Практическое занятие №10

«Стандартные методы приведения типов»

(Продолжительность работы 2 часа)

Цели:

Изучить механизм стандартного приведения типов C++.

1. **Краткие теоретические сведения**

xxx_cast

В языке C++ существует четыре разновидности приведения типа. Все четыре типа записываются в виде:

xxx_cast<type_to>(expression_from)

Материал: http://alenacpp.blogspot.ru/2005/08/c.html

Лучшая практика по приведению типов: не делать этого. Потому что, если в программе потребовалось приведение типов, значит в этой программе с большой долей вероятности что-то неладно. Для довольно редких ситуаций, когда это все-таки действительно нужно, есть четыре способа приведения типов. Старый, оставшийся со времен C, но все еще работающий, лучше не использовать вовсе. Хотя бы потому, что конструкцию вида (Тип) очень сложно обнаружить при чтении кода программы.

const_cast

Самое простое приведение типов. Убирает так называемые cv спецификаторы (cv qualifiers), то есть const и volatile. volatile встречается не очень часто, так что более известно как приведение типов, предназначенное для убирания const. Если приведение типов не удалось, выдается ошибка на этапе компиляции.

При использовании остальных приведений типов cv спецификаторы останутся как были.

```
int i;
const int * pi = &i;
// *pi имеет тип const int,
// но pi указывает на int, который константным не является
int* j = const_cast<int *> (pi);
```

static_cast

Может быть использован для приведения одного типа к другому. Если это встроенные типы, то будут использованы встроенные в C++ правила их приведения. Если это типы, определенные программистом, то будут использованы правила приведения, определенные программистом.

static_cast между указателями корректно, только если один из указателей - это указатель на void или если это приведение между объектами классов, где один класс является наследником другого. То есть для приведения к какому-либо типу от void*, который возвращает malloc, следует использовать static_cast.

```
int * p = static_cast<int*>(malloc(100));
```

Если приведение не удалось, возникнет ошибка на этапе компиляции. Однако, если это приведение между указателями на объекты классов вниз по иерархии и оно не удалось, результат операции undefined. То есть, возможно такое приведение: static_cast<Derived*>(pBase), даже если pBase не указывает наDerived, но программа при этом будет вести себя странно.

dynamic_cast

Безопасное приведение по иерархии наследования, в том числе и для виртуального наследования.

dynamic_cast<derv_class *>(base_class_ptr_expr)

Используется RTTI (Runtime Type Information), чтобы привести один указатель на объект класса к другому указателю на объект класса. Классы должны быть полиморфными, то есть в базовом классе должна быть хотя бы одна виртуальная функция. Если эти условие не соблюдено, ошибка возникнет на этапе компиляции. Если приведение невозможно, то об этом станет ясно только на этапе выполнения программы и будет возвращен NULL.

dynamic_cast<derv_class &>(base_class_ref_expr)

Работа со ссылками происходит почти как с указателями, но в случае ошибки во время исполнения будет выброшено исключение bad_cast.

reinterpret_cast

Самое нахальное приведение типов. Не портируемо, результат может быть некорректным, никаких проверок не делается. Считается, что вы лучше компилятора знаете как на самом деле обстоят дела, а он тихо подчиняется. Не может быть приведено одно значение к другому значению. Обычно используется, чтобы привести указатель к указателю, указатель к целому, целое к указателю. Умеет также работать со ссылками.

reinterpret_cast<whatever *>(some *)
reinterpret_cast<integer_expression>(some *)
reinterpret_cast<whatever *>(integer_expression)

Чтобы использовать reinterpret_cast нужны очень и очень веские причины. Используется, например, при приведении указателей на функции.

Что делает приведение типов в стиле С: пытается использовать static_cast, если не получается, использует reinterpret_cast. Далее, если нужно, использует const_cast .

Примеры

unsigned* и int* никак не связаны между собой. Есть правило приведения между unsigned (int) и int, но не между указателями на них. И привести их с помощьюstatic_cast не получится, придется использовать reinterpret_cast. То есть вот так работать не будет:

```
unsigned* v_ptr;
cout << *static_cast<int*>(v_ptr) <<endl;
```

Приведение вниз по иерархии:

```
class Base { public: virtual ~Base(void) { } };
class Derived1 : public Base { };
class Derived2 : public Base { };
class Unrelated { };

Base* pD1 = new Derived1;
```

Вот такое приведение корректно: dynamic_cast<Derived1 *>(pD1);

А вот такое возвратит NULL: dynamic_cast<Derived2 *>(pD1);

Никак не связанные указатели можно приводить с помощью reinterpret_cast:

```
Derived1  derived1;
Unrelated*  pUnrelated = reinterpret_cast<Unrelated*>(&derived1);
```

Пример использования static_cast:

```
int*   pi;
```

```
void*  vp = pi;
char*  pch = static_cast<char*>(vp);
```

Примеры использования reinterpret_cast:

```
float f (float);
struct S {
  float x;
  float f (float);
  } s;
void g () {
  reinterpret_cast<int *>(&s.x);
  reinterpret_cast<void (*) ()>(&f);
  reinterpret_cast<int S::*>(&S::x);
  reinterpret_cast<void (S::*) ()>(&S::f);
  reinterpret_cast<void**>(reinterpret_cast<long>(f));
}
```

Приведение в стиле С можно использовать, чтобы избавиться от значения, возвращаемого функцией. Польза от этого сомнительная, правда...

```
string sHello("Hello");
(void)sHello.size();  // Throw away function return
```

Также я видела использование приведение типов в стиле С для приведения к приватному базовому классу, но для этого можно использовать и reinterpret_cast

2. **Практическое задание (100%)**

Создать шаблон заданного класса. Определить конструкторы, деструктор, перегруженную операцию присваивания (“=”) и операции, заданные в варианте задания.

Варианты заданий

1. Выяснить для каких преобразований типов, возможно применить static cast? Придумать возможные варианты преобразований с сужением типа и с расширением типа. И отразить это в тестовой программе (проверка преобразований типа static_cast использовать встроенные типы).

2. Написать шаблон функции для вывода дампа памяти переменной в шестнадцатеричном виде. Для типов float, double найти представление вещественного числа в интернете и сопоставить представление числа в памяти вашего ПК и числа. Отразить схему представления вещественного числа в памяти ПК.
   Для примера приведен листинг вывода содержимого переменной int.

```cpp
#include <iostream>

using namespace std;

int main()
{

   int val = 0x1234;
   char* p2ch = reinterpret_cast<char*>(&val); // Noerr
   // char* p2ch = static_cast<char*>(&val); // err

   cout << "sizeof val in bytes "<< sizeof(val) <<endl;

   cout << "P2ch = 0x"<< hex << static_cast<int>(*p2ch)<< " 0x" <<
   static_cast<int>(*(p2ch+1))<< " 0x" <<
   static_cast<int>(*(p2ch+2))<< " 0x" <<
   static_cast<int>(*(p2ch+3))<< endl;

   cout << "Hello World" << endl;

   return 0;
}
```

a. Сделать рисунок представления числа типа int в памяти ПК.
b. Как с помощью static_cast заставить работать закомментированную строку в приведенной выше программе?

3. Список рекомендуемой литературы

1. Павловская Т. А.C/C++. Программирование на языке высокого уровня : для магистров и бакалавров : учеб. для вузов / Т. А. Павловская. - Гриф МО. - Санкт-Петербург: Питер, 2013. - 460 с. : ил.

2. Professional C++, 3rd Edition. Marc Gregoire. ISBN: 978-1-118-85805-9. Paperback 984 pages. September 2014

4. Контрольные вопросы

1. Какое количество неявных преобразований типов выполнит компилятор для функции с N аргументами?
2. Почему по возможности, следует избегать преобразования типов в стиле Си?
3. Для каких целей введена операция const_cast, и в каком случае результат действия программы становится неопределенным после приведения типа?
4. Опишите синтаксис операции приведения типа xxx_cast.
5. Возможно ли преобразование указателя базового класса в указатель на производный класс той же иерархии с помощью приведения типа? Привести пример кода.
6. В каком случае при динамическом преобразовании типа dynamic_cast будет корректно выполнена проверка допустимости преобразования?
7. Что нужно сделать, чтобы включить проверку RTTI в программе?
8. Почему требуется проверять явным образом результат применения операции dynamic_cast?
9. Почему недопустимо вместо dynamic_cast использовать приведение типа в стиле Си (например так: C* c = (C*) p;)?