

C++ для реверс-инженеров

МЕХАНИЗМЫ ОБРАБОТКИ ИСКЛЮЧЕНИЙ

ЯП С: поддержка исключений

- Функция `setjmp` для сохранения контекста вызова
- Функция `longjmp` для восстановления контекста вызова
- Структура контекста `jmp_buf` определяется компилятором
- Перехват исключений по факту выполняется механизмами операционной системы
- Исключения — критические ошибки исполнения программы
(`variable / 0`, `*(int*)0 = 1`, ...)

ЯП С: поддержка исключений (setjmp)

```
.text:0000000180382410
.text:0000000180382410 ; int __cdecl setjmp(jmp_buf Buf)
.text:0000000180382410 _setjmp      proc near          ; CODE XREF: _setjmp_wrapper+3↑j
.text:0000000180382410                               ; setjmp↓j
.text:0000000180382410 mov     [rcx], rdx
.text:0000000180382413 mov     [rcx+8], rbx
.text:0000000180382417 mov     [rcx+18h], rbp
.text:000000018038241B mov     [rcx+20h], rsi
.text:000000018038241F mov     [rcx+28h], rdi
.text:0000000180382423 mov     [rcx+30h], r12
.text:0000000180382427 mov     [rcx+38h], r13
.text:000000018038242B mov     [rcx+40h], r14
.text:000000018038242F mov     [rcx+48h], r15
.text:0000000180382433 lea     r8, [rsp+8]
.text:0000000180382438 mov     [rcx+10h], r8
.text:000000018038243C mov     r8, [rsp]
.text:0000000180382440 mov     [rcx+50h], r8
.text:0000000180382444 stmxcsr dword ptr [rcx+58h]
.text:0000000180382448 fnstcw  word ptr [rcx+5Ch]
.text:000000018038244B movdqa  xmmword ptr [rcx+60h], xmm6
.text:0000000180382450 movdqa  xmmword ptr [rcx+70h], xmm7
.text:0000000180382455 movdqa  xmmword ptr [rcx+80h], xmm8
.text:000000018038245E movdqa  xmmword ptr [rcx+90h], xmm9
.text:0000000180382467 movdqa  xmmword ptr [rcx+0A0h], xmm10
.text:0000000180382470 movdqa  xmmword ptr [rcx+0B0h], xmm11
.text:0000000180382479 movdqa  xmmword ptr [rcx+0C0h], xmm12
.text:0000000180382482 movdqa  xmmword ptr [rcx+0D0h], xmm13
.text:000000018038248B movdqa  xmmword ptr [rcx+0E0h], xmm14
.text:0000000180382494 movdqa  xmmword ptr [rcx+0F0h], xmm15
.text:000000018038249D xor     eax, eax
.text:000000018038249F retn
.text:000000018038249F _setjmp      endp
.text:000000018038249F
```

ЯП С: поддержка исключений (минусы)

- Не выполняется раскрутка стека с освобождением ресурсов (концепция RAII в Си не реализована) – могут остаться открытые соединения, файлы, ...
- По умолчанию функции `setjmp/longjmp` не предназначены для работы в многопоточной среде
- Целесообразно использовать только для обработки критических ситуаций – полноценной симуляции исключений С++ добиться не получится

C++: реализации механизма исключений (примеры)

GCC x64

GCC ARM32

GCC MIPS

LLVM IR

Clang x64

- ♦ MSVC x64
- ♦ MSVC x86
- ♦ Watcom-32

C++: реализации механизма исключений (GCC)

- Тесная связка компилятора и стандартной библиотеки
- Набор функций стандартной библиотеки (`__cxa_allocate_exception`, `__cxa_throw`, `__cxa_begin_catch`, `__cxa_end_catch`, `__cxa_free_exception`, `__gxx_personality_v0`, `_Unwind_Resume`, ...)
- Описывающие обработчики исключений структуры данных (LSDA, language specific data area)
- Обработчики исключений и зоны очистки (освобождения ресурсов)
- Идентификация типа исключения (RTTI) и выбор нужного обработчика

C++: реализации механизма исключений (GCC)

sym.func	16	0x00000c65	0
sym.imp._Unwind_Resume	6 	0x00000b20	0
sym.imp.__cxa_allocate_exception	6 	0x00000ab0	0
sym.imp.__cxa_begin_catch	6 	0x00000aa0	0
sym.imp.__cxa_end_catch	6 	0x00000b00	0
sym.imp.__cxa_finalize	6 	0x00000b30	0
sym.imp.__cxa_get_exception_ptr	6 	0x00000ad0	0
sym.imp.__cxa_throw	6 	0x00000b10	0
sym.imp.__stack_chk_fail	6 	0x00000ac0	0
sym.imp.operator_delete__void	6 	0x00000ae0	0
sym.imp.operator_new__unsigned_long	6 	0x00000a90	0
sym.imp.puts	6 	0x00000af0	0

С++: реализации механизма исключений (выявление обработчиков – Cutter)

X-Refs to 0x00000aa0 (5 results):

Address ▲	Type	Code	Comment
0x00000c86	Call	call __cxa_begin_catch	
0x00000d0d	Call	call __cxa_begin_catch	
0x00000e08	Call	call __cxa_begin_catch	
0x00000e74	Call	call __cxa_begin_catch	
0x00000ed9	Call	call __cxa_begin_catch	

С++: реализации механизма исключений (выявление обработчиков – IDA)

```
        push    rbp
        mov     rbp, rsp
        sub     rsp, 60h

;   try {
        lea     rdi, [rbp+var_8] ; this
        call    _ZN9BlueClassC2Ev ; BlueClass::BlueClass(void)
;   } // starts at 400AF8
        jmp     $+5
; -----


loc_400B06:                ; CODE XREF: subsubfunc(void)+11↑j
;   try {                  ; this
        lea     rdi, [rbp+var_20]
        call    _ZN10GreenClassC2Ev ; GreenClass::GreenClass(void)
;   } // starts at 400B06
        jmp     $+5
; -----
```






С++: реализации механизма исключений (выявление обработчиков – IDA)

```
loc_400B22:                                ; CODE XREF: subsubfunc(void)+2D↑j
      mov     eax, 1
      mov     edi, eax                     ; thrown_size
      call    ___cxa_allocate_exception
;   try {
      mov     ecx, offset _ZTI12IntException ; `typeinfo for'IntException
      mov     esi, ecx                     ; lptinfo
      xor     ecx, ecx
      mov     edx, ecx                     ; void (__fastcall *) (void *)
      mov     rdi, rax                     ; void *
      call    ___cxa_throw
;   } // starts at 400B2E
_z10subsubfuncv endp
```

C++: реализации механизма исключений (32-битный компилятор Watcom)


IDA - example3-watcom.obj I:\Files\seminars\CPP-win\example3-watcom.obj



 xrefs to _setjmp_

Direction	Type	Address	Text
 Down	p	func(void)+49	call _setjmp_
 Down	p	subfunc(void)+7A	call _setjmp_
 Down	p	subsubfunc(void)+49	call _setjmp_
 Down	p	subsubfunc(void)+94	call _setjmp_
 Down	p	subsubfunc(void)+FD	call _setjmp_

C++: реализации механизма исключений (32-битный компилятор Watcom)

```
30 while ( 1 )
31 {
32     v17 = 1;
33     v15 = setjmp(v14);
34     if ( v15 == 2 )
35         break;
36     if ( v15 == 1 )
37     {
38         v17 = 3;
39         v0 = printf(aMyexceptionExc);
40         _wcpp_4_catch_done_(v0);
41         v17 = 1;
42         goto LABEL_8;
43     }
```

 xrefs to _setjmp_

Direction	Typ	Address	Text
	p	example6_main(int,char **)+5C	call _setjmp_
	Do...	p example6_main(int,char **)+E7	call _setjmp_

С++: реализации механизма исключений (32-битный компилятор MSVC)

```
loc_401170:                                ; CODE XREF: _main+13B↑j
                                           ; DATA XREF: _main+136↑o
; __unwind { // _main_SEH
    mov     [ebp+var_4], 0FFFFFFFFh

loc_401177:                                ; CODE XREF: _main+16C↑j
                                           ; .text:loc_40116E↑j
;   try {
    mov     [ebp+var_4], 6
    movss   xmm0, ds:dword_40F3CC
    movss   [ebp+var_1C], xmm0
    push    offset __TI1M      ; pThrowInfo
    lea     ecx, [ebp+var_1C]
    push    ecx                ; pExceptionObject
    call    __CxxThrowException@8 ; _CxxThrowException(x,x)
```

C++: реализации механизма исключений (32-битный компилятор MSVC)

```
loc_401199:                                ; DATA XREF: .rdata:stru_4126D4↓o
;   catch(double) // owned by 401177
        sub     esp, 8
        movsd   xmm0, [ebp+var_58]
        movsd   qword ptr [esp], xmm0
        push    offset aDoubleExceptio_1 ; "Double exception handler: %f\n"
        call    _printf
        add     esp, 0Ch
        mov     eax, offset loc_4011ED
        retn

; -----

loc_4011B9:                                ; DATA XREF: .rdata:004126E4↓o
;   catch(float) // owned by 401177
        cvtss2sd xmm0, [ebp+var_34]
```

Перехват любых (не языковых) исключений в операционных системах Windows и GNU/Linux

- Регистрация обработчиков сигналов в ОС GNU/Linux (функции `signal`, `sigaction`)
- Использование конструкций `__try` и `__except` как специфичных для компилятора MSVC ключевых слов (механизм SEH, по-разному реализуется для 32-битных и 64-битных программ ОС Microsoft Windows)
- Регистрация альтернативных обработчиков исключений функциями `SetUnhandledExceptionFilter`, `AddVectoredContinueHandler` и `AddVectoredExceptionHandler`

Механизм SEH в ОС Microsoft Windows (IA-32)


- Связанный список описывающих обработчики исключений структур размещается в стеке каждого потока, в котором вызываются использующие их подпрограммы
- Список обработчиков формируется компилятором, необходимо использовать специфичные для компилятора MSVC ключевые слова `__try`, `__except` (**`catch`**) и `__finally`
- Раскрутка стека выполняется после уведомления отладчика, подключенного к процессу приложения, об исключении
- Поведение по умолчанию при отсутствии зарегистрированных программой обработчиков исключений различается в зависимости от версии ОС и её настроек (запуск JIT-отладчика, окно об ошибке, окно с возможностью получить расширенную информацию, дамп памяти, ...)

Механизм SEH в ОС Microsoft Windows (IA-32)

- Связанные с механизмом SEH структуры данных: EXCEPTION_EXECUTE_HANDLER, EXCEPTION_POINTERS, EXCEPTION_RECORD, CONTEXT
- Связанные с механизмом SEH WinAPI-функции:
 - ★ RaiseException – генерирует произвольное исключение и вызывается, в частности, определёнными по умолчанию обработчиками исключений ЯП C++ (dwExceptionCode = 0xE06D7363) и C# (dwExceptionCode = 0xE0434f4d)
 - ★ GetExceptionCode – получает код исключения в блоке обработчика исключений __except
 - ★ GetExceptionInformation – получает информацию об исключении (см. GetExceptionCode)

С++: реализации механизма исключений (32-битный компилятор MSVC, режим SEH)

```
; __unwind { // __except_handler4
    push    ebp
    mov     ebp, esp
    push    0FFFFFFFh
    push    offset stru_41DFF8
    push    offset __except_handler4
    mov     eax, large fs:0
    push    eax
    sub     esp, 8
    push    ebx
    push    esi
    push    edi
    mov     eax, __security_cookie
    xor     [ebp+ms_exc.registration.ScopeTable], eax
    xor     eax, ebp
    push    eax
    lea     eax, [ebp+ms_exc.registration]
    mov     large fs:0, eax
    mov     [ebp+ms_exc.old_esp], esp
    mov     [ebp+argc], 0
    mov     [ebp+argv], 0
;   __try { // __except at loc_4010EC
        mov     [ebp+ms_exc.registration.TryLevel], 0
        call    sub_401000
;   } // starts at 4010D1
        mov     [ebp+ms_exc.registration.TryLevel], 0FFFFFFFh
        jmp     short loc_401114
```



Механизм SEH в ОС Microsoft Windows (Intel 64)

- С тёмным прошлым успешной эксплуатации уязвимостей переполнения стековых буферов через подмену SEH-обработчиков покончено навсегда
- Основные структуры данных – `RUNTIME_FUNCTION`, `UNWIND_INFO`, `UNWIND_CODE`, `SCOPE_TABLE` (см. MSDN)
- Экземпляры обозначенных структур размещаются в отдельной секции PE-файла (`.pdata`, `.rdata`) и могут использоваться не по назначению (например, для базовой разметки исполняемого файла в процессе дизассемблирования)

С++: реализации механизма исключений (64-битный компилятор MSVC, режим SEH)

```
loc_14000104E:                                ; DATA XREF: .rdata:0000000140020AD8↓o
;  __try { // __except at loc_140001055
;          call    sub_140001000
;          jmp     short loc_140001078
;  } // starts at 14000104E
; -----

loc_140001055:                                ; DATA XREF: .rdata:0000000140020AD8↓o
;  __except(unknown_libname_43) // owned by 14000104E
;          lea     rdx, aAnExceptionWas ; "An exception was caught in __except."
;          lea     rcx, qword_140025200
;          call    sub_140001080
;          lea     rdx, sub_140002B60
```

Альтернативные обработчики исключений (Windows)

- `SetUnhandledExceptionFilter`, `AddVectoredContinueHandler` и `AddVectoredExceptionHandler` – регистрация основных и дополнительных обработчиков исключений (не имеют отношения к механизму обработки исключений в C++ или к SEH)
- `SetUnhandledExceptionFilter` переопределяет глобальный обработчик исключений
- `AddVectoredContinueHandler`/`AddVectoredExceptionHandler` добавляет очередной обработчик векторных исключений в их цепочку (связный список)
- Каждый из обработчиков может заявить об успешной обработке исключения или необходимости продолжить поиск обработчиков (актуально для векторных исключений) кодом возврата (`EXCEPTION_CONTINUE_SEARCH`, `EXCEPTION_CONTINUE_EXECUTION`)

Обработка исключений в ядре ОС Microsoft Windows

- Отсутствие поддержки исключений ЯП C++ (равно как и ряда других возможностей данного языка) – доступна только их структурная обработка
- Особое поведение определённого по умолчанию обработчика структурных исключений (отказ системы, BSOD)
- Возможность обработки исключений перехватом обработчиков прерываний (для исторической справки, т.к. было актуально до появления механизма защиты памяти ядра Patch Guard – < WinXP x86-64, <= WinXP x86-64)

Перехват любых исключений: основные принципы

- Осмотрительная реализация обработчиков с учётом ограничений, специфичных для каждой операционной системы (преимущественно GNU/Linux)
- В случае ошибок доступа к памяти подходящая стратегия – аварийное завершение программы (возможно, с освобождением отдельных ресурсов и сохранением дампа памяти в файловую систему)
- В некоторых случаях возможно продолжение исполнения программы (например, при получении отдельных управляющих сигналов ОС GNU/Linux – `dd`, `SIGUSR1`)
- В редких случаях допустимо восстановление потока исполнения программы (посредством изменения регистров CPU текущего потока)

Встраиваемые системы (ОСРВ)

- Использование ЯП С++ для ядер встраиваемых ОС – редкость
- Код с исключениями на С++ не достаточно предсказуем в плане времени исполнения
- Исключительные ситуации перехватываются стандартными или переопределёнными пользователем обработчиками прерываний
- Полные сборки встраиваемых ОС часто содержат модули на С++

Интеграция кода на C++ во встраиваемые системы

- Выделение кода на C++ в отдельный программный модуль с Си-совместимым API
- Следование определённым (внутренним) стандартам кодирования (в частности, отказ от использования исключений C++ и RTTI)
- Сборка внедряемого модуля с поддержкой собственной реализации механизма исключений C++ ([Ссылка](#))

ВЫВОДЫ (ВЕРСИЯ 1)

не
использовать
исключения

**НИКОГДА
НЕ
ИСПОЛЬЗОВАТЬ
ИСКЛЮЧЕНИЯ**

УВОЛЬНЯТЬ ВСЕХ, КТО ИСПОЛЬЗУЕТ ИСКЛЮЧЕНИЯ

ВЫВОДЫ (ВЕРСИЯ 2)

- Не использовать функции `setjmp/longjmp` (Linux-атавизм)
- Не использовать глобальный (...) обработчик исключений C++
- Не использовать исключения для возврата кодов ошибок
- Не использовать оператор `throw` в качестве неявного оператора `goto`
- Документировать список типов генерируемых подпрограммой исключений
- Минимизировать объём размещаемого в блоках `try/__try` и `catch/__except` кода
- Использовать все преимущества концепции RAII (не использовать явно `malloc/free/...`)
- Завершать выполнение программы при перехвате не языкового исключения

Домашнее задание

- Освещение аспектов реализации механизма обработки структурных исключений в операционной системе Microsoft Windows для архитектуры ARM (32, 64)
- Сравнительный анализ реализаций механизма обработки исключений в C++ и C# (Python)
- Реализация на ЯП C механизма обработки исключений C++ во встраиваемой ОС (Embox, FreeRTOS, ...) для интеграции использующих парадигму ООП программных модулей
- Описание иерархий исключений в библиотеках STL, Qt и Boost (взгляд программиста)
- Анализ реализаций механизма SEH в компиляторах Borland C++ и Delphi

С++ для реверс-инженеров

20 лет спустя...

ЯП Delphi (Object Pascal): поддержка исключений

- Используются ключевые слова `try`, `except`, `finally` и `raise`
- Возможность перехвата как языковых, так и аппаратных исключений
- В блоке `except` ключевыми словами `on` и `do` можно задавать обработчики исключений конкретных классов
- Определённый по умолчанию обработчик исключений вызывается при отсутствии пользовательских обработчиков и обычно отображает сообщение об ошибке
- Есть некоторые различия в обработке исключений разными компиляторами в разных ОС (состав классов исключений, возможность перехвата отдельных типов исключений)

ЯП Delphi: реализация FPC (Windows, x86, v2.6.2)

```
sub_407380      proc near                                ; CODE XREF: sub_406700+26↑p
                push    ebp
                mov     ebp, esp
                mov     eax, offset TopLevelExceptionFilter
                push    eax                               ; lpTopLevelExceptionFilter
                call    SetUnhandledExceptionFilter
                leave
                retn
```

xrefs to sub_407380

Direction	Typ	Address	Text
Up	p	sub_406700+26	call sub_407380

ЯП Delphi: реализация FPC (Windows, x86, v2.6.2)

```
if ( byte_409E40 )
{
    byte_409E30 = 0;
    ExceptionInfo->ContextRecord->Eip += 3;
    ExceptionInfo->ExceptionRecord->ExceptionCode = 0;
    v1 = -1;
}
else
{
    v2 = -40;
}
}
else if ( (unsigned __int8)sub_406780() )
{
    ExceptionInfo->ExceptionRecord->ExceptionCode = 0;
    v1 = -1;
}
else
{
    v2 = -40;
}
LABEL_37:
if ( v2 && (unsigned __int8)byte_408640 < 0x10u )
{
    000066B4 TopLevelExceptionFilter:126 (4072B4)
```


ЯП Delphi: реализация FPC (Linux, x86-64, v3.0.4)

```
13 begin
14     y:= 0;
15     try
16         raise EMyOwnException.Create( 'Try!' );
17     except
18         on ex : EMyOwnException do
19             begin
20                 writeln( ex.Message );
21                 try
22                     x:= x div y;
23                     x:= p^;
24                 except
25                     on ex : EDivByZero do writeln( 'Division by zero' );
26                     on ex : EAccessViolation do writeln( 'Access violation' );
27                 end;
28             end;
29         on ex : Exception do writeln( 'Exception class' );
30     end;
31     writeln( 'Hello, world!' );
32 end.
```

ЯП Delphi: реализация FPC (Linux, x86-64, v3.0.4)

```
.text:0000000000400A50 rti_sigaction  proc near          ; CODE XREF: Set  
.text:0000000000400A50                               ; sub_422B50+1E↓
```

Occurrences of: syscall

Address	Function	Instruction	
.text:00000000004001AE	sub_40018E	syscall	; LINUX - sys_exit_group
.text:00000000004004CC	sub_4004C0	syscall	; LINUX - sys_getpid
.text:00000000004004FF	sub_4004F0	syscall	; LINUX -
.text:0000000000400532	sub_400520	syscall	; LINUX -
.text:0000000000400575	sub_400560	syscall	; LINUX -
.text:00000000004005B8	sub_4005A0	syscall	; LINUX -
.text:00000000004005FB	sub_4005E0	syscall	; LINUX -
.text:000000000040063F	sub_400620	syscall	; LINUX -
.text:0000000000400A4A	rt_sigreturn	syscall	; LINUX - sys_rt_sigreturn

ЯП Delphi: реализация FPC (Linux, x86-64, v3.0.4)

```
sub_4229C0    proc near                                ; CODE XREF: sub_422CA0+A8↓p
              lea     rsp, [rsp-8]
              lea     rax, unk_62D620
              mov     edi, 8                          ; SIGFPE
              mov     rsi, rax
              call    SetSignalHandler
              lea     rsi, unk_62D650
              mov     edi, 11                          ; SIGSEGV
              call    SetSignalHandler
              lea     rsi, unk_62D680
              mov     edi, 7                          ; SIGBUS
              call    SetSignalHandler
              lea     rsi, unk_62D6B0
              mov     edi, 4                          ; SIGILL
              call    SetSignalHandler
              lea     rsp, [rsp+8]
              retn
sub_4229C0    endp
```

ЯП Delphi: реализация Borland (Windows, x86, v7)

```
y:=365;
while y > 0 do
begin
    y:=y-1;
end;
try
    x:= x div y;
    i_time:= x;
except
    MessageDlg('ИСКЛЮЧЕНИЕ DIVISION BY ZERO! EXCEPT BLOCK', MB_ICONERROR, MB_OK);
end;

y:=123456789;
try
    i_time:= ptr^;
    i_time:= i_time + y;
except
    MessageDlg('ИСКЛЮЧЕНИЕ ACCESS VIOLATION! EXCEPT BLOCK', MB_ICONERROR, MB_OK);
end;

y:=987654321;
try
    i_time:= ptr^;
    i_time:= i_time + y;
finally
    try
        i_time:= ptr^;
    except
        MessageDlg('ИСКЛЮЧЕНИЕ ACCESS VIOLATION! EXCEPT BLOCK', MB_ICONERROR, MB_OK);
    end;
    MessageDlg('ИСКЛЮЧЕНИЕ ACCESS VIOLATION! FINALLY BLOCK', MB_ICONWARNING, MB_OK);
end;

MessageDlg('Вы должны выделить какую-нибудь задачу!', MB_ICONWARNING, MB_OK);
```

ЯП Delphi: реализация Borland (Windows, x86, v7)

```
• CODE:0046007A      mov     eax, 123456789
• CODE:0046007F      xor     edx, edx
• CODE:00460081      push    ebp
• CODE:00460082      push    offset loc_4600A8
• CODE:00460087      push    dword ptr fs:[edx]
• CODE:0046008A      mov     fs:[edx], esp
• CODE:0046008D      mov     edx, [ebp+var_4]
• CODE:00460090      mov     edx, [edx]
• CODE:00460092      mov     ds:dword_465934, edx
• CODE:00460098      add     ds:dword_465934, eax
• CODE:0046009E      xor     eax, eax
• CODE:004600A0      pop     edx
• CODE:004600A1      pop     ecx
• CODE:004600A2      pop     ecx
• CODE:004600A3      mov     fs:[eax], edx
• CODE:004600A6      jmp     short loc_4600C0
CODE:004600A8      ; -----
CODE:004600A8      loc_4600A8: ; DATA XREF: _TForm1_MenuItem3_1Click+46
• CODE:004600A8      jmp     @System@@HandleAnyException$qqrv ; System::__linkproc__
CODE:004600AD      ; -----
• CODE:004600AD      xor     ecx, ecx
• CODE:004600AF      mov     dl, 10h
• CODE:004600B1      mov     eax, offset _str_____ACCE.Text
• CODE:004600B6      call    sub_45F8D0
• CODE:004600BB      call    @System@@DoneExcept$qqrv ; System::__linkproc__ DoneExce
```

ЯП Delphi: реализация Borland (v7)

- Использование механизма структурных исключений для перехвата исключений (режим по умолчанию)
- Продолжение исполнения программы после перехвата аппаратного исключения (ошибка обращения к памяти, деление на ноль, ...) без явных на то указаний разработчика
- Исполнение блока `finally` до определённого по умолчанию обработчика при отсутствии определённого разработчиком обработчика

ЯП C++: реализация Borland (2002, 2010)

- Поведение скомпилированных тестовых программ идентично таковому у скомпилированных другими компиляторами этих же программ
- Используются функции стандартной библиотеки Borland Delphi, связанные с обработкой исключений (встраиваются в исполняемый файл)
- Нет существенных различий в реализации механизма обработки исключений в двух версиях компилятора Borland C++


ЯП С++: реализация Borland (2002, 2010)


```
@_InitExceptBlockLDTC proc near          ; CODE XREF: sub_401234+E↑p
                                         ; sub_401298+E↑p ...

arg_0      = byte ptr 4

        push    ebx
        mov     ebx, ebp
        add     ebx, [eax+8]
        mov     [ebx+8], eax
        lea     eax, [esp+4+arg_0]
        mov     [ebx+0Ch], eax
        mov     dword ptr [ebx+4], offset __ExceptionHandler
        mov     word ptr [ebx+10h], 0
        mov     word ptr [ebx+12h], 0
        mov     dword ptr [ebx+1Ch], 0
        mov     eax, fs:0
        mov     [ebx], eax
        mov     fs:0, ebx
        pop     ebx
        retn

@_InitExceptBlockLDTC endp
```

 xrefs to @_InitExceptBlockLDTC

Direction	Type	Address	Text
	p	_main+E	call @_InitExceptBlockLDTC

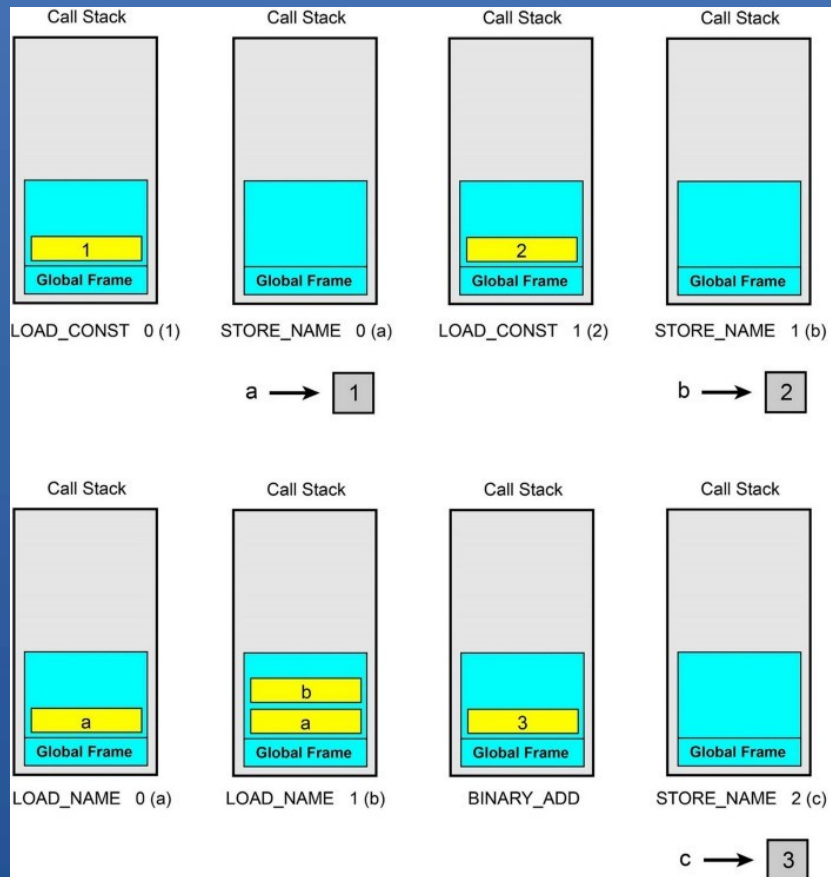
ЯП Python: поддержка исключений

- Простая структура исполняемых файлов (скомпилированных файлов программ) по сравнению с таковыми для других платформ на базе языковых виртуальных машин (в частности, Microsoft .NET)
- Компактность байткода стековой виртуальной машины Python (1 байт на опкод инструкции и несколько байт на аргумент)
- Использование разных стеков для разных нужд: стек вызовов (call stack), вычислительный стек (evaluation stack), стек блоков (block stack)

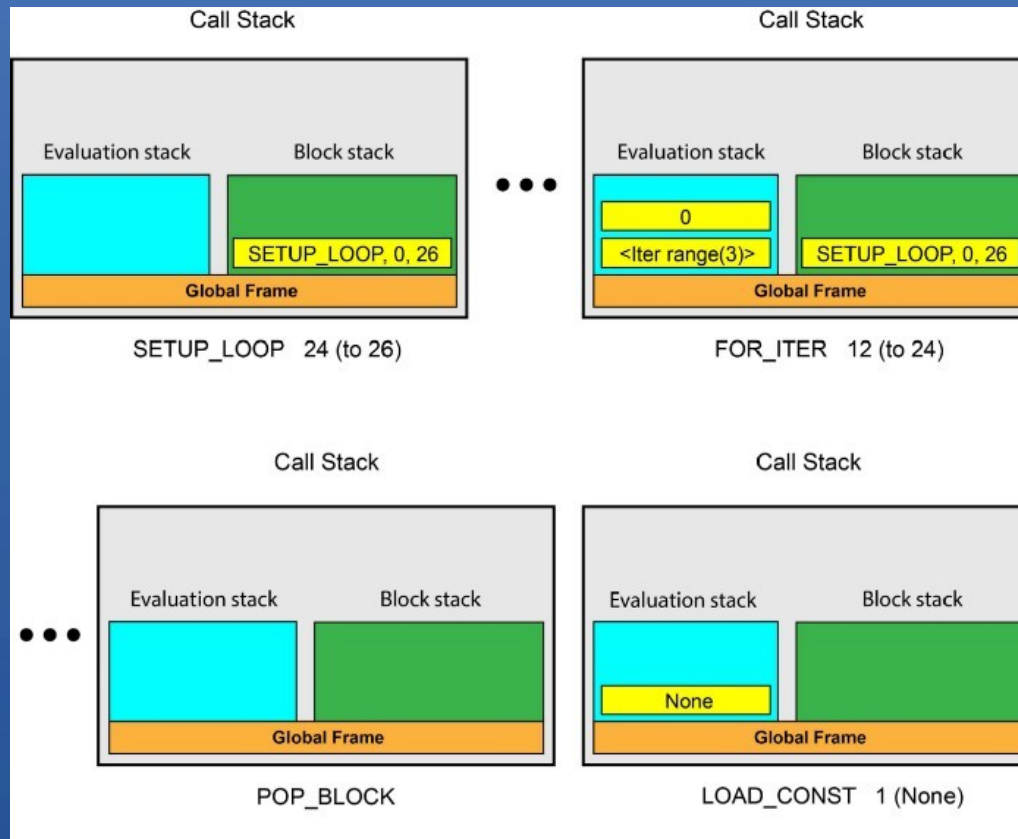
ЯП Python: поддержка исключений

- Стек блоков используется для операторов циклов и обработки исключений для раскрутки вычислительных стеков (выполняется при использовании операторов семейства `goto` – `break`, `continue`, `raise`, ...)
- Стек блоков и вычислительный стек создаются при каждом вызове функции или метода
- Большинство инструкций стековой VM Python оперируют вычислительным стеком фрейма исполняемой в текущий момент функции или глобальным фреймом
- Новая запись в стеке блоков создаётся соответствующей инструкцией байткода при входе в блок оператора цикла или оператора обработки исключений (`try-finally` или `try-except`)

ЯП Python: поддержка исключений (стек вызовов)



ЯП Python: поддержка исключений (стек блоков)



ЯП Python: реализация в версии 2.7

```
import sys
import dis

def test_fn():
    val = "abcd"

    try:
        x = int( val )
    except:
        print( "can't convert to integer" )
    finally:
        print( "finally block" )

test_fn()
dis.dis( test_fn )
```

```
can't convert to integer
finally block
5          0 LOAD_CONST          1 ('abcd')
           3 STORE_FAST           0 (val)

7          6 SETUP_FINALLY       35 (to 44)
           9 SETUP_EXCEPT      16 (to 28)

8          12 LOAD_GLOBAL         0 (int)
           15 LOAD_FAST           0 (val)
           18 CALL_FUNCTION       1
           21 STORE_FAST          1 (x)
           24 POP_BLOCK
           25 JUMP_FORWARD        12 (to 40)

9  >>      28 POP_TOP
           29 POP_TOP
           30 POP_TOP

10         31 LOAD_CONST          2 ("can't convert to integer")
           34 PRINT_ITEM
           35 PRINT_NEWLINE
           36 JUMP_FORWARD        1 (to 40)
           39 END_FINALLY
  >>      40 POP_BLOCK
           41 LOAD_CONST          0 (None)

12  >>      44 LOAD_CONST          3 ('finally block')
           47 PRINT_ITEM
           48 PRINT_NEWLINE
           49 END_FINALLY
           50 LOAD_CONST          0 (None)
           53 RETURN_VALUE
```

ЯП Python: реализация в версии 2.7

```
import sys
import dis

def test_fn():
    val = "fefe"

    try:
        x = int( val )
    except ValueError as ex:
        print( "can't convert to integer" )
    except Exception as ex:
        print( "global exception handler" )

test_fn()
dis.dis( test_fn )
```

```
can't convert to integer
5      0 LOAD_CONST          1 ('fefe')
      3 STORE_FAST           0 (val)

7      6 SETUP_EXCEPT      16 (to 25)

8      9 LOAD_GLOBAL          0 (int)
     12 LOAD_FAST            0 (val)
     15 CALL_FUNCTION        1
     18 STORE_FAST           1 (x)
     21 POP_BLOCK
     22 JUMP_FORWARD          47 (to 72)

9  >> 25 DUP_TOP
     26 LOAD_GLOBAL          1 (ValueError)
     29 COMPARE_OP
     32 POP_JUMP_IF_FALSE    48
     35 POP_TOP
     36 STORE_FAST           2 (ex)
     39 POP_TOP

10     40 LOAD_CONST            2 ("can't convert to integer")
     43 PRINT_ITEM
     44 PRINT_NEWLINE
     45 JUMP_FORWARD          24 (to 72)

11  >> 48 DUP_TOP
     49 LOAD_GLOBAL          2 (Exception)
     52 COMPARE_OP
     55 POP_JUMP_IF_FALSE    71
     58 POP_TOP
     59 STORE_FAST           2 (ex)
     62 POP_TOP

12     63 LOAD_CONST            3 ('global exception handler')
     66 PRINT_ITEM
     67 PRINT_NEWLINE
     68 JUMP_FORWARD          1 (to 72)
>> 71 END_FINALLY
>> 72 LOAD_CONST            0 (None)
     75 RETURN_VALUE
```