CS 343 Fall 2015 – Assignment 1
Instructors: Peter Buhr and Ashif Harji
Due Date: Monday, September 28, 2015 at 22:00
Late Date: Wednesday, September 30, 2015 at 22:00

September 14, 2015

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, i.e., writing a C-style solution for questions is unacceptable, and will receive little or no marks. (You may freely use the code from these example programs.)

1. (a) Transform the program in Figure 1 replacing **throw**/**catch** with longjmp/setjmp. Output from the transformed program must be identical to the original program. Note, type jmp_buf is an array allowing instances to be passed to setjmp/longjmp without having to take the address of the argument.

   (b) i. Compare the original and transformed program with respect to performance by doing the following:
   - Compile the original **throw**/**catch** and setjmp/longjmp programs without print statements.
   - Time each execution using the time command:

     ```
     % time ./a.out
     3.21u 0.02s 0:03.32 100.0%
     ```

     (Output from time differs depending on the shell, but all provide user, system and real time.) 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).
   - Use the program command-line arguments to adjust the amount of program execution to get execution times in the range 10 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 both the experiments again after recompiling the programs with compiler optimization turned on (i.e., compiler flag -O2). Include all 4 timing results to validate the experiments.

   ii. State the observed performance difference between the original and transformed program, without and with optimization.

   iii. Speculate as to the reason for the performance difference.

2. This question requires the use of μC++, which means compiling the program with the u++ command and replacing routine main with member uMain::main.

   (a) Transform the program in Figure 2, p. 3 replacing **_Resume**/**_CatchResume** with fixup routines. Output and control flow from the transformed program must be identical to the original program. No global variables may be created; additional parameters may be created for rouitnes B and C. Use C++11 lambda routines to mimic the **_CatchResume** handlers as fixup routines.

   (b) i. Compare the original and transformed program with respect to performance by doing the following:
   - Compile the original **_Resume**/**_CatchResume** and fixup-routine programs without print statements.
   - Time each execution using the time command:

     ```
     % time ./a.out
     3.21u 0.02s 0:03.32 100.0%
     ```

     (Output from time differs depending on the shell, but all provide user, system and real time.) 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).

```
#include <iostream>
#include <cstdlib>
using namespace std;
#include <unistd.h>                    // getpid

#ifdef NOOUTPUT
#define print( x )
#else
#define print( x ) x
#endif

struct E {};

long int freq = 5;

long int Ackermann( long int m, long int n ) {
    if ( m == 0 ) {
        if ( random() % freq == 0 ) throw E();
        return n + 1;
    } else if ( n == 0 ) {
        if ( random() % freq == 0 ) throw E();
        try {
            return Ackermann( m - 1, 1 );
        } catch( E ) {
            print( cout << "E1 " << m << " " << n << endl );
        } // try
    } else {
        try {
            return Ackermann( m - 1, Ackermann( m, n - 1 ) );
        } catch( E ) {
            print( cout << "E2 " << m << " " << n << endl );
        } // try
    } // if
    return 0;    // recover by returning 0
}

int main( int argc, const char *argv[] ) {
    long int Ackermann( long int m, long int n );
    long int m = 4, n = 6, seed = getpid();  // default values

    switch ( argc ) {
      case 5: freq = atoi( argv[4] );
      case 4: seed = atoi( argv[3] );
      case 3: n = atoi( argv[2] );
      case 2: m = atoi( argv[1] );
    } // switch
    srandom( seed );
    cout << m << " " << n << " " << seed << " " << freq << endl;
    try {
        cout << Ackermann( m, n ) << endl;
    } catch( E ) {
        print( cout << "E3" << endl );
    } // try
}
```

Figure 1: Throw/Catch

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

#ifdef NOOUTPUT
#define print( x )
#else
#define print( x ) x
#endif

_Event E1 {
  public:
    int &i, &j;
    E1( int &i, int &j ) : i( i ), j( j ) {}
};
_Event E2 {
  public:
    int &i;
    E2( int &i ) : i( i ) {}
};
_Event E3 {
  public:
    int i;
    E3( int i ) : i( i ) {}
};

int C( int i, int j ) {
    print( cout << i << " " << j << endl );
    _Resume E1( i, j );
    print( cout << i << " " << j << endl );
    _Resume E2( j );
    print( cout << i << " " << j << endl );
    _Resume E3( 27 );
    return i;
}
int B( int i, int j ) {
    if ( i > 0 ) B( i - 1, j );
    return C( i, j );
}
int A( int i, int j, int times ) {
    int k = 27, ret;
    try {
        for ( int i = 0; i < times; i += 1 ) {
            ret = B( i, j );
        }
        return ret;
    } _CatchResume( E1 e ) {
        e.i = i;
        e.j = j;
    } _CatchResume( E2 e ) {
        e.i = k;
    } _CatchResume( E3 e ) {
        print( cout << e.i << " " << j << endl );
    }
}
void uMain::main() {
    long int m = 4, n = 6, times = 1;     // default values

    switch ( argc ) {
      case 4: times = atoi( argv[3] );
      case 3: n = atoi( argv[2] );
      case 2: m = atoi( argv[1] );
    } // switch
    cout << m << " " << n << " " << A( m, n, times ) << endl;
}
```

Figure 2: Resume/CatchResume

- Use the program command-line arguments to adjust the amount of program execution to get execution times in the range 10 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 both the experiments again after recompiling the programs with compiler optimization turned on (i.e., compiler flag `-O2`). Include all 4 timing results to validate the experiments.

ii. State the observed performance difference between the original and transformed program, without and with optimization.

iii. Speculate as to the reason for the performance difference.

3. This question requires the use of μC++, which means compiling the program with the `u++` command and replacing routine `main` with member `uMain::main`.

   Write a *semi-coroutine* to verify a string of bytes is a valid Unicode Transformation Format 8-bit character (UTF-8). UTF-8 allows any universal character to be represented while maintaining full backwards-compatibility with ASCII encoding, which is achieved by using a variable-length encoding. The following table provides a summary of the Unicode value ranges in hexadecimal, and how they are represented in binary for UTF-8.

Unicode ranges	UTF-8 binary encoding
000000-00007F	0xxxxxxx
000080-0007FF	110xxxxx 10xxxxxx
000800-00FFFF	1110xxxx 10xxxxxx 10xxxxxx
010000-10FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

   (UTF-8 is restricted to U+10FFFF so it matches the constraints of the UTF-16 character encoding.) For example, the symbol £ is represented by Unicode value 0xA3 (binary `1010 0011`). Since £ falls within the range of 0x80 to 0x7FF, it is encoded by the UTF-8 bit string `110xxxxx 10xxxxxx`. To fit the character into the eleven bits of the UTF-8 encoding, it is padded on the left with zeroes to `00010100011`. The UTF-8 encoding becomes `11000010 10100011`, where the x's are replaced with the 11-bit binary encoding giving the UTF-8 character encoding `0xC2A3` for symbol £. Note, UTF-8 is a minimal encoding; e.g., it is incorrect to represent the value 0 by any encoding other than the first one. Use unformatted I/O (`ifstream::read`) to read the Unicode bytes and the public interface given in Figure 3 (you may only add a public constructor/destructor and private members)

   After creation, the coroutine is resumed with a series of bytes from a string (one byte at a time). The coroutine suspends after each byte or throws one of these two exception:

   - `Match` means the bytes form a valid encoding and no more bytes can be sent,
   - `Error` means the bytes form an invalid encoding and no more bytes can be sent.
     Throw `Error` for an incorrectly encoded-byte or a correct encoding that falls outside the accepted range. For example, the bytes `0xe09390` (`11100000 1001 0011 1001 000`) form a 3-byte UTF-8 character (known from the first byte). However, the character is invalid because its value `0x4d0` (`xx010011 xx010000`) $<$ `0x800`, the lower bound for a 3-byte UTF-8 character. This error can be detected at the second byte, but all three bytes must be read to allow finding the start of the next UTF-8 character for a sequence of UTF-8 characters or extra characters in this question.

   After the coroutine throws an exception, it must terminate; sending more bytes to the coroutine after this point is undefined.

   Write a program `utf8` that checks if a string follows the UTF-8 encoding. The shell interface to the `utf8` program is as follows:

   ```
   utf8 [ filename ]
   ```

   (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. Issue appropriate runtime error messages for incorrect usage or if a file cannot be opened.

   The input file contains an unknown number of UTF-8 characters separated by newline characters. Hence, the value `0xa` cannot be a UTF-8 character as it denotes newline (`'\n'`). For every non-empty input line, print the

```
_Coroutine Utf8 {
  public:
    _Event Match {
      public:
        unsigned int unicode;
        Match( unsigned int unicode ) : unicode( unicode ) {}
    };
    _Event Error {};
  private:
    union UTF8 {
        unsigned char ch;               // character passed by cocaller
        struct {                        // types for 1st UTF-8 byte
            unsigned char dt : 7;       // data
            unsigned char ck : 1;       // check
        } t1;
        struct {
            unsigned char dt : 5;       // data
            unsigned char ck : 3;       // check
        } t2;
        struct {
            // YOU FIGURE IT OUT
        } t3;
        struct {
            // YOU FIGURE IT OUT
        } t4;
        struct {                        // type for extra UTF-8 bytes
            // YOU FIGURE IT OUT
        } dt;
    } utf8;
    // YOU MAY ADD PRIVATE MEMBERS
  public:
    // YOU MAY ADD CONSTRUCTOR/DESTRUCTOR IF NEEDED
    void next( unsigned char c ) {
        utf8.ch = c;                    // insert character into union for analysis
        resume();
    }
};
```

Figure 3: UTF-8 Interface

bytes of the UTF-8 character in hexadecimal as each is checked. If a valid UTF-8 character is found, print `"valid"` followed by the Unicode value of the UTF-8 character in hexadecimal; if an invalid UTF-8 character is found, print `"invalid"`. If there are any additional bytes on a line after determining if a byte sequence is valid/invalid, the coroutine throws Match or Error, print an appropriate warning message and the additional bytes in hexadecimal. Print a warning for an empty input line (i.e., a line containing only `'\n'`). Hint: to print a character in hexadecimal use the following cast:

```
char ch = 0xff;
std::cout << std::hex << (unsigned int)(unsigned char)ch << std::endl;
```

The following is example output:

```
0x23 : valid 0x23
0x23 : valid 0x23. Extra characters 0x23
0xd790 : valid 0x5d0
0xd7 : invalid
0xc2a3 : valid 0xa3
 : Warning! Blank line.
0xb0 : invalid
0xe0e3 : invalid
0xe98080 : valid 0x9000
0xe98080 : valid 0x9000. Extra characters 0xfff8
0xe09390 : invalid
0xff : invalid. Extra characters 0x9a84
0xf09089 : invalid
0xf0908980 : valid 0x10240
0x1 : valid 0x1
```

WARNING: On little-endian architectures (e.g., like AMD/Intel x86), the compiler reverses the bit order; hence, the bit-fields in variable utf8 above must be reversed. While it is unfortunate C/C++ bit-fields lack portability across hardware architectures, they are the highest-level mechanism to manipulate bit-specific information. Your assignment will only be tested on a little endian computer.

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 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 5.3.1 in Understanding Control Flow with Concurrent Programming using μC++ for details on this issue.

Submission Guidelines

Please follow these guidelines very carefully. Review the Assignment Guidelines and C++ Coding Guidelines *before* starting each assignment. **Each text file, i.e., *.*txt file, must be ASCII text and not exceed 500 lines in length, where a line is a maximum of 120 characters.** 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. q1longjmp.{cc,C,cpp} – code for question 1a, p. 1. **No program documentation needs to be present in your submitted code. No test, user or system 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, minus one aspect of its output.**

2. q1longjmp.txt – contains the information required by question 1b, p. 1.

3. q2fixup.{cc,C,cpp} – code for question 2b, p. 1. **No program documentation needs to be present in your submitted code. No test, user or system 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.**

4. q2fixup.txt – contains the information required by question 2a, p. 1.

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. No user or system 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.**

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

7. Use the following Makefile to compile the programs for questions 1, 2 and 3:

```
CXX = u++                                       # compiler
CXXFLAGS = -g -Wall -Wno-unused-label -MMD -std=c++11 # compiler flags
MAKEFILE_NAME = ${firstword ${MAKEFILE_LIST}}  # makefile name

OBJECTS01 = q1throwcatch.o                      # optional build of given program
EXEC01 = throwcatch                             # 0th executable name

OBJECTS1 = # object files forming 1st executable with prefix "q1"
EXEC1 = longjmp                                 # 1st executable name

OBJECTS02 = q2resumption.o                      # optional build of given program
EXEC02 = resumption

OBJECTS2 = # object files forming 2nd executable with prefix "q2"
EXEC2 = fixup                                   # 2nd executable name

OBJECTS3 = # object files forming 3rd executable with prefix "q3"
EXEC3 = utf8                                    # 3rd executable name

OBJECTS = ${OBJECTS1} ${OBJECTS2} ${OBJECTS3}
DEPENDS = ${OBJECTS:.o=.d}
EXECS = ${EXEC1} ${EXEC2} ${EXEC3}
######################################################################

.PHONY : all clean

all : ${EXECS}                                  # build all executables

q1.o : q1.cc                                    # change compiler 1st executable, ADJUST SUFFIX (.cc)
    g++-4.9 ${CXXFLAGS} -c $< -o $@

q1%.o : q1%.cc                                  # change compiler 1st executable, ADJUST SUFFIX (.cc)
    g++-4.9 ${CXXFLAGS} -c $< -o $@

${EXEC01} : ${OBJECTS01}
    g++-4.9 ${CXXFLAGS} $^ -o $@

${EXEC1} : ${OBJECTS1}
    g++-4.9 ${CXXFLAGS} $^ -o $@

${OBJECTS02} : q2resumption.cc
    ${CXX} ${CXXFLAGS} -c $< -o $@

${EXEC02} : ${OBJECTS02}
    ${CXX} ${CXXFLAGS} $^ -o $@

q2.o : q2.cc                                    # change compiler 2nd executable, ADJUST SUFFIX (.cc)
    g++-4.9 ${CXXFLAGS} -c $< -o $@

q2%.o : q2%.cc                                  # change compiler 2nd executable, ADJUST SUFFIX (.cc)
    g++-4.9 ${CXXFLAGS} -c $< -o $@

${EXEC2} : ${OBJECTS2}
    g++-4.9 ${CXXFLAGS} $^ -o $@

${EXEC3} : ${OBJECTS3}
    ${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 longjmp
$ longjmp ...
$ make fixup
$ fixup
$ make utf8
$ utf8 ...
```

Put this Makefile in the directory with the programs, name the source files as specified above, and then type make longjmp or make fixup or make utf8 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!