizeof(long) && sizeof(long) < sizeof(long long)`

(d) `sizeof(short) <= sizeof(int) && sizeof(int) <= sizeof(long) && sizeof(long) <= sizeof(long long)`

2.7 Using **typedef**, create a type alias for each of the types listed below.

(a) a pointer to a **char**

(b) a pointer to a **const char**

(c) a **const** pointer to a **char**

(d) a pointer to a function taking a **float** parameter and returning an **int**

(e) a pointer to an array of 16 elements of an array of 8 elements of type **long**

(f) an lvalue reference to an array of 8 **int**s

(g) an rvalue reference to a pointer to an **int**

2.8 Repeat the previous problem with a **using** statement instead of a **typedef** statement.

2.9 In the code given below, state the type of each of following objects: `a`, `b`, `c`, `d`, `e`, and `f`.

```
1  int main()
2  {
3      const int i = 42;
4      int j = 0;
5      auto a = i;
6      auto b = j;
7      decltype(i) c = 0;
8      decltype((i)) d = 0;
9      decltype(j) e;
10     decltype(&i) f;
11 }
```


2.10 For each of the literals given below, state its type.

(a) `123`

(b) `3.14`

(c) `1.0f`

(d) `"Hello, World!\n"`

(e) `0UL`

2.11 Each of the code fragments below contains an error. In each case, explain what the error is, why it is an error, and how the error can be fixed.

(a)
```
const char a = 'a';
char *c = &a;
```

(b)
```
const int i = 42;
auto& j = i;
++j;
```

2.12 What is the size of the character array `s` below and what is the length of the character string in this array as returned by the `strlen` function?

```
char s[] = "Hello";
```

2.13 In the code given below, state the type of each of following objects: `a`, `b`, `c`, `d`, `e`, `f`, `g`, and `h`.

```
1  int main()
2  {
3      const int ci = 42;
4      int i = ci;
5      auto& a = i;
6      auto& b = ci;
7      auto&& c = 0;
8      const auto&& d = 0;
9      auto e = &ci;
10     auto f = &i;
11     auto const g = &i;
12     const auto h = &i;
13 }
```

Operators and Expressions

2.14 Fully parenthesize each of the expressions given below.

(a) `a = b + c * d >> 1 & 2`

(b) `a == 0 && b != 0 || c < 0`

(c) `a & 15 != 15`

(d) `a++, b = a`

(e) `0 <= i < 8`

(f) `a = b = c = 0`

(g) `a[2][1] *= f(1, 2) + 1`

(h) `a << b << c << d`

(i) `c = a < 0`

(j) `a+++b + a + ++b`

(k) `a *= * b < 0 ? - * b : * b`

2.15 For each expression identified by a comment in the code below, state whether the expression is an lvalue or rvalue.

```
int abs(int);

void func()
{
    int x = 0;
    int y = 0;
    int z;

    ++x;
      // x
      // ++x
    y++;
      // y++
    z = x + y;
      // x + y
      // z = x + y
    z = abs(x + 1);
      // abs(x + 1)
    x = y;
      // x = y
}
```

Control-Flow Constructs and Functions

2.16 Write a program that prints to standard output the size and alignment of each of following types: **`int`**, **`long`**, **`float`**, and **`double`**. The program should also print similar information for the type corresponding to a pointer to each of these types.

2.17 Consider the execution of the program whose source listing is given below. For each line of the source code marked by a comment `/* ??? */` indicate the value of the objects `x`, `y`, and `z`, after the line of code completes execution.

```
int main()
{
    int x = 0;
    int y = 1;
    int& z = x; /* ??? */
    x = y;      /* ??? */
    y = z + 1; /* ??? */
    ++z;       /* ??? */
}
```


2.18 Write a function called `strlen` that takes a pointer to the first character of a C-style string (i.e., a null-terminated string) and returns an **`int`** indicating the number of characters in the string (not counting the terminating null character).

2.19 Write a function called `swap` that exchanges the values of two **`int`** objects, such that the two objects to be exchanged are passed to the function using:

(a) two pointers to **`int`** objects

(b) two references to **`int`** objects

Which of these two approaches is preferable? Explain your answer.

2.20 State the output that will be produced by the execution of the program listed below. Explain your answer.

```
1  #include <iostream>
2
3  void increment(int x)
4  {
5      ++x;
6  }
7
8  int main()
9  {
10     int i = 0;
11     increment(i);
12     std::cout << i << "\n";
13 }
```

2.21 Write a program that outputs the lowercase letters (i.e., a to z) and decimal digits (i.e., 0 to 9) and their corresponding integer values. (Note: The C++ language standard does not require that lower case letters be numbered consecutively. The decimal digits, however, must be numbered consecutively.)

2.22 Write a program to output the smallest and largest values of the following types: **`char`**, **`short`**, **`int`**, **`long`**, **`long long`**, **`float`**, **`double`**, **`long double`**, and **`unsigned int`**. (Hint: Use `std::numeric_limits`.)

2.23 Consider a dataset that consists of a sequence of records, where each record consists of the following two fields: 1) a name, which is a string (containing no whitespace characters) and 2) a value, which is a real number. Records and fields within records are delimited by whitespace. The same name can appear in multiple records. The number of records in the dataset may be extremely large (e.g., quadrillions of records). Develop a program that reads the above type of dataset from standard input and writes the following information to standard output: 1) the number of (distinct) names; 2) for each name, the minimum, maximum, and average of the values for all of the records with that particular name; and 3) the minimum, maximum, and average of the values for all of the records. (Hint: Use `std::map`.)

2.24 (a) Write a function `str_to_int` that converts a C-style string representing a signed integer (i.e., a string consisting of a possible plus or minus sign followed by one or more digits) to its corresponding numeric value. The function should take a single parameter that is a pointer to the C-style string of digits and return an **`int`**. If the string is not formatted correctly, the value zero should be returned.

(b) Write a program to test the `str_to_int` function. Also, test the `str_to_int` function with the program below.


```
#include <iostream>

// Place the code for the str_to_int function here.

int main()
{
   const char s[] = "123";
   std::cout << str_to_int(s) << "\n";
}
```

2.25 (a) Write a function `str_concat` that takes two C-style strings as parameters and returns the concatenation of these two strings as a C-style string. Use **`new`** to obtain the memory to hold the concatenated string.

(b) The interface provided by the `str_concat` function has an important shortcoming that is likely to lead bugs in practice. Identify this shortcoming and the type of bug that is likely to arise.

2.26 (a) Write a function `copy_ints` that copies a specified number of **`int`**s from one area of memory to another. The function should have a **`void`** return type and take the following parameters: 1) a pointer specifying the start of the source area for the copy operation; 2) an integral type specifiying how many **`int`**s to copy; and 3) a pointer specifying the start of the destination area for the copy operation.

(b) Write a program to test the `copy_ints` function. Also, test the function with the program below.

```
1  #include <iostream>
2  #include <cassert>
3
4  // Place the code for the copy_ints function here.
5
6  int main()
7  {
8     const int src[4] = {1, 2, 3, 4};
9     int dst[4] = {0, 0, 0, 0};
10    copy_ints(src, 4, dst);
11    assert(!memcmp(src, dst, sizeof(src)));
12 }
```

2.27 Consider the following types: i) a function taking arguments of type pointer to **`char`** and lvalue reference to **`int`** and returning no value; ii) a pointer to a function of the type in (i); iii) a function taking a pointer of the type in (ii) as an argument and returning no value; and iv) a function taking no arguments and returning a pointer to a function of the type in (iii). Write a declaration for each of the preceding types. Write the definition of a function `func` that takes as an argument a pointer to a function of the type in (iv) and returns its argument as the return value (without any type conversion).

2.28 Develop a program that writes the contents of one or more files in succession to standard output. (That is, the program writes the concatenation of a number of files to standard output.) The sequence of files to be processed should be specified as command-line arguments (i.e., via the arguments passed to the `main` function).

2.29 Consider the code for the function given below. Identify the one very serious bug in this code. Explain how this bug could be fixed.


```
#include <cmath>

double& get_value(double x)
{
    double result = 0.0;
    for (int i = 0; i < 100; ++i) {
        result += exp(x + i / 100.0);
    }
    return result;
}
```

Namespaces

2.30 (a) Consider the program whose source listing is given below. Modify this program so that the scope resolution operator `::` (e.g., as appears in `std::cout`) is no longer needed. Resist the temptation to commit the cardinal sin of writing "**`using namespace`** `std;`".

```
#include <iostream>

int main()
{
    std::cout << "Hello, World!\n";
}
```

(b) Explain why a using a construct like "**`using namespace`** `std;`" is truly evil.

2.31 Write your own implementation of the `strlen` function from the standard library. To avoid naming conflicts, place your `strlen` function in the namespace `mine`. Write a program that reads a line of text from standard input and then prints the length of the line of text (in characters) using your `strlen` function and also the one from the standard library. (Of course, both functions should yield the same answer.)

Preprocessor

2.32 What is either erroneous or potentially problematic with each of the following preprocessor macro definitions?

(a) **`#define`** `multiply(x, y) x * y`

(b) **`#define`** `maximum(x, y) x > y ? x : y`

(c) **`#define`** `factorial(x) ((x) * factorial((x) - 1))`

2.33 For the particular compiler that you are using, find the directory in which standard library header files (such as `iostream`) are kept.

2.34 An include guard is a construct used to avoid the potential problems caused by multiple inclusion of header files resulting from the use of the **`#include`** directive. There are two types of include guards: internal and external. An external include guard performs a test outside the header file it is guarding and includes it only once per compilation. An internal include guard performs a test inside the header file it is guarding. Write a simple program using a single header file with an internal include guard. Modify the code to use an external include guard. Discuss the advantages and disadvantages of each of these two approaches.

2.35 Preprocessor macros have many shortcomings. Explain how macros do not interact well with namespaces.

