

Chapter 16: Exception Handling

Instructor: Mark Edmonds

edmonds_mark@smc.edu

Exceptions

- Exceptions are a simple concept, but a powerful one.
- So far, if our program has runtime problem (error), we have no way to handle or correct it.
 - Imagine if every program you ran immediately crashed upon a problem (e.g. the internet was not connect, a hard drive was removed, etc). Very hard to use a computer!
 - You may remember older programs that would crash and say “Program exited with code 47” (or some other code) without providing much detail.
 - * These were unhandled exceptions, and the program crashing was the way to “fix” the problem by not allowing more problems to occur in a bad program state.
- In the programs we’ve written, you can imagine having issues in a number of ways:
 - A user could pass the wrong parameters to a function
 - Data files that need to be opened for reading or writing could not exist
 - ...just about anything you can imagine could go wrong, may go wrong

BankAccount Exceptions

- Suppose we want to use the + operator to add two `BankAccounts` together.
- This only makes sense if the accounts are owned by the same person
- But what if the user tries to add two bank accounts together that belong to different people?

```
1 BankAccount operator+ (const BankAccount& b1, const BankAccount& b2){
2     BankAccount result;
3     if (b1.my_Name == b2.my_Name) {
4         result = BankAccount(b1.my_Name, b1.my_Balance + b2.my_Balance);
5     }
6     return result;
7 }
```

- If the user passes two bank accounts that match as arguments, this works great
- If the user passes two bank accounts that don’t match as arguments, this doesn’t work well
 - We return an uninitialized bank account, but is that the behavior we really want?
 - How can the user tell whether or not the operation (adding two bank accounts) succeeded?
 - * What if both bank accounts were empty...?

- This is problematic, because it expects the user to be able to interpret a default-initialized bank account as an error
 - The function still returns a value when we really encountered an error - probably not the behavior we want
 - What if we could inform the user of an error in a different way, that didn't require a special interpretation of an otherwise "normal" execution during an error?
 - * This is what exceptions are for!

Caller-Callee Relationship revisited

- Remember our Caller-Callee relationships for functions:

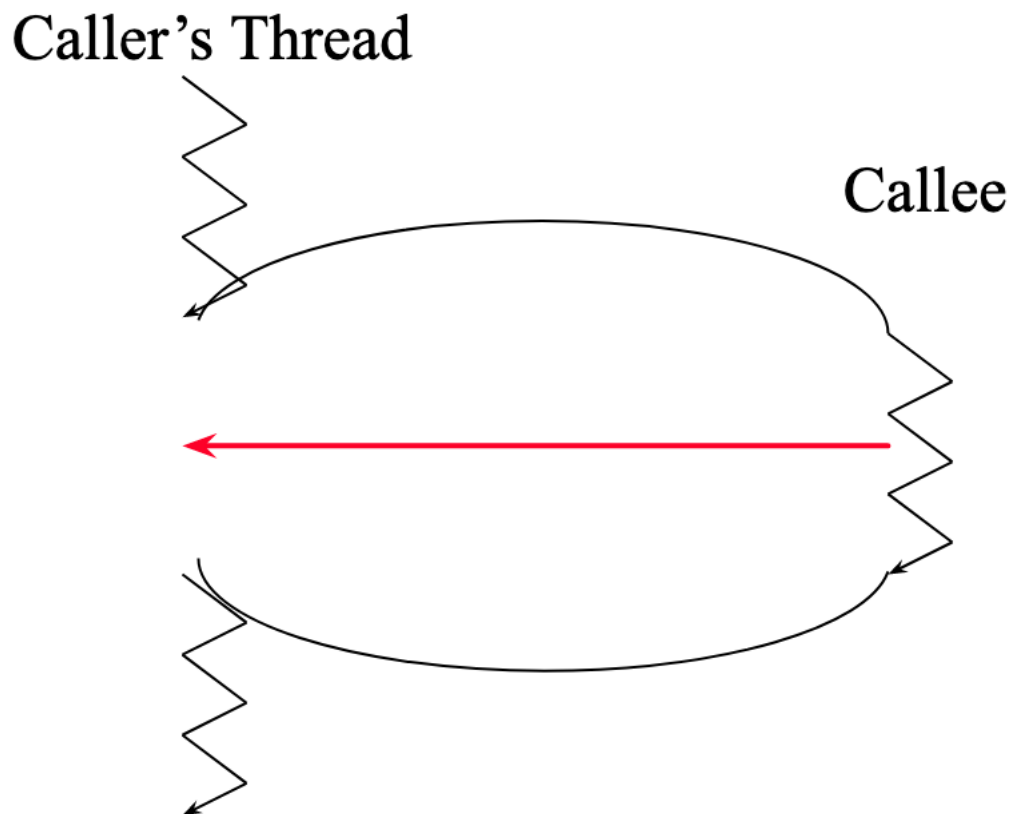


Figure 1: Caller-callee relationship

- The red line indicates that we could return to the Caller function during the Callee's execution if we encounter an error
- An exception is an "alternate" return mechanism to designate an error

- The caller then must handle the exception some way, or the program will crash
- Sending an exception to the caller is called “throwing” an exception
- Receiving and handling the exception in the caller is called “catching” an exception
- So the Callee can throw an exception, and if the Caller doesn’t catch the exception, then the program crashes
 - The analogy is like playing catch with a ball, except if the ball is dropped, the program crashes.

Throwing Exceptions

- To throw an exception, we’ll use the **throw** statement:

```
1 throw(std::logic_error("Always write a description of the problem as  
the argument to the logic_error constructor"))
```

- This is like a return statement, in the sense we “pass” a value back to the caller
- `std::logic_error` is a class
 - `#include <stdexcept>` to use it
 - We’ll eventually learn how to write our own exceptions, but for now, we can use the ones defined by the Standard Library

Catching Exceptions

- The caller needs to **try** { } to execute some code that may produce an error and **catch** () { } any errors that occur
 - You can have as many **catch** statements as necessary (meaning you can put multiple, similar to multiple **else if** statements)

```
1 try {  
2     // execute code that could throw an exception inside of a "try" block  
3     some_function_that_may_throw_a_logic_error();  
4 } catch (std::logic_error e) {  
5     // catch the exception, and do some error recovery procedure.  
6     // In this case, we just print out the exception message  
7     cout << e.what() << endl; // e.what() will return the message  
    associated with the exception  
8 }
```

Example: BankAccount with exceptions

- This example shows how to throw and use exceptions to process potentially invalid data in a loop.
- To cause an exception to be thrown, do the following:
 1. *Create* an account
 2. *Deposit* or *Withdraw* and use a different name than the name you used when you created the account

Example: ExceptionBankAccount.h

```
1  //-----
2  // INTERFACE FILE: baccount.h
3  //
4  // Defines class BankAccount
5  //
6  //-----
7  // SAFEGUARDS AND INCLUDES
8  #ifndef BANKACCOUNT_H    // Avoid redeclaring class BankAccount.
9  #define BANKACCOUNT_H    // This code is compiled only once
10 #include <string>        // for class string
11
12 namespace cs52 {
13
14  //////////////////////////////////////
15  ////////// class BankAccount defintion //////////
16  //////////////////////////////////////
17
18  class BankAccount {
19  public:    // class member functions
20
21  //--constructors
22      BankAccount();
23
24      BankAccount(std::string initName, double initBalance);
25      // post: A BankAccount with two arguments when called like this:
26      //      BankAccount anAcct("Hall", 100.00);
27
28  //--modifiers
29
30      void deposit(double depositAmount);
31      // post: depositAmount is credited to this object's balance
```

```
32
33     void withdraw(double withdrawalAmount);
34     // post: withdrawalAmount is debited from this object's balance
35
36     //--accessors
37
38     double balance() const;
39     // post: return this account's current balance
40
41     std::string name() const;
42     // post return the account name
43
44     void setName( std::string initName );
45     // post updates the member variable my_name
46
47     // ADDED CODE BEGINS HERE
48     friend std::ostream& operator << ( std::ostream& outs, const
        BankAccount& b );
49     friend std::istream& operator >> ( std::istream& ins, BankAccount&
        b );
50     friend BankAccount operator + ( const BankAccount& left, const
        BankAccount& right );
51     friend BankAccount operator - ( const BankAccount& left, const
        BankAccount& right );
52     friend bool operator ==( const BankAccount& left, const BankAccount
        & right );
53     friend bool operator < ( const BankAccount& left, const BankAccount
        & right );
54     friend bool operator > ( const BankAccount& left, const BankAccount
        & right );
55
56 private:
57     std::string my_name;    // Uniquely identify an object
58     double my_balance;    // Store the current balance (non-persistent)
59 };
60
61 }
62
63 #endif    // ifndef BANKACCOUNT_H
```

Example: ExceptionBankAccount.cpp

```
1  //-----
2  // IMPLEMENTATION FILE: baccount.cpp
3  //
4  // Implements 1. class BankAccount member functions
5  //
6  //-----
7  #include "ExceptionBankAccount.h" // allows for separate compilation
   if you want
8  #include <iostream>    // for ostream << and istream >>
9  #include <string>      // for class string
10 #include <stdexcept>   // supports Linux exception classes
11
12 using namespace std;
13
14 namespace cs52 {
15
16     //--constructors
17
18     BankAccount::BankAccount()
19     {
20         my_name = "?name?";
21         my_balance = 0.0;
22     }
23
24     BankAccount::BankAccount(string initName, double initBalance)
25     {
26         my_name = initName;
27         my_balance = initBalance;
28     }
29
30     //--modifiers
31
32     void BankAccount::deposit(double depositAmount)
33     {
34         my_balance = my_balance + depositAmount;
35     }
36
37     void BankAccount::withdraw(double withdrawalAmount)
38     {
39         my_balance = my_balance - withdrawalAmount;
40     }
41
```

```
42  //--accessors
43
44  double BankAccount::balance() const
45  {
46      return my_balance;
47  }
48
49  string BankAccount::name() const
50  {
51      return my_name;
52  }
53
54  void BankAccount::setName( string initName )
55  {
56      my_name = initName;
57  }
58
59
60  // NEW CODE STARTS HERE
61  std::ostream& operator << ( std::ostream& outs, const BankAccount& b )
62  {
63      outs << b.my_name << " " << b.my_balance << endl;
64      return( outs );
65  }
66  std::istream& operator >> ( std::istream& ins, BankAccount& b ) {
67      ins >> b.my_name >> b.my_balance;
68      return( ins );
69  }
70
71  BankAccount operator + ( const BankAccount& left, const BankAccount&
72      right ) {
73      BankAccount newB;
74      if (left.my_name == right.my_name) {
75          newB.deposit( left.my_balance );
76          newB.deposit( right.my_balance );
77      }
78      else {
79          cerr << "YIKES!  These two accounts can't be added together
80              since the names differ!" << endl;
81          throw logic_error( "Bad account names" );
82      }
83      return( newB );
```

```
82 }
83
84 BankAccount operator - ( const BankAccount& left, const BankAccount&
    right ) {
85     BankAccount newB;
86     if (left.my_name == right.my_name) {
87         newB.deposit( left.my_balance );
88         newB.withdraw( right.my_balance );
89     }
90     else {
91         cerr << "YIKES! These two accounts can't be subtracted
            together since the names differ!" << endl;
92         throw logic_error( "Bad account names" );
93     }
94     return( newB );
95 }
96
97 bool operator ==( const BankAccount& left, const BankAccount& right ) {
98     return( (left.my_balance == right.my_balance) && (left.my_name ==
        right.my_name) );
99 }
100
101 bool operator < ( const BankAccount& left, const BankAccount& right ) {
102     return( left.my_balance < right.my_balance );
103 }
104
105 bool operator > ( const BankAccount& left, const BankAccount& right ) {
106     return( left.my_balance > right.my_balance );
107 }
108
109 }
```

Example: ExceptionBanker.cpp

```
1 // This program demonstrates how to make use of existing objects.
2 // This program uses a BankAccount class with the interface described
3 // in class.
4
5 #include <iostream>                // for std::cout
6 #include <string>                  // for string class
7 #include "ExceptionBankAccount.h" // for BankAccount class
8 #include <stdexcept>              // supports Linux exceptions
```



```
9
10 using namespace std;                // supports cout
11 using namespace cs52;                // for BankAccount class
12
13 enum CHOICE { CREATE, DEPOSIT, WITHDRAW, PRINT, QUIT };
14
15 CHOICE menu();
16
17 int main( )
18 {
19     CHOICE choice;
20     BankAccount account, withdrawaccount, depositaccount;
21     string name;
22     double balance;
23
24     cout.setf( ios::fixed );
25     cout.setf( ios::showpoint );
26     cout.precision( 2 );
27
28     cout << endl << endl << "\t\tWelcome to the Bank of SMC!" << endl;
29     do {
30         choice = menu();
31         try {
32             switch (choice) {
33                 case CREATE:
34                     cout << "Please enter your name and opening bank balance: "
35                         ;
36                     cin >> name >> balance;
37                     account.setName(name);
38                     account.deposit(balance);
39                     break;
40                 case DEPOSIT:
41                     cout << "Please enter your name and amount to withdrawl: "
42                         ;
43                     cin >> name >> balance;
44                     depositaccount.setName(name);
45                     depositaccount.deposit(balance);
46                     account = account + depositaccount;
47                     break;
48                 case WITHDRAW:
49                     cout << "Please enter your name and amount to withdrawl: "
50                         ;
51                     cin >> name >> balance;
```

```
49         withdrawaccount.setName(name);
50         withdrawaccount.deposit(balance);
51         account = account - withdrawaccount;
52         break;
53     case PRINT:
54         cout << account;
55         break;
56     case QUIT:
57         break;
58     }
59 } catch (logic_error le) {
60     cout << "Caught logic_error" << endl;
61     cout << "Transaction failed to process" << endl;
62     cout << "Please try again!" << endl;
63 }
64
65 } while (choice != QUIT);
66
67 return 0;
68 }
69
70 CHOICE menu() {
71     CHOICE result;
72     char answer;
73     cout << "(C)reate (D)eposit (W)ithdrawal (P)rint (Q)uit ";
74     cin >> answer;
75     switch (answer) {
76     case 'C':
77     case 'c':
78         result = CREATE;
79         break;
80     case 'D':
81     case 'd':
82         result = DEPOSIT;
83         break;
84     case 'W':
85     case 'w':
86         result = WITHDRAW;
87         break;
88     case 'P':
89     case 'p':
90         result = PRINT;
91         break;
```

```
92     case 'Q':
93     case 'q':
94         result = QUIT;
95         break;
96     }
97     return( result );
98 }
```

- This is a good example because classes typically throw exceptions to indicate failure
- This is sense, the class is typically the callee and the user of the class is the caller

Auto example

- Exceptions are good because they allow you to greatly simplify your error checking using a consistent system that handles all error checking in one place
- To illustrate this, let's consider the following example
- We'll imagine we have a `Car` class that can fail for a number of reasons, each of which is specific to a reasonable real-world circumstance a car may face

Example: `auto_if.cpp`

```
1
2
3  WITH A C-MENTALITY AND NO EXCEPTION HANDLING....
4
5
6  ///  Supposing I Have The Class Auto
7  ///  I Am Going To Drive To Work...
8
9  Car c( "Honda", "Prelude" );
10 rv = c.openDoor();
11 if (rv == DOOR_LOCKED || rv == CAR_STOLEN || rv == WRONG_KEYS || rv ==
    WRONG_CAR ) {
12     // something bad happened...
13 }
14 else {
15     rv = c.insertKey();
16     if (rv == WRONG_KEYS || rv == KEY_UPSIDE_DOWN || rv == WRONG_CAR ) {
17         // something bad happened...
18     }
19     else {
```

```
20     rv = c.turnKey();
21     if (rv == DEAD_BATTERY || rv == NO_GAS || rv ==
22         ASTEROID_HITS_ENGINE || rv == SADDAM_IN_ENGINE) {
23         // something bad happened...
24     }
25     else {
26         rv = c.intoReverse();
27         if (rv == CLUTCH_DIED || rv == GEAR_FAILED || rv == FLAT_TIRE
28             || rv == NO_GAS || rv == PARKING_BRAKE_UP) {
29             // something bad happened...
30         }
31     }
32     else {
33         rv = c.drive();
34         if (rv == CLUTCH_DIED || rv == GEAR_FAILED || rv == NO_GAS
35             || rv == NUCLEAR_WAR) {
36             // something bad happened...
37         }
38     }
39     else {
40         rv = c.intoFirst();
41         if (rv == CLUTCH_DIED || rv == GEAR_FAILED || rv == NO_GAS ||
42             rv == SPACE_SHUTTLE_DEBRIS_HITS_WINDSHIELD) {
43             // something bad happened...
44         }
45     }
46     else {
47         // Isn't this approach ridiculous???
48         // I've literally spent so much time checking for errors,
49         // that I can't figure
50         // out what my code was actually supposed to do...
51     }
52 }
53
54
55
56
57
```

VERSUS

```

58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```
58 Car c( "Honda", "Prelude" );
59 try {
60     c.openDoor();
61     c.insertKey();
62     c.turnKey();
63     c.intoReverse();
64     c.drive();
65     c.intoFirst();
66 } catch( OutOfGasError ooge ) {
67     // something bad happened...
68 } catch( WrongKeysError wke ) {
69     // something bad happened...
70 } catch( ClutchDiedError cde ) {
71     // something bad happened...
72 } catch( GearFailedError gfe ) {
73     // something bad happened...
74 } catch( FlatTireError fte ) {
75     // something bad happened...
76 }
```

Example: auto_exception.cpp

- The above is rather hard to read, hard to maintain, and hard to expand
- Consider the following similar approach using exceptions

```
1
2
3 WITH A C-MENTALITY AND NO EXCEPTION HANDLING....
4
5
6 ///  Supposing I Have The Class Auto
7 ///  I Am Going To Drive To Work...
8
9 Car c( "Honda", "Prelude" );
10 rv = c.openDoor();
11 if (rv == DOOR_LOCKED || rv == CAR_STOLEN || rv == WRONG_KEYS || rv ==
    WRONG_CAR ) {
12     // something bad happened...
13 }
14 else {
15     rv = c.insertKey();
16     if (rv == WRONG_KEYS || rv == KEY_UPSIDE_DOWN || rv == WRONG_CAR ) {
```

```
17     // something bad happened...
18 }
19 else {
20     rv = c.turnKey();
21     if (rv == DEAD_BATTERY || rv == NO_GAS || rv ==
22         ASTEROID_HITS_ENGINE || rv == SADDAM_IN_ENGINE) {
23         // something bad happened...
24     }
25     else {
26         rv = c.intoReverse();
27         if (rv == CLUTCH_DIED || rv == GEAR_FAILED || rv == FLAT_TIRE
28             || rv == NO_GAS || rv == PARKING_BRAKE_UP) {
29             // something bad happened...
30         }
31         else {
32             rv = c.drive();
33             if (rv == CLUTCH_DIED || rv == GEAR_FAILED || rv == NO_GAS
34                 || rv == NUCLEAR_WAR) {
35                 // something bad happened...
36             }
37             else {
38                 rv = c.intoFirst();
39                 if (rv == CLUTCH_DIED || rv == GEAR_FAILED || rv == NO_GAS ||
40                     rv == SPACE_SHUTTLE_DEBRIS_HITS_WINDSHIELD) {
41                     // something bad happened...
42                 }
43             }
44         }
45     }
46 }
47 }
48 }
49
50
51
52 VERSUS
53
54
```

```
55  ///  Supposing I Have The Class Auto
56  ///  I Am Going To Drive To Work...
57
58  Car c( "Honda", "Prelude" );
59  try {
60      c.openDoor();
61      c.insertKey();
62      c.turnKey();
63      c.intoReverse();
64      c.drive();
65      c.intoFirst();
66  } catch( OutOfGasError ooge ) {
67      // something bad happened...
68  } catch( WrongKeysError wke ) {
69      // something bad happened...
70  } catch( ClutchDiedError cde ) {
71      // something bad happened...
72  } catch( GearFailedError gfe ) {
73      // something bad happened...
74  } catch( FlatTireError fte ) {
75      // something bad happened...
76  }
```