Лабораторная работа № 8
Структурная обработка исключений
(2 часа)

Содержание: фреймовая обработка исключений; фильтры и обработчики исключений; получение информации об исключении; финальная обработка исключений; обработка вложенных исключений.

Цель: изучить основы использования фреймовой и финальной обработки исключений; получить навыки использования расширений языка C++ в Microsoft Visual Studio для SEH, написания функции-фильтра и получения информации об исключении в пользовательских программах.

При разработке программного обеспечения достаточно часто требуется обработка исключительных ситуаций. Операционная система Windows поддерживает встроенный механизм структурной обработки исключений – SEH. В отличие от C++ механизмы SEH ориентированы не только на обработку программных ошибок, но и на обработку аппаратных ошибок.

Для использования механизма SEH в программах на C++ в Microsoft Visual Studio C++ вводится расширение из нескольких ключевых слов:

`__try` - отмечает фрейм – блок с охраняемым кодом;
`__except` - отмечает обработчик исключительной ситуации;
`__finally` - отмечает блок финальной обработки исключения.

Используется механизм структурной обработки исключений по следующей схеме:

```
__try
{
// код, в котором может возникнуть исключительная ситуация
}
__except (выражение-фильтр)
{
// обработка исключительной ситуации, если она возникла
}
```

Или по схеме:

```
__try
{
// код, в котором может возникнуть исключительная ситуация
}
__finally
{
// обработка исключительной ситуации, если она возникла
}
```

Выражение-фильтр определяет, как должна выполняться программа после возникновения исключения. Это выражение должно вернуть одну из констант:

– EXCEPTION_EXECUTE_HANDLER – управление должно быть передано обработчику исключений (коду внутри блока __except);
– EXCEPTION_CONTINUE_SEARCH – поиск обработчика исключения должен быть продолжен (выбирается следующий блок __except);
– EXCEPTION_CONTINUE_EXECUTION – управление должно быть передано в точку в которой возникло исключение (внутрь блока __try – делается попытка восстановления исключительной ситуации);

Если требуется в любом случае выполнять код обработчика исключительной ситуации, то соответствующую константу можно вписать вместо выражения-фильтра, например:

```
__try
{
…
}
__except (EXCEPTION_EXECUTE_HANDLER)
{
// обработка исключения
}
```

Внутри выражения-фильтра можно использовать функции:

DWORD GetExceptionCode(VOID); - позволяет получить код произошедшей исключительной ситуации;

LPEXCEPTION_POINTERS GetExceptionInformation(VOID); - позволяет получить подробную информацию о произошедшем исключении.

Задание 1. Добавьте в любую программу из предыдущих заданий разметку для охраняемых фреймов кода и обработку возможных исключительных ситуаций (ошибки целочисленной арифметики, ошибки работы с памятью и т.д.).

Примерами кода исключения могут быть:

EXCEPTION_ACCESS_VIOLATION – попытка прочитать или записать в блок памяти, для которого соответствующее право не определено;

EXCEPTION_BREAKPOINT – исключительная ситуация из-за точки останова в программе;

EXCEPTION_SINGLE_STEP – возникает при трассировке программы, когда выполнена одна инструкция;

EXCEPTION_ARRAY_BOUNDS_EXCEEDED – выход за пределы массива, если выполняется соответствующая проверка;

EXCEPTION_FLT_DENORMAL_OPERAND – один из операндов с плавающей точкой является ненормализованным;

EXCEPTION_FLT_DIVIDE_BY_ZERO – попытка деления на 0 в операциях с плавающей точкой;

и т.д.

Функция GetExceptionInformation позволяет получить детальную информацию об исключении – структуру:

```
typedef struct EXCEPTION_POINTERS {
    PEXCEPTION_RECORD ExceptionRecord;
    PCONTEXT Context;
} EXCEPTION_POINTERS, *PEXCEPTION_POINTERS;
```

В этой структуре ExceptionRecord указывает на структуру, содержащую информацию об исключении:

DWORD ExceptionCode; - код исключения

DWORD ExceptionFlags; - признак возможности восстановления после исключительной ситуации (0 – возможно продолжение программы, EXCEPTION_NONCONTINUABLE – не возможно);

struct _EXCEPTION_RECORD *ExceptionRecord; - указатель на следующую структуру, если блоки обработки исключения вложены один в другой;

PVOID ExceptionAddress; - адрес инструкции в программе по которому произошло исключение;

DWORD NumberParameters; - количество параметров, переданных в структуре ExceptionInformation;

ULONG_PTR ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS]; - массив параметров, описывающих исключительную ситуацию.

Context указывает на структуру CONTEXT, включающую содержимое регистров CPU в момент исключения.

Задание 2. Напишите программу, демонстрирующую восстановление после произошедшей при работе ошибки. Программа должна определять тип возникшей ошибки и пытаться исправить её. Например, после ошибки связанной с доступом по указателю, для которого не выделена память – выделить блок памяти требуемого размера или после ошибки, связанной с делением на ноль – исправить значение делителя.