Structured Exception Handling
for
INTEL Win32

This document describes the structured exception handling architecture used by the INTEL implementation of Win32. This document is targetted at language implementors. This exception handling architecture is language and vendor independent. The Win32 exception handling implementation will support mixed language and mixed vendor development. A proposed extension to the C language is included in this document. This proposed extension has been implemented by the Microsoft Languages Group and by the MIPS Compiler Group. It is presented and used in this document as a convenient form of notation. Constraints of the syntax should not be construed as limitations of the exception handling architecture of Win32.

Definitions

Resumptive Exception Handling
A model for exception handling that allows the exception filter to either abort the procedure that caused the exception, and then invoke the exception handler, or correct the exceptional condition and continue the procedure from the point the exception occurred.

Terminative Exception Handling
A model for exception handling that requires the exception filter to abort the procedure that raised the exception, and then invoke the exception handler.

C++ Structured Exception Handling
The extensions to C++ described in the Annotated Reference Manual for C++ and adopted by the ANSI C++ committee. This structured exception handling is based on the *terminative* model.

Microsoft C++ Structured Exception Handling
The extensions to C++ described in the Annotated Reference Manual for C++ and adopted by the ANSI C++ committee, only based on the *resumptive* model of exception handling rather than the *terminative* model.

Guarded Body
The code body that is enclosed by the _try body of a _try_except or a _try_finally statement. Exceptions occuring in this code will invoke the *system exception filter.*

Lanuage Specific Exception Filter
The routine installed by the compiler in the exception handling chain. This routine is notified during the search for an exception handler, and also during the search for a termination handler. This routine is not written by the user of the language.

Exception Filter
The code body that composes the expression component of the _try_except statement. The exception filter is written by the user of the language to determine whether or not the exception that occurred can be handled by the exception handler specificied in the body of the _except statement. The exception filter is also responsible for fixing the exceptional condition and resuming execution if the resumptive model of exception handling is supported by the language implementor.

Exception Handler
The code body that is enclosed by the _except body of a _try_except statement. This code will only be executed if an exception occurs within the guarded body and the exception filter indicates that this exception handler can deal with the exception that occurred.

Termination Handler
The code body that is enclosed by the _finally body of a _try_finally statement. This code is guaranteed to get executed. It is executed after the guarded body completes or during the unwind process that occurs if an exception handler was invoked.

C Syntax Extensions

For exception handling, The syntax is

_try-_except-statement ::=

```
    _try compound-statement
    _except (expression) compound-statement
```

The compound-statement after _try is know as the **guarded statement**.
The expression after _except is know as the **exception filter**.
The compound-statement after _except is know as the **exception handler**.

For termination handling it is

_try-_finally-statement ::=

```
    _try compound-statement
    _finally  compound-statement
```

The compound-statement after _try is know as the **guarded statement**.
The compound-statement after _finally is know as the **termination handler**.

Examples:

```
_try
   {
    // guarded body
   }
_except( // exception filter
      GetExceptionCode() == EXCEPTION_ACCESS_VIOLATION      // EXAMPLE
      ? EXCEPTION_EXECUTE_HANDLER
      : EXCEPTION_CONTINUE_SEARCH)
   {
   // exception handler
   }

_try
   {
   // guarded body
   }
_finally
   {
   // termination handler
   }
```

Jumps (via goto) are not allowed into the guarded body, the exception handler, or the termination handler.

The **leave** statement is only valid within a **guarded body**. It is used to exit the **guarded body** and continue execution after the _try_except or _try_finally statement.

Intrinsic Functions

Microsoft's structured exception handling is supported by three intrinsic functions. These functions are **GetExceptionCode**, **GetExceptionInformation**, and **AbnormalTermination**.

GetExceptionCode is used to retrieve the type of exception that has occurred.

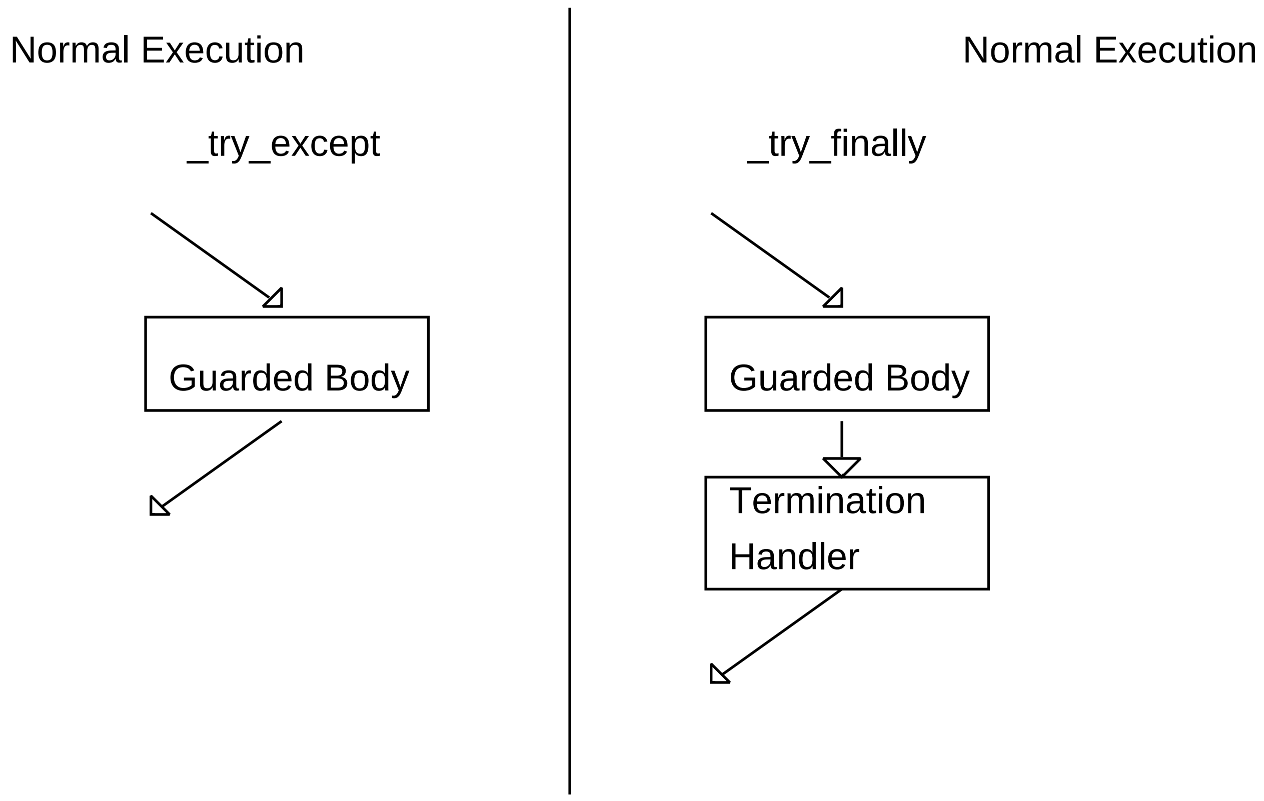
GetExceptionInformation is used to retrieve detailed information about the exception. GetExceptionInformation can only be called from within the *exception filter* of the _try-_except statement. The information provided is destroyed when RtlUnwind is called, therefore it can not be used within the *exception handler*, unless it has been copied to safe storage by the *exception filter* or *language specific exception filter*.

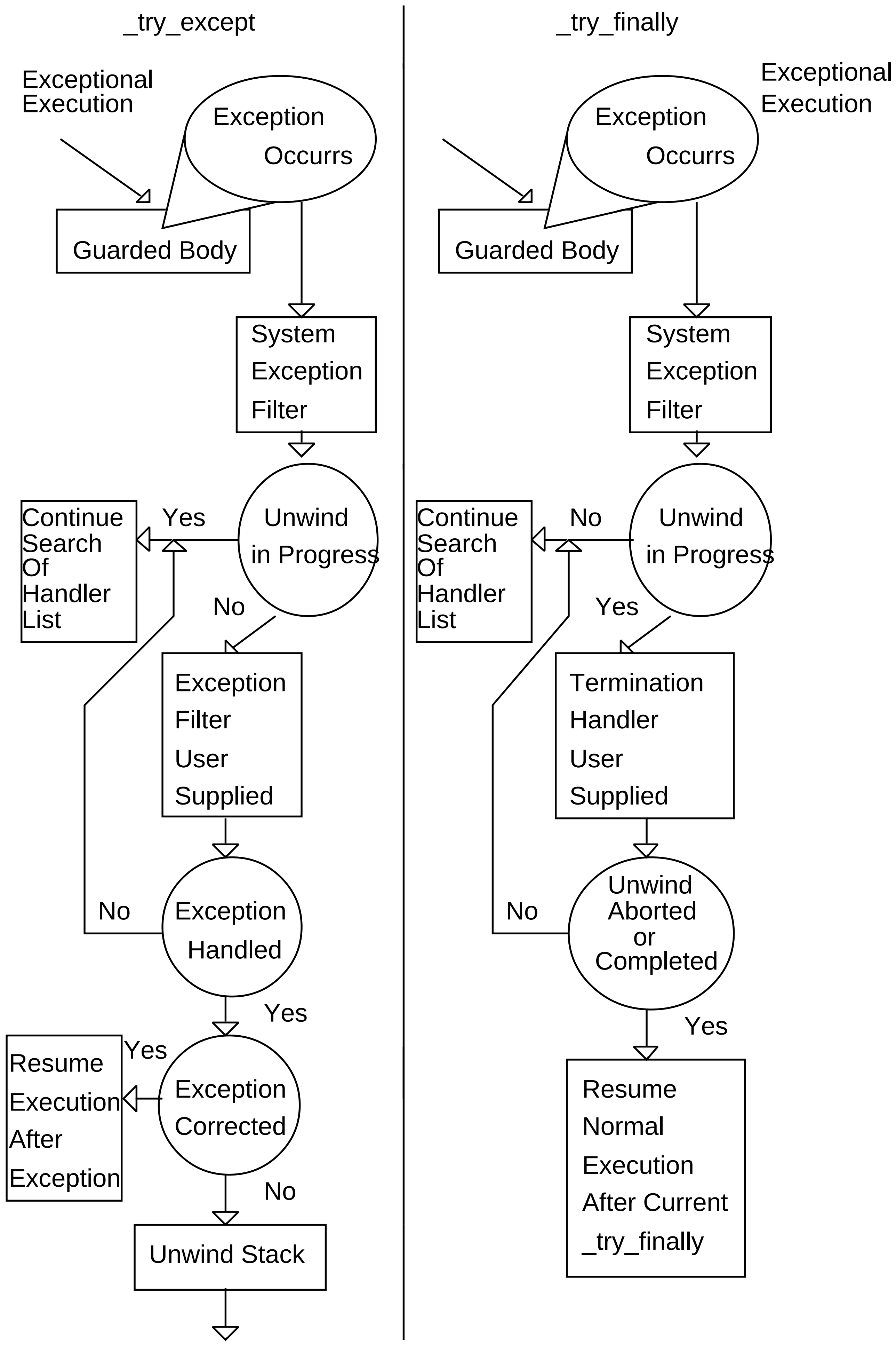
Prototype:

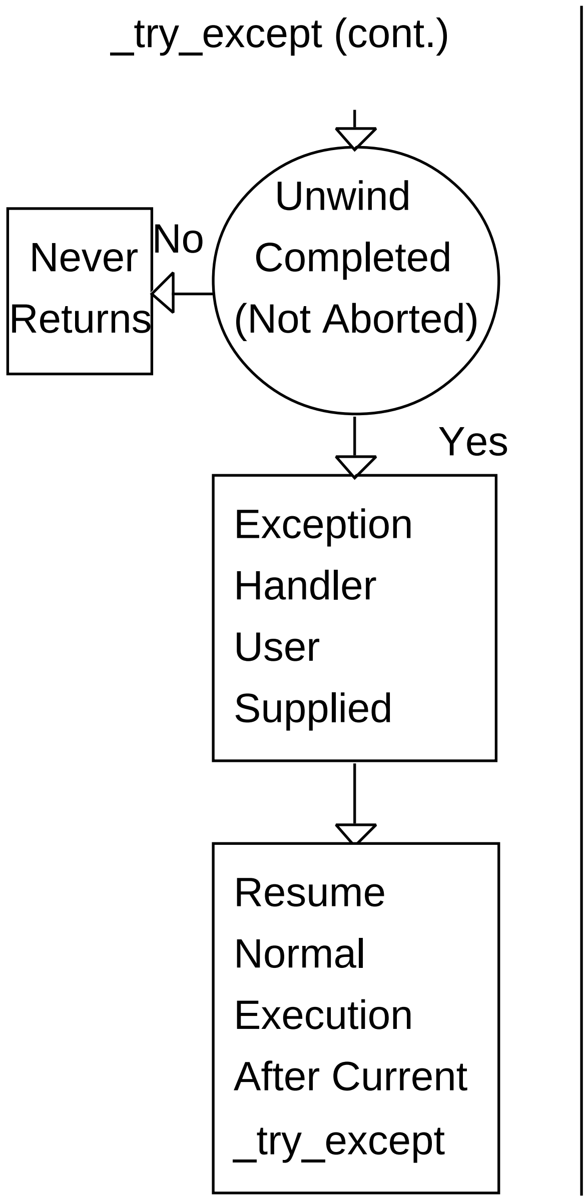
RtlUnwind(Target Frame Pointer to Unwind to,Address of execution return point,Program control flag == 0 means continue execution when done unwinding)

AbnormalTermination is used by the _try-_finally *termination handler* to determine if the handler was reached through an exception or through normal execution. This intrinsic function can only be called from within a *termination handler*.

Exception Handling Control Flow - Overview


Normal Execution
_try_except
Guarded Body
Normal Execution
_try_finally
Guarded Body
Termination Handler


_try_except
Exceptional Execution
Exception Occurrs
Guarded Body
System Exception Filter
Unwind in Progress
Yes
Continue Search Of Handler List
No
Exception Filter User Supplied
No
Exception Handled
Yes
Yes
Resume Execution After Exception
Exception Corrected
No
Unwind Stack
_try_finally
Exceptional Execution
Exception Occurrs
Guarded Body
System Exception Filter
No
Unwind in Progress
Continue Search Of Handler List
Yes
Termination Handler User Supplied
No
Unwind Aborted or Completed
Yes
Resume Normal Execution After Current _try_finally


_try_except (cont.)
Unwind Completed (Not Aborted)
No
Never Returns
Yes
Exception Handler User Supplied
Resume Normal Execution After Current _try_except

Exception handling flow control

The guarded body is executed. If an exception occurrs the operating system starts looking for an exception handler to deal with the exception. The language specific exception filter is called with pertinent information about the exception. The language specific exception filter calls the exception filter provided by the user. If the exception filter indicates that it can not deal with the exception that has occurred, then the operating system continues searching for an appropriate exception handler. If the exception filter indicates that it has corrected the exceptional event the operating system returns program control to the point exception occurred. The final option is for the exception filter to unwind the stack to the _try_except statement that installed it. After this has been accomplished the exception handler is executed and following that, normal execution resumes immediately after the _try_except statement.

Termination handling flow control

The guarded body is executed. If an exception is not generated during the execution of the guarded body, the termination handler will execute immediately after completion of the guarded body. If the guarded body generates an exception, the termination handler will be executed during the stack unwinding process that occurs after an exception filter capable of handling the exception has been found and has terminated the execution path that lead to the exception by unwinding the stack. If the exceptional condition is corrected by the exception filter, the guarded body execution path is resumed, and the termination handler will get executed as if no exception had occurred.

Execution Handling Flow Control - Code Examples

The following eight execution examples should provide a clear illustration of how the Win32 exception handling model works. The discussion in the following paragraphs will refer constantly to the attached addendums A and B, which comprise the sample program EXCEPT which should be considered part of this document.

EXCEPTION HANDLER EXAMPLES

EXCEPT 1 0

```
    Exception handling demonstration program
    Exception Handler Example (0)
        [30] ExampleTryExcept::     Guarded body enter
        [30] SEH Pointer            == 12ff78
        [31] ExampleTryExcept::     Guarded body leave
        [32] ExampleTryExcept::     Normal execution resumed
```

EXCEPT 1 1

```
    Exception handling demonstration program
    Exception Handler Example (1)
        [30] ExampleTryExcept::     Guarded body enter
        [30] SEH Pointer            == 12ff78
        [33] ExampleTryExcept::     Language specific exception filter
        [35] ExampleTryExcept::     User supplied exception filter
        [37] ExampleTryExcept::     Start stack unwind
        [37] SEH Pointer            == 12ff78
        [37] SEH Pointer            == 12ffd4
        [38] ExampleTryExcept::     Unwind complete,
                                    prepare to call exception handler
        [38] Stack Pointer          == 12ff74
        [38] Frame Pointer          == 12ff90
        [39] ExampleTryExcept::     Exception handler executing
        [32] ExampleTryExcept::     Normal execution resumed
```

EXCEPT 1 2

```
    Exception handling demonstration program
    Exception Handler Example (2)
        [30] ExampleTryExcept::     Guarded body enter
        [30] SEH Pointer            == 12ff78
        [10] SEH Pointer            == 12ff5c
        [10] UnwindTerm::           Guarded body enter
        [14] UnwindTerm::           Language specific exception filter
        [17] UnwindTerm::           Exception Handler search in progress
        [17] Stack Pointer          == 12fde4
        [33] ExampleTryExcept::     Language specific exception filter
        [35] ExampleTryExcept::     User supplied exception filter
        [37] ExampleTryExcept::         Start stack unwind
        [37] SEH Pointer                == 12ff78
        [37] SEH Pointer                == 12ffd4
        [14] UnwindTerm::               Language specific exception filter
        [15] UnwindTerm::               Unwind in progress
        [16] SEH Pointer                == 12ff78
        [16] Stack Pointer              == 12fc60
        [16] Frame Pointer              == 12ff6c
        [18] UnwindTerm::               Termination Handler
        [19] UnwindTerm::               Abnormal Termination
        [38] ExampleTryExcept::         Unwind complete,
                                        prepare to call exception handler
        [38] Stack Pointer              == 12ff74
        [38] Frame Pointer              == 12ff90
        [39] ExampleTryExcept::         Exception handler executing
        [32] ExampleTryExcept::         Normal execution resumed
```

EXCEPT 1 3

```
Exception handling demonstration program
Exception Handler Example (3)
        [30] ExampleTryExcept::         Guarded body enter
        [30] SEH Pointer                == 12ff78
        [10] SEH Pointer                == 12ff5c
        [10] UnwindTerm::               Guarded body enter
        [14] UnwindTerm::               Language specific exception filter
        [17] UnwindTerm::               Exception Handler search in progress
        [17] Stack Pointer              == 12fde4
        [33] ExampleTryExcept::         Language specific exception filter
        [35] ExampleTryExcept::         User supplied exception filter
        [37] ExampleTryExcept::         Start stack unwind
        [37] SEH Pointer                == 12ff78
        [37] SEH Pointer                == 12ffd4
        [14] UnwindTerm::               Language specific exception filter
        [15] UnwindTerm::               Unwind in progress
        [16] SEH Pointer                == 12ff78
        [16] Stack Pointer              == 12fc60
        [16] Frame Pointer              == 12ff6c
        [18] UnwindTerm::               Termination Handler
        [19] UnwindTerm::               Abnormal Termination
        [20] UnwindTerm::               Aborting unwind
        [13] UnwindTerm::               Normal execution resumed
                                        after unwind was terminated
        [31] ExampleTryExcept::         Guarded body leave
        [32] ExampleTryExcept::         Normal execution resumed
```

TERMINATION HANDLER EXAMPLES

EXCEPT 2 0

```
Exception handling demonstration program
Termination Handler Example (0)
        [ 1] SEH Pointer                == 12ff78
        [ 2] ExampleTryFinally::        Guarded body enter
        [ 3] ExampleTryFinally::        Guarded body leave
        [ 9] Stack Pointer              == 12ff88
        [ 9] ExampleTryFinally::        Termination handler executing
        [ 4] ExampleTryFinally::        Normal excution resumed
```

EXCEPT 2 1

```
Exception handling demonstration program
Termination Handler Example (1)
    [ 1] SEH Pointer                == 12ff78
    [ 2] ExampleTryFinally::        Guarded body enter
    [ 5] ExampleTryFinally::        Language specific exception filter
    [ 8] ExampleTryFinally::        Exception handler search in progress
    [ 8] Stack Pointer              == 12fe00
    ------>  OPERATING SYSTEM DIALOG
             FOR EXCEPTIONS APPEARS HERE
    [ 5] ExampleTryFinally::        Language specific exception filter
    [ 6] ExampleTryFinally::        Unwind in progress
    [ 6] Stack Pointer              == 47037c
    [ 6] Frame Pointer              == 12ff90
    [ 9] Stack Pointer              == 12fc58
    [ 9] ExampleTryFinally::        Termination handler executing
```

EXCEPT 2 2

```
Exception handling demonstration program
Termination Handler Example (2)
    [ 1] SEH Pointer                == 12ff78
    [ 2] ExampleTryFinally::        Guarded body enter
    [10] SEH Pointer                == 12ff5c
    [10] UnwindTerm::               Guarded body enter
    [14] UnwindTerm::               Language specific exception filter
    [17] UnwindTerm::               Exception Handler search in progress
    [17] Stack Pointer              == 12fde4
    [ 5] ExampleTryFinally::        Language specific exception filter
    [ 8] ExampleTryFinally::        Exception handler search in progress
    [ 8] Stack Pointer              == 12fde4
    ------>  OPERATING SYSTEM DIALOG
             FOR EXCEPTIONS APPEARS HERE
    [14] UnwindTerm::               Language specific exception filter
    [15] UnwindTerm::               Unwind in progress
    [16] SEH Pointer                == 12ff78
    [16] Stack Pointer              == 12fc3c
    [16] Frame Pointer              == 12ff6c
    [18] UnwindTerm::               Termination Handler
    [19] UnwindTerm::               Abnormal Termination
    [ 5] ExampleTryFinally::        Language specific exception filter
    [ 6] ExampleTryFinally::        Unwind in progress
    [ 6] Stack Pointer              == 47035a
    [ 6] Frame Pointer              == 12ff90
    [ 9] Stack Pointer              == 12fc3c
    [ 9] ExampleTryFinally::        Termination handler executing
```

EXCEPT 2 3

```
Exception handling demonstration program
Termination Handler Example (3)
    [ 1] SEH Pointer                == 12ff78
    [ 2] ExampleTryFinally::        Guarded body enter
    [10] SEH Pointer                == 12ff5c
    [10] UnwindTerm::               Guarded body enter
    [14] UnwindTerm::               Language specific exception filter
```

```
[17] UnwindTerm::             Exception Handler search in progress
[17] Stack Pointer            == 12fde4
[ 5] ExampleTryFinally::      Language specific exception filter
[ 8] ExampleTryFinally::      Exception handler search in progress
[ 8] Stack Pointer == 12fde4
------>  OPERATING SYSTEM DIALOG
         FOR EXCEPTIONS APPEARS HERE
[14] UnwindTerm::             Language specific exception filter
[15] UnwindTerm::             Unwind in progress
[16] SEH Pointer              == 12ff78
[16] Stack Pointer            == 12fc3c
[16] Frame Pointer            == 12ff6c
[18] UnwindTerm::             Termination Handler
[19] UnwindTerm::             Abnormal Termination
[20] UnwindTerm::             Aborting unwind
[13] UnwindTerm::             Normal execution resumed
                              after unwind was terminated
[ 3] ExampleTryFinally::      Guarded body leave
[ 9] Stack Pointer            == 12ff88
[ 9] ExampleTryFinally::      Termination handler executing
[ 4] ExampleTryFinally::      Normal excution resumed
```

Appendix A
EH.ASM

```
.386p

        _DATA segment para public 'DATA'
        _DATA ends

        FLAT group _DATA

        ; OS Support Functions

        extrn _RtlUnwind:near
        extrn _printf:near

        P EQU 8         ; offset to stack parameters, from EBP

        _UNWINDING              EQU     2
        _UNWINDING_FOR_EXIT     EQU     4
        _UNWIND_IN_PROGRESS     EQU     _UNWINDING + _UNWINDING_FOR_EXIT

        _CONTINUE_SEARCH        EQU 1
        _CONTINUE_EXECUTION     EQU 0

        ; Operating System Dependent
        _REGISTRATION_RECORD struc
                _RegistrationRecordPrev         dd      ?
                _RegistrationRecordFilter       dd      ?
        _REGISTRATION_RECORD ends

        ; Registration Record used by this example, language vendor specific
        ; Contains Operating System Dependent fields and two implementation
        ; specific fields
        _LV_REG_RECORD struc
                _LVRegRecordPrev        dd      ?       ; REQUIRED BY OS
                _LVRegRecordFilter      dd      ?       ; REQUIRED BY OS
                _LVRegRecordFramePtr    dd      ?       ; saves EBP
                _LVRegRecordStackPtr    dd      ?       ; saves ESP
        _LV_REG_RECORD ends

        ; Operating System Dependent
        _EXCEPTION_REPORT_RECORD struc
                _ExceptNumber           dd      ?
                _ExceptFlags    dd      ?
                _ExceptRecord           dd      ?
                _ExceptAddress          dd      ?
                _ExceptParameters       dd      ?
                _ExceptInfo     dd      ?
        _EXCEPTION_REPORT_RECORD ends

        ReportRecord    EQU     0
        RegRecord       EQU     4
        ContextRecord           EQU     8
        DispatchRecord          EQU     12

        ;ExceptionFilter(&ReportRecord,&RegistrationRecord,&ContextRecord,&DispatcherRecord);
        ;
        ;       ReportRecord
        ;       RegistrationRecord is a pointer to the SEH structure being processed
        ;       ContextRecord
        ;       DispatcherRecord is reserved for use by the host OS
        ;

        _ACCESS_VIOLATION equ 0C0000005h

        GENERATE_EXCEPTION macro
                xor     EAX,EAX
                mov     [EAX],EAX
                endm

        GET_EXCEPTION_CODE_INTRINSIC macro
                mov     EAX,[EBP].P+ReportRecord
                mov     EAX,[EAX]._ExceptNumber
                endm

        ABNORMAL_TERMINATION_INTRINSIC_FALSE macro
                xor     EAX,EAX
                endm

        ABNORMAL_TERMINATION_INTRINSIC_TRUE macro
                mov     EAX,1
                endm

        ABNORMAL_TERMINATION_INTRINSIC macro
                or      EAX,EAX
                endm

        PRINT_SEH macro ID,VALUE
                push    VALUE
                push    ID
                push    offset __Seh
                call    _printf
                add     ESP,12
                endm

        PRINT_STACK macro ID,VALUE
                push    ECX
                mov     ECX,VALUE
                push    VALUE
                push    ID
                push    offset __Stack
                call    _printf
                add     ESP,12
                pop     ECX
                endm

        PRINT_FRAME macro ID,VALUE
```

```
                        push        VALUE
                        push        ID
                        push        offset __Frame
                        call        _printf
                        add         ESP,12
                        endm

            PRINT_STR macro ID,STRING
                        push        ID
                        push        STRING
                        call        _printf
                        add         ESP,8
                        endm

            _TEXT segment para public 'CODE'
                        assume CS:_TEXT

            extrn __Seh    :            near
            extrn __Frame:              near
            extrn __Stack  :            near

            extrn _DT1     :            near
            extrn _DT2     :            near
            extrn _DT3     :            near
            extrn _DT4     :            near
            extrn _DT5     :            near
            extrn _DT6     :            near
            extrn _DT7     :            near
            extrn _DT8     :            near
            extrn _DT9     :            near
            extrn _DT10    :            near
            extrn _DT11    :            near
            extrn _DT12    :            near
            extrn _DT13    :            near
            extrn _DT14    :            near
            extrn _DT15    :            near
            extrn _DT16    :            near
            extrn _DT17    :            near

            extrn _DT30    :            near
            extrn _DT31    :            near
            extrn _DT32    :            near
            extrn _DT33    :            near
            extrn _DT34    :            near
            extrn _DT35    :            near
            extrn _DT36    :            near
            extrn _DT37    :            near
            extrn _DT38    :            near

            SUCCESS     equu        1
            ERROR       equ         0

            public      _ExampleTryFinally
            public      _ExampleTryExcept

;           void ExampleTryFinally(int pFlag)
;           {
;               _try
;                   {
;                   if (pFlag == 0)
;                       {
;                           // normal execution
;                       }
;                   else if (pFlag == 1)
;                       {
;                       uint *A = NULL;
;
;                       *A = 0;         // generate exception
;                       }
;                   else
;                       {
;                       UnwindTerm(pFlag);          // demonstrate unwind
;                                                   // 2 normal unwind
;                                                   // 3 aborts unwind, and resumes execution
;                       }
;                   }
;               _finally
;                   {
;                   }
;           }
;
;           void UnwindTerm(int pFlag)
;           {
;               _try
;                 {
;                 void *B = NULL;
;
;                 *B = 0;
;                 }
;               _finally
;                   {
;                   if (pFlag == 3
;                       && AbnormalTermination())
;                       {
;                       return ERROR;   // executed if (pFlag == 3
;                       }               //   and exception occurred, normal execution
;                   }                   //   resumes at the return ERROR statement
;               return SUCCESS;
;           }

            public _ExampleTryFinally

_ExampleTryFinally proc near
            push        EBP
            mov         EBP,ESP

; exception handler install beg

                        ; implementation specific
                        ; perserve register variables here
                        ; EDI and ESI are my examples

            push        EDI
            push        ESI

                        ; implementation specific portion of seh
```

```
                    ; is stored first

                    ; reserve space for stack pointer
        push        EBP
                    ; save frame pointer
        push        EBP

                    ; operating system specific portion of seh
                    ; is stored last

                    ; pointer to filter function
        push        offset FilterTerm
                    ; prev registration record
        push        FS:0

                    ; implementation specific
                    ; save stack pointer in reserved space

        mov         [ESP]+12,ESP

                    ; install filter in eh chain

        mov         FS:0,ESP

; exception handler install end

        PRINT_SEH 1,FS:0

; { guarded body of _try_finally statement

        PRINT_STR 2,<offset _DT1>

                    ; [EBP]+P == 0 means normal execution path

        cmp         dword ptr [EBP]+P,0
        je          _tm1

                        ; [EBP]+P == 1 means simple exception in guarded body

            cmp         dword ptr [EBP]+P,1
            je          _tm0

                            ; [EBP]+P > 1 means exception in called function
                            ; exception handler search and termination handler
                            ; unwind example

                push        [EBP]+P
                call        UnwindTerm
                add         ESP,4
                jmp         short _tm1

_tm0:       GENERATE_EXCEPTION

_tm1:

        PRINT_STR 3,<offset _DT2>

; } guarded body

        ; re-entry point for normal execution from exception handler
_tm2:
; exception handler remove beg
                    ; remove seh from eh chain by
                    ; restoring the previous seh pointer
                    ; to the operating system eh hook

        pop         FS:0

                    ; implementation specific
                    ; advance stack pointer to where register
                    ; variables are perserved, removing seh from stack

        add         ESP,12


                    ; implementation specific
                    ; restore register variables here
                    ; EDI and ESI are my examples

        pop         ESI
        pop         EDI

; exception handler remove end

                    ; implementation specific
                    ; set up flag to HandlerTerm that can
                    ; be used by compiler intrinsic AbnormalTermination
                    ; which is used to determine whether or not
                    ; an exceptional event has cause the execution
                    ; of the finally clause

        ABNORMAL_TERMINATION_INTRINSIC_FALSE

                    ; execute finally clause for
                    ; cases of normal program execution

        call        HandlerTerm

        PRINT_STR 4,<offset _DT3>

; return to caller
        pop         EBP
        ret
_ExampleTryFinally endp

                    ; implementation specific
                    ; routine that gets called as the exception
                    ; list is being walked. this routine is given
                    ; sufficient information to determine what
                    ; action, if any, it should take

                    ; in the case of FilterTerm, it tells the
                    ; operating system to CONTINUE SEARCH ing for
                    ; all events, and it calls HandlerTerm only
                    ; when it is called during an unwind process
                    ; it does nothing but return CONTINUE SEARCH ing
                    ; when called upon to handle an exception

        FilterTerm proc near
                    push        EBP
                    mov         EBP,ESP

                    PRINT_STR 5,<offset _DT4>

                    mov         EAX,[EBP].P+ReportRecord
                    test        dword ptr [EAX]._ExceptFlags,_UNWIND_IN_PROGRESS
                    jz          _H0


                      push        EBP
                      PRINT_STR 6,<offset _DT5>
                      pop         EBP
                                  ; execute only during
                                  ; unwind, not during
                                  ; search for exception
                                  ; handler

                                  ; restore EBP so local variables
                                  ; from ExampleTryFinally can be
                                  ; accessed by HandlerTerm, since it
                                  ; exists in the context of ExampleTryFinally

                                  ; note that the value of ESP has
                                  ; no relation to the value of ESP during
                                  ; normal execution

                      mov         EAX,[EBP].P+RegRecord
                      mov         EBP,[EAX]._LVRegRecordFramePtr

                      PRINT_STACK 6,ESP
                      PRINT_FRAME 6,EBP

                                  ; implementation specific
                                  ; set up flag to HandlerTerm that can
                                  ; be used by compiler intrinsic AbnormalTermination
                                  ; which is used to determine whether or not
                                  ; an exceptional event has cause the execution
                                  ; of the finally clause

                      ABNORMAL_TERMINATION_INTRINSIC_TRUE

                                  ; execute finally clause for
                                  ; cases of exceptional program execution

                      call       HandlerTerm

                      mov        EAX,_CONTINUE_SEARCH
                      pop        EBP
                      ret

        _H0:
                    PRINT_STR 8,<offset _DT6>
                    PRINT_STACK 8,ESP

                    mov         EAX,_CONTINUE_SEARCH
                    pop         EBP
                    ret
        FilterTerm endp

        HandlerTerm proc near
                    PRINT_STACK 9,ESP
                    PRINT_STR 9,<offset _DT7>

                    ret
        HandlerTerm endp


UnwindTerm proc near
        push        EBP
        mov         EBP,ESP

; termination handler install beg

                    ; implementation specific portion of seh
                    ; is stored first

                    ; reserve space for stack pointer

        push        EBP

                    ; save frame pointer

        push        EBP

                    ; operating system specific portion of seh
                    ; is stored last

                    ; pointer to filter function

        push        offset FilterTermUnwind

                    ; prev registration record

        push        FS:0

                    ; implementation specific
                    ; save stack pointer in reserved space

        mov         [ESP]+12,ESP

        mov         FS:0,ESP

        PRINT_SEH 10,ESP
        PRINT_STR 10,<offset _DT8>

; termination handler install end

        GENERATE_EXCEPTION

                    ; WILL NEVER EXECUTE THE CODE FROM HERE
                    ; TO _TU2 THE WAY THIS CODE IS WRITTEN
; termination handler remove beg

                    ; remove seh from eh chain by
                    ; restoring the previous seh pointer
                    ; to the operating system eh hook

        pop     FS:0

                    ; implementation specific
                    ; remove seh from stack

        mov     ESP,EBP

; termination handler remove end

                    ; implementation specific
                    ; set up flag to HandlerTerm that can
                    ; be used by compiler intrinsic AbnormalTermination
                    ; which is used to determine whether or not
                    ; an exceptional event has cause the execution

                    ; of the finally clause

        PRINT_STR 11,<offset _DT9>

        ABNORMAL_TERMINATION_INTRINSIC_FALSE

                    ; execute finally clause for
                    ; cases of normal program execution

        call    HandlerTermUnwind

  _TU1:
                    ; clean up stack

        mov     ESP,EBP

        PRINT_STR 12,<offset _DT10>
; return to caller

        mov     EAX,SUCCESS
        pop     EBP
        ret

  _TU2:

; RETURN ERROR entry point from _finally statement

; exception handler remove beg

                    ; remove seh from eh chain by
                    ; restoring the previous seh pointer
                    ; to the operating system eh hook

        pop     FS:0

                    ; implementation specific
                    ; remove seh from stack

        mov     ESP,EBP

        PRINT_STR 13,<offset _DT11>
; exception handler remove end

        mov     EAX,ERROR
        pop     EBP
        ret
UnwindTerm endp

        FilterTermUnwind proc near
                    push    EBP
                    mov     EBP,ESP

                    PRINT_STR 14,<offset _DT12>

                    mov     EAX,[EBP].P+ReportRecord
                    test    dword ptr [EAX]._ExceptFlags,_UNWIND_IN_PROGRESS
                    jz      _H2


                      push    EBP
                      PRINT_STR 15,<offset _DT13>
                      pop     EBP

                                  ; execute only during
                                  ; unwind, not during
                                  ; search for exception
                                  ; handler

                                  ; restore EBP so local variables
                                  ; from UnwindTerm can be
                                  ; accessed by HandlerTermUnwind, since it
                                  ; exists in the context of UnwindTerm

                                  ; note that the value of ESP has
                                  ; no relation to the value of ESP during
                                  ; normal execution

                      mov     EAX,[EBP].P+RegRecord
                      mov     EBP,[EAX]._LVRegRecordFramePtr

                      mov     EBX,[EAX]._LVRegRecordPrev
                      push    EBX

                      PRINT_SEH 16,EBX
                      PRINT_STACK 16,ESP
                      PRINT_FRAME 16,EBP

                      pop     EBX

                                  ; implementation specific
                                  ; set up flag to HandlerTerm that can
                                  ; be used by compiler intrinsic AbnormalTermination
                                  ; which is used to determine whether or not
                                  ; an exceptional event has cause the execution
                                  ; of the finally clause
                    ABNORMAL_TERMINATION_INTRINSIC_TRUE

                            ; execute finally clause for
                            ; cases of exceptional program execution

                    call    HandlerTermUnwind

                    jmp     short _H3

        _H2:
            PRINT_STR 17,<offset _DT14>
            PRINT_STACK 17,ESP
        _H3:
            mov     EAX,_CONTINUE_SEARCH
            pop     EBP
            ret
        FilterTermUnwind endp

        HandlerTermUnwind proc near
                    ; save abnormal termination flag

            push    EAX

                    ; save unwind frame target

            push    EBX

            PRINT_STR 18,<offset _DT15>

                    ; restore unwind frame target

            pop     EBX

                    ; restore abnormal termination flag

            pop     EAX

            ABNORMAL_TERMINATION_INTRINSIC

            jz      _H3

                    PRINT_STR 19,<offset _DT16>

                    cmp     dword ptr [EBP].P,3
                    jne     _H3

                    PRINT_STR 20,<offset _DT17>

                            ; ignore this parameter

                    push    0
                            ; resume execution immediately after the
                            ; unwind
                    push    offset _H4
                            ; target seh for RtlUnwind to unwind the
                            ; stack too
                    push    EBX
                            call    _RtlUnwind
                    _H4:        ; execution should resume here after unwind

                            ; jump back to UnwindTerm and expect
                            ; it to remove exception handler

                    push    offset _TU2
        _H3:
            ret
        HandlerTermUnwind endp

;       void ExampleTryExcept(int pFlag)
;       {
;           _try
;               {
;               printf("[%2d] \n",1);
;               if (pFlag == 0)
;                   {
;                   }
;               else if (pFlag == 1)
;                   {
;                   int *C = NULL;;
;
;                   *C = 0;
;                   }
;               else
;                   {
;                   UnwindTerm(pFlag);
;                   }
;               }
;           _except(GetExceptionCode() == ACCESS_VIOLATION)
;               {
;               }
;       }

        public _ExampleTryExcept

_ExampleTryExcept proc near
        push    EBP
        mov     EBP,ESP

; exception handler install beg

                    ; implementation specific
                    ; perserve register variables here
                    ; EDI and ESI are my examples

        push    EDI
        push    ESI

                    ; implementation specific portion of seh
                    ; is stored first

                    ; reserve space for stack pointer

        push    EBP

                    ; save frame pointer
        push    EBP

                ; operating system specific portion of seh
                ; is stored last

                ; pointer to filter function

        push    offset FilterExcept

                ; prev registration record

        push    FS:0

                ; implementation specific
                ; save stack pointer in reserved space

        mov     [ESP]+12,ESP

                ; install filter in eh chain

        mov     FS:0,ESP

        PRINT_STR 30,<offset _DT30>
        PRINT_SEH 30,ESP

; exception handler install end

; { guarded body of _try_except statement

                ; [EBP]+P == 0 means normal execution path

        cmp     dword ptr [EBP]+P,0
        je      _ex1

                    ; [EBP]+P == 1 means simple exception in guarded body

            cmp     dword ptr [EBP]+P,1
            je      _ex0

                        ; [EBP]+P > 1 means exception in called function
                        ; exception handler search and termination handler
                        ; unwind example

                push    [EBP]+P
                call    UnwindTerm
                add     ESP,4
                jmp     short _ex1

_ex0:
            GENERATE_EXCEPTION

_ex1:

        PRINT_STR 31,<offset _DT31>

; } guarded body

_ex2:

; exception handler remove beg

                  ; remove seh from eh chain by
                  ; restoring the previous seh pointer
                  ;

        pop     FS:0

                  ; implementation specific
                  ; advance stack pointer to where register
                  ; variables are perserved, removing seh from stack

        add     ESP,12

                  ; implementation specific
                  ; restore register variables here
                  ; EDI and ESI are my examples

        pop     ESI
        pop     EDI

; exception handler remove end

; NOTE: handler is not executed unless an exception occurs

        PRINT_STR 32,<offset _DT32>

; return to caller
        pop     EBP
        ret
_ExampleTryExcept endp

        ; implementation specific
        ; routine that gets called as the exception
        ; list is being walked. this routine is given
        ; sufficient information to determine what
        ; action, if any, it should take

        ; in the case of FilterExcept, it tells the
        ; operating system to CONTINUE SEARCH ing for
        ; all exceptions other than ACCESS VIOLATIONS.
        ; it also tells the operating system to CONTINUE
        ; SEARCH ing during unwinds, because it does not
        ; contain a termination handler

        FilterExcept proc near
                push    EBP
                mov     EBP,ESP
                PRINT_STR 33,<offset _DT33>
                mov     EAX,[EBP].P+ReportRecord
                test    dword ptr [EAX]._ExceptFlags,_UNWIND_IN_PROGRESS
                jz      _F0

                    PRINT_STR 34,<offset _DT34>

                                ; unwind in progress
                                ; no work for exception handler
                                ; so inform operating system to
                                ; continue search

                    mov             EAX,_CONTINUE_SEARCH
                    pop             EBP
                    ret

        _F0:
                PRINT_STR 35,<offset _DT35>
                GET_EXCEPTION_CODE_INTRINSIC
                cmp             EAX,_ACCESS_VIOLATION
                je              _F1

                    PRINT_STR 36,<offset _DT36>
                    mov             EAX,_CONTINUE_SEARCH
                    pop             EBP
                    ret

        _F1:
                            ; unwind stack frame, which causes all
                            ; termination handlers installed after
                            ; the exception handler to be invoked
                            ; note that RtlUnwind does not change
                            ; the stack pointer, but termination handlers
                            ; may alter ESP

                PRINT_STR 37,<offset _DT37>

                            ; ignore this parameter

                push            0

                            ; resume execution immediately after the
                            ; unwind

                push            offset _F2

                mov             EAX,[EBP].P+RegRecord
                push            EAX

                            ; print out start of this registration record

                PRINT_SEH 37,EAX
                pop             EAX
                            ; target seh for RtlUnwind to unwind the stack
                            ; too
                push            EAX


                mov             EAX,[EAX]
                            ; print out start of previous registration record
                PRINT_SEH 37,EAX

                call            _RtlUnwind
        _F2:                ; execution should resume here after unwind

                            ; reset EBP for Handler, so it has access to
                            ; the local variables of the function it was
                            ; declared in, likewise reset ESP

                mov             EAX,[EBP].P+RegRecord
                mov             EBP,[EAX]._LVRegRecordFramePtr
                mov             ESP,[EAX]._LVRegRecordStackPtr

                PRINT_STR 38,<offset _DT38>
                PRINT_STACK 38,ESP
                PRINT_FRAME 38,EBP

                call            HandlerExcept

                                  ; resume execution after handler
                                  ; completes execution

                push            offset _ex2

                ret
        FilterExcept endp

        ; EXCEPTION HANDLER OF _TRY_EXCEPT STATEMENT

        HandlerExcept proc near
                PRINT_STR 39,<offset _DT39>

                ret
        HandlerExcept endp

_TEXT   ends
        end
        end
```

Appendix B
EXCEPT.C

```
#include        <stdio.h>
#include        <stdlib.h>

char _Seh[]     =           "[%2d] SEH Pointer              == %x\n";
char _Stack[]   =           "[%2d] Stack Pointer            == %x\n";
char _Frame[]   =           "[%2d] Frame Pointer            == %x\n";

char DT1[]      =           "[%2d] ExampleTryFinally:: Guarded body enter\n";
char DT2[]      =           "[%2d] ExampleTryFinally:: Guarded body leave\n";
char DT3[]      =           "[%2d] ExampleTryFinally:: Normal excution resumed\n";
char DT4[]      =           "[%2d] ExampleTryFinally:: Language specific exception filter\n";
char DT5[]      =           "[%2d] ExampleTryFinally:: Unwind in progress\n";
char DT6[]      =           "[%2d] ExampleTryFinally:: Exception handler search in progress\n";
char DT7[]      =           "[%2d] ExampleTryFinally:: Termination handler executing\n";
char DT8[]      =           "[%2d] UnwindTerm::            Guarded body enter\n";
char DT9[]      =           "[%2d] UnwindTerm::            Guarded body leave\n";
char DT10[]     =           "[%2d] UnwindTerm::            Normal execution resumed\n";
char DT11[]     =           "[%2d] UnwindTerm::            Normal execution resumed after unwind was terminated\n";
char DT12[]     =           "[%2d] UnwindTerm::            Language specific exception filter\n";
char DT13[]     =           "[%2d] UnwindTerm::            Unwind in progress\n";
char DT14[]     =           "[%2d] UnwindTerm::            Exception Handler search in progress\n";
char DT15[]     =           "[%2d] UnwindTerm::            Termination Handler\n";
char DT16[]     =           "[%2d] UnwindTerm::            Abnormal Termination\n";
char DT17[]     =           "[%2d] UnwindTerm::            Aborting unwind\n";

char DT30[]     =           "[%2d] ExampleTryExcept::  Guarded body enter\n";
char DT31[]     =           "[%2d] ExampleTryExcept::  Guarded body leave\n";
char DT32[]     =           "[%2d] ExampleTryExcept::  Normal execution resumed\n";
char DT33[]     =           "[%2d] ExampleTryExcept::  Language specific exception filter\n";
char DT34[]     =           "[%2d] ExampleTryExcept::  Unwind in progress\n";
char DT35[]     =           "[%2d] ExampleTryExcept::  User supplied exception filter\n";
char DT36[]     =           "[%2d] ExampleTryExcept::  Exception is not handled by this exception handler, continue search\n";
char DT37[]     =           "[%2d] ExampleTryExcept::  Start stack unwind\n";
char DT38[]     =           "[%2d] ExampleTryExcept::  Unwind complete, prepare to call exception handler\n";
char DT39[]     =           "[%2d] ExampleTryExcept::  Exception handler executing\n";

void ExampleTryExcept(int a);
void ExampleTryFinally(int a);

void main(int argc,char **argv)
{
int i;

        printf("Exception handling demonstration program\n");
        if (argc <= 1)
            {
            printf("Usage...\n\
test statement condition statement condition ...\n\
\n\
statement can be:\n\
   1 for _try_except\n\
   2 for _try_finally\n\
\n\
condition can be:\n\
   0 for normal execution\n\
   1 for exception in the guarded body, no unwinding\n\
   2 for exception in function called by guarded body, unwinding\n\
   3 to stop unwinding from exception in function called by guarded body\n\
\n\
example:\n\
   test 1 0 2 0 1 1 2 3 1 2\n\n");
            }
        else
            {
            for (i = 1;
                i < argc;
                ++i)
                {
                int x;
                int y;

                x = atoi(argv[i]);
                if (i + 1 < argc)
                    {
                    y = atoi(argv[++i]);
                    }
                else
                    {
                    y = 0;
                    }
                switch (x)
                    {
                    case 1:
                        {
                        printf("Exception Handler Example (%d)\n",y);
                        ExampleTryExcept(y);
                        break;
                        }
                    case 2:
                        {
                        printf("Termination Handler Example (%d)\n",y);
                        ExampleTryFinally(y);
                        break;
                        }
                    default:
                        {
                        printf("\nUnrecognized Example (%d)\n\n",y);
                        }
                    }
                }
            }
}
```

Appendix C
MAKEFILE

```
except.exe : eh.obj except.obj
        link eh.obj except.obj -subsystem:console -entry:mainCRTStartup -out:except.exe
                c:\nt\mstools\lib\console.lib c:\nt\mstools\lib\ntdll.lib c:\nt\mstools\lib\base.lib
                c:\nt\mstools\lib\libc.lib

eh.obj : eh.asm
        masm386 -D__NT__ -DM_I386=1 -Mx -z -DI8086N eh.asm;
        vtomf eh.obj

except.obj : except.c
        cl386 -c -DI386 except.c
        cvtomf except.obj
```