Преобразования в стиле C++ Обработка исключительных ситуаций

Преобразования в стиле языка C++

Преобразования в стиле языка C++

Операция static_cast

Операция static_cast используется на этапе компиляции между:

- целыми типами;
- целыми и вещественными;
- целыми и перечислимыми;
- указателями и ссылками на объекты одной иерархии, при условии, что оно однозначно и не связанно с понижающим преобразованием виртуального базового класса.

Преобразования в стиле языка C++

Формат операции:

static_cast<тип>(выражение)

Результат операции имеет указанный тип, который может быть ссылкой, указателем, арифметическим или перечислимым типом.

При выполнении операции внутреннее представление может быть модифицировано, хотя численное значение остается неизменным.

Преобразования в стиле языка C++

Например,

```
float f = 100;
int i = static_cast<int>(f);
```

Преобразования подобного рода должны иметь серьезное основание. Результат преобразования остается на совести программиста.

Перечисленные преобразования попробуйте самостоятельно.

Преобразования в стиле языка C++

Мы же рассмотрим пример преобразований в иерархии родственных классов, что для нас имеет больший интерес.

Рассмотрим пример простой иерархии.

Преобразования в стиле языка C++

Базовый класс:

```
class Base
{
protected:
    int base;
public:
    Base(){};
    void Out();
};
void Base::Out()
{
cout << " Base class " << endl;
}
```

Преобразования в стиле языка C++

Производный класс:

```
class Derived : public Base
{
    int derived;
public:
    Derived():Base(){};
    void Out();
};
void Derived::Out()
{
    cout << " Derived class " << endl;
}
```

Преобразования в стиле языка C++

Рассмотрим несколько примеров преобразований между этими классами.

Преобразования «вверх», то есть от объекта производного типа к типу базового класса, относятся к стандартным преобразованиям и не требуют явных преобразований. Например,

```
Derived der;
Base base = der;
```

Преобразования в стиле языка C++

Выражение Base base = der; вполне оправдано, хотя правильнее (понятней) была бы запись: Base base = (Base)der;,

или Base base = static_cast<Base>(der);.

Как было сказано, что преобразования «вверх» относятся к стандартным, то последнее преобразование не является обязательным.

Преобразования в стиле языка C++

Рассмотренное преобразование типа производного класса возможно, если производный класс описан с обобществленным базовым классом, как в нашем примере:

class Derived : public Base {}

Если объявить базовый класс как защищенный или закрытый

class Derived : protected Base {}

Подобные преобразования будут возможны, но не доступны.

Преобразования в стиле языка C++

Преобразования объектов производного класса к типу базового встречается на практике достаточно часто, например при инициализации объекта базового класса объектом производного.

Более интересный пример – попытка преобразования «вниз», то есть из типа базового класса в производный тип.

Преобразования в стиле языка C++

Следует оговориться, что преобразования допустимы только уровне указателей или ссылок, но не объектов.

Для рассмотренных классов:

```
Base *ptr_Base = new Base();
ptr_Derived = static_cast<Derived *>(ptr_Base);
ptr_Derived->Out();
```

Что можно ожидать в этом случае?

Преобразования в стиле языка C++

Сработает функция производного класса и выведет «Derived class».

А что будет при вызове ptr_Base->Out(); ?

В данном случае вывод будет: «Base class».

Ничего странного, ничего не обычного.

Изменим несколько объявления:

```
Base *ptr_Base = new Derived();
```

Преобразования в стиле языка C++

Такое объявление также допустимо, указатель на базовый класс инициализируется значением указателя на производный.

Вызов `ptr_Base->Out();` приведет к активизации функции базового класса.

Для того чтобы активизировать функцию производного класса, функцию базового класса нужно описать как виртуальную: `virtual void Out();`

Это проявление полиморфизма в C++.

Преобразования в стиле языка C++

Операция dynamic_cast

Эта операция используется в основном для преобразования указателей и ссылок на объекты базового типа в указатели и ссылки на производный тип. При этом во время выполнения программы появляется возможность проверки (контроля) допустимости преобразований.

Преобразования в стиле языка C++

Несложно догадаться, что преобразования будут выполняться в период выполнения программы.

Общий формат операции:

dynamic_cast<тип>(выражение)

Выражение должно быть указателем или ссылкой на класс, тип – базовым или производным для данного класса.

Преобразования в стиле языка C++

В случае успешного выполнения операции формируется результат заданного типа, в противном случае для указателей результат равен нулю, а для ссылок порождается исключение bad_cast. Если заданный тип не относится к одной иерархии, преобразования не допускаются.

Все дальнейшие рассуждения при рассмотрении наследования и полиморфизма.

Преобразования в стиле языка C++

Операция reinterpret_cast

Операция reinterpret_cast применяется для преобразования не связанных между собой типов, например, указателе в целые типы или наоборот, а также указателей типа void в конкретный тип. При этом внутреннее представление данных не меняется, меняется только точка зрения компилятора на данные.

Преобразования в стиле языка C++

Рассмотрим пример:

```
struct Struct
{
    int str_int;
    string str_string;
    Struct(int s_i, string s_s):
    str_int(s_i), str_string(s_s){};
    void Out()
    { cout << str_int << ' ' << str_string << endl; }
};
```

Преобразования в стиле языка C++

Далее использование объекта этого типа через указатель на тип void *

```
Struct str(120, "string");
void *ptr_void = &str;
reinterpret_cast<Struct *>(ptr_void)->Out();
```

Этот пример того, что не следует делать в практическом программировании, поскольку результат операции останется на совести программиста.

Преобразования в стиле языка C++

Практическое использование этого оператора при форматированном вводе-выводе числовых величин.

```
#include<fstream>
#include<iostream>
const int MAX = 100;
int buffer[MAX];
```

Преобразования в стиле языка C++

```
// создаем выходной поток
   ofstream os(“data.dat”, ios::binary);
// записываем в него
   os.write(reinterpret_cast<char *>(buffer),
   Max*sizeof(int));
// закрываем поток
   os.close();
```

Преобразования в стиле языка C++

В данном случае использование оператора reinterpret_cast вполне оправдано, поскольку его действие позволяет существенно сократить ресурсы памяти.

Обработка исключительных ситуаций

Исключительная ситуация или исключение – это возникновение непредвиденной или аварийной события, которое может порождаться самыми различными причинами, например из-за недостаточности распределяемых ресурсов. Эти средства можно использовать для обработки ошибок, возникающих при работе программы.

Обработка исключительных ситуаций

Ошибку в программе следует рассматривать как частный случай исключительной ситуации.

Если не использовать средств обработки исключений, то программа завершается с выдачей сообщения об ошибке. Обработка же исключения дает возможность продолжить выполнение программы и кроме того, позволит диагностировать исключение.

Обработка исключительных ситуаций

Общий механизм обработки исключений

Формат программного кода, в котором может произойти ошибка, должен входить в *контролируемый блок*. Контролируемый блок представляет собой составной оператор, перед которым записано служебное слово try (проба, тест, испытание).

Обработка исключительных ситуаций

Обработка исключительной ситуации реализуется следующим образом:

1. Обработка исключения начинается с проверки появления ошибки (должен предусмотреть разработчик). Во фрагменте программного кода, где обнаружена ошибка, генерируется исключение (обязанность разработчика). Для генерации исключения используют стандартное слово throw (бросать, возбуждать), с параметром, определяющим вид исключения.

Обработка исключительных ситуаций

Параметр исключения может быть константой, переменной, или объявлением класса и используется для передачи информации об исключении его обработчику.

2. Отыскивается соответствующий обработчик (catch), ему передается управление и информация об исключении (например из throw).

Обработка исключительных ситуаций

3. Если обработчик исключения не найден, вызывается стандартная функция terminate(), которая в свою очередь вызывает функцию abort(), которая аварийно завершает текущий процесс. Программист может переопределить (перегрузить) функцию terminate() самостоятельно с учетом особенностей исходной задачи.

Обработка исключительных ситуаций

Синтаксис исключений

Служебное слово try предназначено для обозначения контролируемого блока – кода, в котором может генерироваться исключение. Блок заключен в фигурные скобки:

```
try
{
 ..............
}
```

Обработка исключительных ситуаций

Блок содержит последовательность операторов, включая вызовы функций, которые проверяются на возникновение исключения. Такой блок называют защищенным блоком. В рамках защищенного блока должно быть выражение, вызывающее исключение.

Обработка исключительных ситуаций

Генерация исключения происходит по служебному слову throw, которое используется либо с выражением, либо без него:

throw[выражение];

Отметим, что служебное слово throw (бросать) было использовано вместо слов signal и raise, так как стандартная С-библиотека уже имеет функции с такими именами.

Обработка исключительных ситуаций

Тип выражения, стоящего после throw, определяет тип порождаемого исключения.

Тип может быть стандартным или типом, определенный пользователем (класс, структура).

Чаще всего исключение порождается не в самом блоке try, а в функциях, вызываемых в блоке.

Обработка исключительных ситуаций

При необходимости try-блоки могут вкладываться друг в друга. В этом случае исключение передается на более высокий уровень с помощью throw без выражения.

Обработчик исключения начинается с ключевого слова cath (перехватить), за которым в круглых скобках следует тип обрабатываемого исключения.

Обработчики обычно располагают за контрольным блоком try.

Обработка исключительных ситуаций

Синтаксис обработчиков напоминает определение функции с одним параметром – типом исключения. Существует три формата исключений.

```
// формат 1
catch( тип имя)
{
 // тело обработчика
}
```

Обработка исключительных ситуаций

```
// формат 2
   catch( тип )
   {
    // тело обработчика
   }
// формат 3
   catch(  … )
   {
    // тело обработчика
   }
```

Обработка исключительных ситуаций

Первый формат применяется, когда имя параметра используется в теле обработчика для выполнения каких-либо действий. Чаще всего – это вывод об исключении.

Второй формат не предусматривает использование информации об исключении, его роль играет только тип.

Третий формат обозначает, что обработчик перехватывает все исключения.

Обработка исключительных ситуаций

Обычно обработчики по третьему формату записывают в самую последнюю очередь, то есть « в противном случае».

Перехват исключений

Когда с помощью throw генерируется исключение, функция исполнительной системы C++ выполняет следующие действия:

Обработка исключительных ситуаций

1. Создается копия выражение - параметра из throw в виде статического объекта, который существует до тех пор, пока не исключение не будет обработано.

2. В поисках подходящего обработчика исключения раскручивается стек. При раскрутке стека вызываются деструкторы локальных объектов, выходящих из области действия и деструкторы копий аргументов классов, передаваемых по значению.

Обработка исключительных ситуаций

3. Передается статический объект (копию выражения из throw) и передается управление обработчику, имеющему параметр, совместимый по типу с этим объектом.

Обработчик считается найденным, если тип выражения, указанного после throw:

- тот же, что и тип в параметре catch (параметр может быть типа T, или T&, или const T&, где T – тип исключения);

Обработка исключительных ситуаций

- является производным от указанного в параметре catch (если наследование производилось с ключом public);
- является указателем, который может быть преобразован по стандартным правилам преобразования к типу указателя в параметре catch.

Обработка исключительных ситуаций

*Простой пример:

```
int main()
{
    int d, m;
    double dp;
try
{
    cout << " d "; cin >> d;
    cout << " m "; cin >> m;
    if(m <= 0) throw d;
    dp = double(d)/double(m);
    cout << " dp: " << dp << endl;
}
```

Обработка исключительных ситуаций

```
catch(int e)
{
    cout << e << endl;
}
    return 0;
}
```

Обработка исключительных ситуаций

Более сложный пример. Рассмотрим класс.

```
class Hello
{
public:
    Hello()
    { cout << " Hello! " << endl; }
    ~Hello()
    { cout << " Bye! " << endl; }
};
```

Обработка исключительных ситуаций

И две функции:

```
void func_1()
{
    ifstream ifs("\\INVALID\\FILE\\NAME");
    if(!ifs)
    {
     cout << " Генерация исключения " << endl;
     throw " Ошибка открытия файла! \n\n ";
    }
}
```

Обработка исключительных ситуаций

```
void func_2()
{
   Hello H;
   //   Вызов функции, генерирующую
   исключение
   func_1();
}
```

Обработка исключительных ситуаций

```
int main()
{
    try
    {
     func_2();
     cout << " Выход из try блока " << endl;
    }
    catch(const char *s)
    {
    cout << " Обработчик const char * " << s << endl;
    }
```

Обработка исключительных ситуаций

```
    catch( ... )
    {
     cout << " Обработчик остальных исключений " << endl;
     return 20;
    }
    return 0;
}
```

Обработка исключительных ситуаций

Обработка исключительных ситуаций

Обработка исключительных ситуаций

Обработка исключительных ситуаций

Обработка исключительных ситуаций

Обработка исключительных ситуаций

Обработка исключительных ситуаций

Обработка исключительных ситуаций