CS 343 Fall 2023 – Assignment 1
Instructor: Peter Buhr
Due Date: Wednesday, September 20, 2023 at 22:00
Late Date: Friday, September 22, 2023 at 22:00

August 15, 2023

This assignment introduces exception handling and coroutines in μC++. Use it to become familiar with these new facilities, and ensure you use these concepts in your assignment solution. Unless otherwise specified, writing a C-style solution for questions is unacceptable and receives little or no marks. (You may freely use the code from these example programs.)

1. (a) Transform the C++ program in Figure 1 replacing the **throw/catch** for exceptions Ex1, Ex2, and Ex3 with:
   i. C++ program using global status-flag variables. Return codes may NOT be returned from the routines.
   ii. C++ program using a C++17 variant return-type as return codes. There are two approaches: passing the exceptions by value or pointer (using inheritance) in the variant return-type.
   iii. C program using a tagged **union** return-type as return codes, and multi-level exits in the program main to handle command-line arguments.

   Output from the transformed programs must be identical to the original program. Use printf format "%g" to print floating-point numbers in C.

   (b) i. Compare the original and transformed programs with respect to performance by doing the following:
   - Time the executions using the time command:
     ```
     $ /usr/bin/time -f "%Uu %Ss %E" ./a.out 100000000 10000 1003
     3.21u 0.02s 0:03.32
     ```
     Output from time differs depending on the shell, so use the system time command. Compare the *user* time (3.21u) only, which is the CPU time consumed solely by the execution of user code (versus system and real time).
   - If necessary, change the first command-line parameter times to adjust program execution into the range 1 to 100 seconds. (Timing results below 1 second are inaccurate.) Use the same command-line values for all experiments, if possible; otherwise, increase/decrease the arguments as necessary and scale the difference in the answer.
   - Run the experiments again after recompiling the programs with compiler optimization turned on (i.e., compiler flag -O2).
   - Include the 8 timing results to validate the experiments.

   ii. State the performance difference (larger/smaller/by how much) between the original and transformed programs, and the reason for the difference.
   iii. State the performance difference (larger/smaller/by how much) between the original and transformed programs when compiler optimization is used.

   (c) i. Run a similar experiment with compiler optimization turned on but vary the exception period (second command-line parameter eperiod) with values 1000, 100, and 50.
   - Include the 12 timing results to validate the experiments.

   ii. State the performance difference (larger/smaller/by how much) between the original and transformed programs as the exception period decreases, and the reason for the difference.

2. (a) Transform the C++ program in Figure 2, p. 3 by replacing *only* the **throw/catch** for exceptions E with longjmp/setjmp. No changes are allowed to the return type or parameters of routine Ackermann. No dynamic allocation is allowed, but creation of a global variable is allowed. No more calls to setjmp are allowed than the number of **try** ... **catch**(E) statements. Note, type jmp_buf is an array allowing instances to be passed to setjmp/longjmp without having to take the address of the argument. Output from the transformed program must be identical to the original program, **except for one aspect, which you will discover in the transformed program**.

```
#include <iostream>
#include <cstdlib>                                    // access: rand, srand
#include <cstring>                                    // access: strcmp
using namespace std;
#include <unistd.h>                                   // access: getpid

struct Ex1 { short int code; };
struct Ex2 { int code; };
struct Ex3 { long int code; };

intmax_t eperiod = 10000;                             // exception period
int randcnt = 0;
int Rand() { randcnt += 1; return rand(); }

double rtn1( double i ) {
  if ( Rand() % eperiod == 0 ) { throw Ex1{ (short int)Rand() }; } // replace
      return i + Rand();
}
double rtn2( double i ) {
  if ( Rand() % eperiod == 0 ) { throw Ex2{ Rand() }; }      // replace
      return rtn1( i ) + Rand();
}
double rtn3( double i ) {
  if ( Rand() % eperiod == 0 ) { throw Ex3{ Rand() }; }       // replace
      return rtn2( i ) + Rand();
}

static intmax_t convert( const char * str ); // copy from https://student.cs.uwaterloo.ca/~cs343/examples/convert.h

int main( int argc, char * argv[] ) {
      intmax_t times = 100000000, seed = getpid();            // default values
      struct cmd_error {};
      try {
            switch ( argc ) {
              case 4: if ( strcmp( argv[3], "d" ) != 0 ) {         // default ?
                  seed = convert( argv[3] ); if ( seed <= 0 ) throw cmd_error(); }
              case 3: if ( strcmp( argv[2], "d" ) != 0 ) {         // default ?
                  eperiod = convert( argv[2] ); if ( eperiod <= 0 ) throw cmd_error(); }
              case 2: if ( strcmp( argv[1], "d" ) != 0 ) {         // default ?
                  times = convert( argv[1] ); if ( times <= 0 ) throw cmd_error(); }
              case 1: break;                                       // use all defaults
              default: throw cmd_error();
            } // switch
      } catch( ... ) {
            cerr << "Usage: " << argv[0] << " [ times > 0 | d [ eperiod > 0 | d [ seed > 0 | d ] ] ]" << endl;
            exit( EXIT_FAILURE );
      } // try
      srand( seed );
      double rv = 0.0;
      int ev1 = 0, ev2 = 0, ev3 = 0;
      int rc = 0, ec1 = 0, ec2 = 0, ec3 = 0;

      for ( int i = 0; i < times; i += 1 ) {
            try { rv += rtn3( i ); rc += 1; }                      // replace
            // analyse exception
            catch( Ex1 ev ) { ev1 += ev.code; ec1 += 1; }          // replace
            catch( Ex2 ev ) { ev2 += ev.code; ec2 += 1; }          // replace
            catch( Ex3 ev ) { ev3 += ev.code; ec3 += 1; }          // replace
      } // for
      cout << "randcnt " << randcnt << endl;
      cout << "normal result " << rv << " exception results " << ev1 << ' ' << ev2 << ' ' << ev3 << endl;
      cout << "calls "  << rc << " exceptions " << ec1 << ' ' << ec2 << ' ' << ec3 << endl;
}
```

Figure 1: Dynamic Multi-Level Exit

```
#include <iostream>
#include <cstdlib>                                    // access: rand, srand
#include <cstring>                                    // access: strcmp
using namespace std;
#include <unistd.h>                                   // access: getpid
#ifdef NOOUTPUT
#define PRINT( stmt )
#else
#define PRINT( stmt ) stmt
#endif // NOOUTPUT
struct E {};                                          // exception type
intmax_t eperiod = 100, excepts = 0, calls = 0, dtors = 0, depth = 0; // counters
PRINT( struct T { ~T() { dtors += 1; } }; )
long int Ackermann( long int m, long int n, long int depth ) {
    calls += 1;
    if ( m == 0 ) {
        if ( rand() % eperiod <= 2 ) { PRINT( T t; ) excepts += 1; throw E(); } // replace
        return n + 1;
    } else if ( n == 0 ) {
        try { return Ackermann( m - 1, 1, depth + 1 );      // replace
        } catch( E ) {                                    // replace
            PRINT( cout << " depth " << depth << " E1 " << m << " " << n << " |" );
            if ( rand() % eperiod <= 3 ) { PRINT( T t; ) excepts += 1; throw E(); } // replace
        } // try
        PRINT( cout << " E1X " << m << " " << n << endl );
    } else {
        try { return Ackermann( m - 1, Ackermann( m, n - 1, depth + 1 ), depth + 1 ); // replace
        } catch( E ) {                                    // replace
            PRINT( cout << " depth " << depth << " E2 " << m << " " << n << " |" );
            if ( rand() % eperiod == 0 ) { PRINT( T t; ) excepts += 1; throw E(); } // replace
        } // try
        PRINT( cout << " E2X " << m << " " << n << endl );
    } // if
    return 0;                                             // recover by returning 0
}
static intmax_t convert( const char * str ); // copy from https://student.cs.uwaterloo.ca/~cs343/examples/convert.h
int main( int argc, char * argv[] ) {
    volatile intmax_t m = 4, n = 6, seed = getpid();      // default values (volatile needed for longjmp)
    struct cmd_error {};
    try {                                                 // process command-line arguments
        switch ( argc ) {
          case 5: if ( strcmp( argv[4], "d" ) != 0 ) {    // default ?
              eperiod = convert( argv[4] ); if ( eperiod <= 0 ) throw cmd_error(); }
          case 4: if ( strcmp( argv[3], "d" ) != 0 ) {    // default ?
              seed = convert( argv[3] ); if ( seed <= 0 ) throw cmd_error(); }
          case 3: if ( strcmp( argv[2], "d" ) != 0 ) {    // default ?
              n = convert( argv[2] ); if ( n < 0 ) throw cmd_error(); }
          case 2: if ( strcmp( argv[1], "d" ) != 0 ) {    // default ?
              m = convert( argv[1] ); if ( m < 0 ) throw cmd_error(); }
          case 1: break;                                  // use all defaults
          default: throw cmd_error();
        } // switch
    } catch( ... ) {
        cerr << "Usage: " << argv[0] << " [ m (>= 0) | d [ n (>= 0) | d"
            " [ seed (> 0) | d [ eperiod (> 0) | d ] ] ] ]" << endl;
        exit( EXIT_FAILURE );
    } // try
    srand( seed );                                        // seed random number
    try {                                                 // replace
        PRINT( cout << "Arguments " << m << " " << n << " " << seed << " " << eperiod << endl );
        long int val = Ackermann( m, n, 0 );
        PRINT( cout << "Ackermann " << val << endl );
    } catch( E ) {                                        // replace
        PRINT( cout << "E3" << endl );
    } // try
    cout << "calls " << calls << " exceptions " << excepts << " destructors " << dtors << endl;
}
```

Figure 2: Throw/Catch

(b) i. Explain why the output is not the same between the original and transformed program.

ii. Compare the original and transformed programs with respect to performance by doing the following:

- Recompile both the programs with preprocessor option −DNOOUTPUT to suppress output.
- Time the executions using the time command:

  ```
  $ /usr/bin/time -f "%Uu %Ss %E" ./a.out 12 12 103 28
  3.21u 0.02s 0:03.32
  ```

  Output from time differs depending on the shell, so use the system time command. Compare the *user* time (3.21u) only, which is the CPU time consumed solely by the execution of user code (versus system and real time).
- If necessary, change the command-line parameters to adjust program execution into the range 1 to 100 seconds. (Timing results below 1 second are inaccurate.) Use the same command-line values for all experiments, if possible; otherwise, increase/decrease the arguments as necessary and scale the difference in the answer.
- Run the experiments again after recompiling the programs with compiler optimization turned on (i.e., compiler flag −O2).
- Include the 4 timing results to validate the experiments.

iii. State the performance difference (larger/smaller/by how much) between the original and transformed programs, and the reason for the difference.

iv. State the performance difference (larger/smaller/by how much) between the original and transformed programs when compiler optimization is used.

3. This question requires the use of μC++, which means compiling the program with the u++ command, which is available on the undergraduate computing environment.

Write a *semi-coroutine* with the following public interface (you may only add a public destructor and private members):

```
_Coroutine FloatConstant {
    char ch;                            // character passed by cocaller
    // YOU ADD MEMBERS HERE
    void main();                        // coroutine main
  public:
    enum { EOT = '\003' };              // end of text
    _Event Match {                      // last character match
      public:
        double value;                   // floating-point value
        Match( double value ) : value( value ) {}
    };
    _Event Error {};                    // last character invalid
    void next( char c ) {
        ch = c;                         // communication input
        resume();                       // activate
    }
};
```

which verifies a string of characters corresponds to a C++ decimal floating-point constant described by the following grammar:

floating-constant : $sign_{opt}$ *fractional-constant* $exponent\text{-}part_{opt}$ $floating\text{-}suffix_{opt}$ |
  $sign_{opt}$ *digit-sequence* *exponent-part* $floating\text{-}suffix_{opt}$

fractional-constant : $digit\text{-}sequence_{opt}$ “.” *digit-sequence* | *digit-sequence* “.”

exponent-part : { “e” | “E” } $sign_{opt}$ *digit-sequence*

sign : “+” | “−”

digit-sequence : *digit* | *digit-sequence digit*

floating-suffix : “f” | “l” | “F” | “L”

digit : “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

Where X_{opt} means X may be empty. In addition, there is a maximum of 16 digits for the mantissa (non-exponent digits) and 3 digits for the characteristic (exponent digits). Note, leading zeros of the mantissa or characteristic are not part of the 16 or 3 digit restriction. The following are examples of valid and invalid C++ floating-point constants:

valid	invalid
+12.E−2	a
−12.5	+.
12.	␣12.0
−.5	12.0␣␣␣
.1e+123	1.2.0a
−12.5F	−␣12.5F
002.l	123.ff
+01234567890123456.	0123456789.01234567E−0124

To simplify parsing, assume a *valid* floating-point constant starts at the beginning of an input line, i.e., there is no leading whitespace. No checking is required for this assumption and no test data contains examples of this form. See the C library routine isdigit(c), which returns true if character c is a digit. As well, there is a cheap and cheerful C pattern for creating the value for the integer mantissa, characteristic, and exponent from the character digits; these three integer components are used to construct the floating-point value. Marks will be deducted for expensive approaches for creating the floating-point value, such as stringstream.

After creation, the coroutine is resumed with a series of characters from a string (one character at a time). Eventually, the coroutine raises one of the following exceptions at its resumer:

- Match means the characters form a valid string.
- Error means the last character forms an invalid string.

After the coroutine raises an exception, it *must* terminate and NOT be resumed again; sending more characters to a terminated coroutine generates an error.

Write a program floatconstant that checks if a string is a floating-point constant. The shell interface to the floatconstant program is as follows:

```
floatconstant [ infile ]
```

(Square brackets indicate optional command line parameters, and do not appear on the actual command line.) If no input file name is specified, input comes from standard input. Output is sent to standard output. *For any specified command-line file, check it exists and can be opened. You may assume I/O reading and writing do not result in I/O errors.* There is an example μC++ programs demonstrating how to handling command-line parameters.

The program main should:

- read a line from the input file into a string,
- create a FloatConstant coroutine,
- pass characters from the input line to the coroutine one at time, plus an EOT (end-of-text) character after all characters are passed and there is no error,
- print an appropriate message when the coroutine returns exception Match or Error.
- check for extra characters,
- terminate the coroutine, and
- repeat these steps for each line in the file.

For every non-empty input line, print the line, how much of the line is parsed, and the string yes and its floating-point value if the string is valid and no otherwise. Figure 3 shows some example output. The floating-point value is printed with a precision of 16 digits (see setprecision). If there are extra characters (including whitespace) on a line after parsing, print these characters with an appropriate warning. Print the warning Warning! Blank line for an empty input line, i.e., a line containing only '\n'.

WARNING: When writing coroutines, try to reduce or eliminate execution "state" variables and control-flow statements using them. A state variable contains information that is not part of the computation and exclusively used for control-flow purposes (like flag variables). Use of execution state variables in a coroutine usually indicates you

```
"+12.E-2" : "+12.E-2" yes, value 1200
"-12.5" : "-12.5" yes, value -12.5
"12." : "12." yes, value 12
"-.5" : "-.5" yes, value -0.5
".1e+123" : ".1e+123" yes, value 1e+122
"-12.5F" : "-12.5F" yes, value -12.5
"002.l" : "002.l" yes, value 2
"+01234567890123456." : "+01234567890123456." yes, value 1234567890123456
"" : Warning! Blank line.
"a" : "a" no
"+." : "+." no
" 12.0" : " " no -- extraneous characters "12.0"
"12.0  " : "12.0 " no -- extraneous characters "  "
"1.2.0a" : "1.2." no -- extraneous characters "0a"
"- 12.5F" : "- " no -- extraneous characters "12.5F"
"123.ff" : "123.f" yes, value 123 -- extraneous characters "f"
"0123456789.01234567E-0124" : "0123456789.01234567" no -- extraneous characters "E-0124"
```

Figure 3: Floating-Point Examples

are not using the ability of the coroutine to remember prior execution information. *Little or no marks will be given for solutions explicitly managing "state" variables.* See Section 3.1.3 in the Course Notes for details on this issue. Also, make sure a coroutine's public methods are used for passing information to the coroutine, but not for doing the coroutine's work, which must be done in the coroutine's main.

Submission Guidelines

Follow these guidelines carefully. Review the Assignment Guidelines and C++ Coding Guidelines *before* starting each assignment. **Each text or test-document file, e.g., *.{txt,testdoc} file, must be ASCII text and not exceed 500 lines in length, using the command fold -w120 *.testdoc | wc -l.** Programs should be divided into separate compilation units, i.e., *.{h,cc,C,cpp} files, where applicable. Use the submit command to electronically copy the following files to the course account.

1. q1returnglobal.{cc,C,cpp}, q1returntype.{cc,C,cpp}, q1returntypec.c – code for question 1a, p. 1. **No program documentation needs to be present in your submitted code. No test documentation is to be submitted for this question. Output for this question is checked via a marking program, so it must match exactly with the given program.**

2. q1returntype.txt – contains the information required by questions 1b, p. 1 and 1c, p. 1.

3. q2longjmp.{cc,C,cpp} – code for question 2a, p. 1. **No program documentation needs to be present in your submitted code. No test documentation is submitted for this question. Output for this question is checked via a marking program, so it must match exactly with the given program.**

4. q2longjmp.txt – contains the information required by question 2b, p. 4.

5. q3*.{h,cc,C,cpp} – code for question 3, p. 4. Split your code across *.h and *.{cc,C,cpp} files as needed. **Program documentation must be present in your submitted code. Output for this question is checked via a marking program, so it must match exactly with the given program.**

6. q3*.testdoc – test documentation for question 3, which includes the input and output of your tests. **Poor documentation of how and/or what is tested can results in a loss of all marks allocated to testing.**

7. Modify the following Makefile to compile the programs for question 1, p. 1, question 2a, p. 1, and question 3, p. 4 by inserting the object-file names matching your source-file names.

```
OUTPUT = OUTPUT
GVERSION = -12
CXX = u++                                   # uC++ compiler
#CXX = /u/cs343/cfa-cc/bin/cfa               # CFA compiler
CXXFLAGS = -g -Wall -Wextra -MMD -Wno-implicit-fallthrough -D${OUTPUT} # compiler flags
MAKEFILE_NAME = ${firstword ${MAKEFILE_LIST}}# makefile name

.SUFFIXES : .cfa                            # CFA default rules
.cfa.o :
	${CXX} ${CXXFLAGS} -c $<

OBJECTS01 = q1exception.o                   # optional build of given program
EXEC01 = exception                          # given executable name

OBJECTS1 = q1returnglobal.o                 # 1st executable object files
EXEC1 = returnglobal                        # 1st executable name

OBJECTS2 = q1returntype.o                   # 2nd executable object files
EXEC2 = returntype                          # 2nd executable name

OBJECTS3 = q1returntypec.o                  # 3rd executable object files
EXEC3 = returntypec                         # 3rd executable name

OBJECTS02 = q2throwcatch.o                  # optional build of given program
EXEC02 = throwcatch                         # given executable name

OBJECTS4 = q2longjmp.o                      # 4th executable object files
EXEC4 = longjmp                             # 4th executable name

OBJECTS5 = # object files forming 5th executable with prefix "q3"
EXEC5 = floatconstant                       # 5th executable name

OBJECTS = ${OBJECTS1} ${OBJECTS2} ${OBJECTS3} ${OBJECTS4} ${OBJECTS5}
DEPENDS = ${OBJECTS:.o=.d}
EXECS = ${EXEC1} ${EXEC2} ${EXEC3} ${EXEC4} ${EXEC5}

#############################################################

.PHONY : all clean

all : ${EXECS}                              # build all executables

${EXEC01} : ${OBJECTS01}                    # optional build of given program
	g++${GVERSION} ${CXXFLAGS} $^ -o $@

q1%.o : q1%.cc                              # change compiler 1st executable, ADJUST SUFFIX (for .C/.cpp)
	g++${GVERSION} ${CXXFLAGS} -std=c++17 -c $< -o $@

${EXEC1} : ${OBJECTS1}                      # compile and link 1st executable
	g++${GVERSION} ${CXXFLAGS} $^ -o $@

${EXEC2} : ${OBJECTS2}                      # compile and link 2nd executable
	g++${GVERSION} ${CXXFLAGS} $^ -o $@

q1%.o : q1%.c                               # change compiler 2nd executable
	gcc${GVERSION} ${CXXFLAGS} -c $< -o $@

${EXEC02} : ${OBJECTS02}                    # optional build of given program
	g++${GVERSION} ${CXXFLAGS} $^ -o $@

${EXEC3} : ${OBJECTS3}                      # compile and link 3rd executable
	g++${GVERSION} ${CXXFLAGS} $^ -o $@
```

```
q2%.o : q2%.cc                          # change compiler 4th executable, ADJUST SUFFIX (for .C/.cpp)
	g++${GVERSION} ${CXXFLAGS} −c $< −o $@

${EXEC4} : ${OBJECTS4}                  # compile and link 4th executable
	g++${GVERSION} ${CXXFLAGS} $^ −o $@

${EXEC5} : ${OBJECTS5}                  # compile and link 5th executable
	${CXX} ${CXXFLAGS} $^ −o $@
#############################################################

${OBJECTS} : ${MAKEFILE_NAME}           # OPTIONAL : changes to this file => recompile

−include ${DEPENDS}                     # include *.d files containing program dependences

clean :                                 # remove files that can be regenerated
	rm −f *.d *.o ${EXEC01} ${EXEC02} ${EXECS}
```

This makefile is used as follows:

```
$ make returnglobal
$ ./returnglobal ...
$ make returntype
$ ./returntype ...
$ make returntypec
$ ./returntypec ...
$ make longjmp
$ ./longjmp ...
$ make floatconstant
$ ./floatconstant ...
```

Put this Makefile in the directory with the programs, name the source files as specified above, and then type make returnglobal, make returntype, make returntypec, make longjmp, or make floatconstant in the directory to compile the programs. This Makefile must be submitted with the assignment to build the program, so it must be correct. Use the web tool Request Test Compilation to ensure you have submitted the appropriate files, your makefile is correct, and your code compiles in the testing environment.

Follow these guidelines. Your grade depends on it!