C++ offers the

notion of a symbolic constant, that is, a named object to which you can’t give a new value after it has been initialized

constexpr double pi = 3.14159;

constexpr int max = 17; *// a literal is a constant expression*

int val = 19;

max+2 *// a constant expression (a const int plus a literal)*

val+2 *// not a constant expression: it uses a variable*

***A constexpr symbolic constant must be given a value that is known at compile time***

constexpr int max = 100;

void use(int n)

{

constexpr int c1 = max+7; // OK: c1 is 107

constexpr int c2 = n+7;// error: we don’t know the value of c2

}

**To handle cases where the value of a “variable” that is initialized with a value that is not known at compile time but never changes after initialization, C++ offers a second form of constant (a const):**

constexpr int max = 100;

void use(int n)

{

constexpr int c1 = max+7; **// OK: c1 is 107**

const int c2 = n+7; **// OK, but don’t try to change the value of c2**

c2 = 7; **// error: c2 is a const**

}

Note that a<b<c means (a<b)<c and that a<b evaluates to a Boolean value: true or false. So, a<b<c will be equivalent to

either true<c or false<c.

Use **++a** for increment

Please note that this is a logical argument about readability and correctness, not an argument about efficiency. Contrary

to popular belief, modern compilers tend to generate exactly the same code from a=a+1 as for **++a** when a is one of the built-

in types. Similarly, we prefer a\*=scale over a=a\*scale.

**Swtich statement**

**Here are some technical details about switch-statements:**

**1. The value on which we switch must be of an integer, char, or enumeration (§9.5) type. In particular, you cannot switch**

**on a string.**

**2. The values in the case labels must be constant expressions (§4.3.1). In particular, you cannot use a variable in a case**

**label.**

**3. You cannot use the same value for two case labels.**

**4. You can use several case labels for a single case.**

**5. Don’t forget to end each case with a break. Unfortunately, the compiler probably won’t warn you if you forget.**

To select based on a string you have to use an if-statement or a map (Chapter 21).

A switch-statement generates optimized code for comparing against a set of constants. For larger sets of constants, this

typically yields more efficient code than a collection of if-statements. However, this means that the case label values must be

constants and distinct.

For loop vs while loop

However, using a for-statement yields more

easily understood and more maintainable code whenever a loop can be defined as a for-statement with a simple initializer,

condition, and increment operation. Use a while-statement only when that’s not the case.

Range-for-loop

vector<int> v = {5, 7, 9, 4, 6, 8};

for (int x : v) // for each x in v

cout << x << '\n';

5.6 Exceptions

The fundamental idea is to separate detection of an error (which should be done in a called function) from the handling of an error (which should be done in the calling function) while ensuring that a detected error cannot be ignored;

Note that we used cerr rather than cout for our error output: cerr is exactly like cout except that it is meant for error

output. By default both cerr and cout write to the screen, but cerr isn’t optimized so it is more resilient to errors, and on some

operating systems it can be diverted to a different target, such as a file. Using cerr also has the simple effect of documenting

that what we write relates to errors. Consequently, we use cerr for error messages.

int main()

try {

// our program

**return 0;**

// 0 indicates success

}

catch (exception& e) {

cerr << "error: " << e.what() << '\n';

keep\_window\_open();

**return 1;**

// 1 indicates failure

}

catch (...) {

cerr << "Oops: unknown exception!\n";

keep\_window\_open();

**return 2;**

// 2 indicates failure

}

Note again that the return value from main() is passed to “the system” that invoked the program. Some systems (such as

Unix) often use that value, whereas others (such as Windows) typically ignore it. A zero indicates successful completion and a

nonzero return value from main() indicates some sort of failure.