**1.** Name four major types of errors and briefly define each one.

Compile-time errors - Errors found by compiler. There are 2 types of compile-time errors: syntax errors and type errors

Link-time errors - Errors found by linker when he tries to combine object files in executable program

Run-time errors - Errors found during the usage of the program

Logical errors - Errors found by programmer if the program produces false output

**2.** What kinds of errors can we ignore in student programs?

1. Errors caused by misbehaving hardware.

2. Errors caused by misbehaving system software.

3. Termination after encountering an error.

**3.** What guarantees should every completed project offer?

1. It has to produce correct results for appropriable input.

2. It has to give reasonable error message for illegal input.

**4.** List three approaches we can take to eliminate errors in programs and produce acceptable software.

1. Organize software in such way, that there will be minimum errors.

2. Find and eliminate all errors.

3. Ensure that remaining errors are not serious.

**5.** Why do we hate debugging?

Because in most cases no matter how hard we try we can’t eliminate all bugs, neither by debugging nor by testing.

**6.** What is a syntax error? Give five examples.

Syntax error happens when we forgot to put semicolon or close parentheses and etc.

For example:

1. cout << “Hello

2. if (1>0 cout >> {“Hello!”;

3. vector<int s(9;

4. int s(int w) }

5. char s = ‘s;

**7.** What is a type error? Give five examples.

Type error occurs when the value of one type doesn’t fit in the object of another type.

1. int s = “s”;

2. int s = 2.9;

3. char c = “String”;

4. bool = “String”;

5. char c;

int s = 20000;

c = s;

**8.** What is a linker error? Give three examples.

When the function or some object is defined more than one time or declared with a wrong type linker reports an error.

1. int f(int x, int y);

int main() {

f(2,3);

}

2. int f(int x, string y) {

return y;

}

3. int f(int x, string y) {

/\* some definition \*/

}

int f(int x, string y) {

/\* some definition \*/

}

**9.** What is a logic error? Give three examples.

Logic error is a cause of erroneous results, even if there are no more compile-time and link-time errors.

1. int square (int x, int y) return x + y /2; // it should be (x + y)/2

2. vector<int> temps(20);

int temp;

int min = 0;

int max = 0;

int main() {

while (cin >> temp) {

temps.push\_back(temp);

}

min = temps[0];

max = temps[19]; // forgot to sort the vector

}

3. i = 0;

j = 9;

while (i < 10) j++; // not ‘j’ but ‘i’

if (j < i) cout << “Success”;

**10.** List four potential sources of program errors discussed in the text.

1. Poor specification

2. Bad input

3. Logical errors

4. Incomplete programs

**11.** How do you know if a result is plausible? What techniques do you have to answer such questions?

A common technique to check the plausibility of result is “back of envelope” calculation. That’s it, we need to take a piece of paper, calculate everything by ourselves and compare it with the result of produced by our program.

**12.** Compare and contrast having the caller of a function handle a run-time error vs. the called function’s handling the run-time error.

Reasons why “callee handles the error” approach is better than “caller handles the error”

1. The code for handling the error is only in one place and that allow us to easily access it.

2. We can easily access arguments of code for handling the error.

3. We no longer have to write a test for each call of function.

**13.** Why is using exceptions a better idea than returning an “error value”?

By using exceptions we separate the error handling process from error finding process. With the use of exceptions the callee handles the error finding process and the caller handles the error itself.

**14.** How do you test if an input operation succeeded?

We can check with “if-statement” if cin operation returned “true” value or not by writing:

int d = 0;

int main() {

if (cin>>d) {

/\* some code \*/

}

else /\*what to do if an input operation wasn’t succeeded\*/

**15.** Describe the process of how exceptions are thrown and caught.

1. When appropriate piece of error-checking code in callee is succeeded the callee throws an exception

int f (int x, int y) {

MyClass { };

cin >> x >> y;

if ( x < 0 || y < 0) throw MyClass;

}

2. Code is being run in try-block

int main()

try {

f(-1,0);

}

3. If exception is thrown, the piece of code in appropriate catch-block is executed

catch (MyClass& e) {// we create reference to e

cerr << “Error, wrong arguments”;

}

**16.** Why, with a vector called v, is v[v.size()] a range error? What would be the result of calling this?

Member-function size() returns the number of elements in vector, but the count of indexes in vector begins with 0. There is no element with index v.size() in vector v because count of vector elements in size begins with 1.

The result of calling this is throwing of out\_of\_range exception by vector.

**17.** Define *pre-condition* and *post-condition*; give an example (that is not the area() function from this chapter), preferably a computation that requires a loop.

#include "std\_lib\_facilities.h"

int num = 0;

vector<int> numbers;

int main() {

while (cin >> num) {

if (num >= 1 && num <= 100) { // Pre-condition: User has to enter numbers in range between 1 and 100

numbers.push\_back(num); // Insert values in range between 1 and 100 in vector

}

else { // If the value is bigger than 100 or less than 1 output error message

cerr << "Error!\n";

continue;

}

}

sort(numbers.begin(),numbers.end()); // Sort the values in vector to find the lowest and the highest

for (int x: numbers) cout << x << "\n"; // Output values of the vector

if (numbers[0] < 1 || numbers[numbers.size() - 1] > 100) { // Post-condition: check if the lowest and the highest values of array is in the range between 1 and 100

cerr << "Double Error";

}

}

**18.** When would you *not* test a pre-condition?

1.When we know that there will be no bad input

2. When it can slow down the program

3. When it is complicated to perform the check

**19.** When would you *not* test a post-condition?

When we know that pre-condition is enough to get correct output

**20.** What are the steps in debugging a program?

1. Eliminate all compile-time errors

2. Get rid of all link-time errors

3. Ensure that program do what it supposed to do

**21.** Why does commenting help when debugging?

Because we can’t express such things like version of program, author’s name, purpose and etc. directly in code. And it let another programmer to become familiar with source code faster.

**22.** How does testing differ from debugging?

Testing is a systematic way of finding errors which is conducted by using a lot of different values as an input, where debugging is not so systematic.