Тема: Приведение типа

- Использование конструкторов для приведения типа
- Операция приведения типа

- Приведение типа для значения, указателя и ссылки

Приведение типа в C++

- Не влияет на приводимый объект
- Генерируется временный анонимный объект требуемого типа и тем же значением
- Анонимный объект может быть в стеке или в регистре соответствующего типа
- Действия компилятора
  - выделение памяти для анонимной переменной
  - инициализация анонимной переменной
  - использование анонимной переменной

Упрощенный пример

- Приведение типа обеспечивается генерацией компилятором кода, содержание которого соответствует следующему примеру

Пример

```
float fun( int i , int j ) {
    return (float) i / j ;
}
// компилятор генерирует
float fun( int i , int j ) {
    float temp_i = i ,  temp_j = j ;
    return temp_i / temp_j ;
}
```

Приведение к типу класса

- Обеспечивает конструктор с *одним параметром*

Пример

```
class complex {
  public:
    complex ( );
    complex (float re );
    complex (float re , float im );
  private:
     float  real ,  image ;
}
```

```
void func( const complex & );
int main (int , char*[ ] ) {
    func( complex(4.0) ) ;
    func( (complex) 4.0 ) ;

// сравнить с
    func( complex(0.5, 0.1) ) ;
    return 0 ;
}
```

Приведение к типу *String*

```
#include "mystring.h"
#include <string.h>
int main (int , char*[ ] ) {
    String name ;
    name = (String) "Bilbo" + String("Baggins") ;
    name.print( ) ;
    ( (String) "Bilbo Baggins" ).print( ) ;
    return 0 ;
}
```

Упрощение класса *String*

```
const int max_string_length = 128;
class String {
    public:
      String() ;
      String( const char* ) ;
      String( const String& ) ;
      ~String() ;
      String& operator= ( const String& );
      String& operator= ( const char* );
      int  length( ) const ;
      int  read( );                  // чтение из stdin
      void  print( ) const;// вывод в  stdout
      char& operator[ ] ( int );
      const char& operator[ ] ( int ) const;
      String substring (int start, int len) const;
      friend String operator+ (const String&, const String& );
      friend String operator+ (const String&, const char* );
      friend String operator+ (const char*, const String& );
    private:
      char text [ max_string_length+1 ];
};
```

Преобразование типа не происходит для вызывающего объекта

```
#include "mystring.h"
#include <string.h>
ind main (int , char*[ ] ) {
    String name ;
    name = "Bilbo" + "Baggins" ;
    name.print() ;
    ( (String) "Bilbo Baggins" ).print() ;
    return 0 ;
}
```

Конструкторы и передача аргументов

```
#include "mystring.h"

String sentence(String words, char* punctuation = "." ) ;
void main (int , char*[ ] ) {
    String statement = "Hello, Bilbo" ;
     sentence(statement).print() ;
     sentence("Do you have a ring" , "?").print() ;
}
String sentence(String words, char* punctuation ) {
    return words+punctuation ;
}
```

Неоднозначности

- Правила разрешения ссылок
  1. точное соответствие типа аргументов
  2. тривиальные преобразования типа (**type[]→type***, **type→const type**) или преобразования с повышением точности (**char→int**, **short→int**, **float→double**, **double→long double** и т.п.)
  3. стандартные преобразования (**int→double**, **double→int**, **type*→void***, **int→unsigned, unsigned → int** и т.п.)
  4. преобразования, определенные пользователем
  5. неконтролируемое количество и тип аргументов (…)
- Если есть несколько возможных вариантов, то неоднозначность разрешается явным приведением типа

Пример неоднозначности

```
#include "mystring.h"
class example {
    public:
     example( const char* );
     // …
} ;
void f1( const String& );
void f1( const example& );
void main( int, char*[ ] ) {
//  f1("Hello, World") ;   неоднозначность
    f1( (String)"Hello, World") ;
    f1( (example)"Hello, World") ;
//  или  надо  void f1( const char*)
}
```

Операция преобразования типа

- Проблема преобразования типа класса в другой тип

  Пример

  String s=“Пробная строка” ;

  strlen(*s***)**; // ошибка

- Проблема преобразования в другой пользовательский тип (м.б. решена конструктором того типа, но не всегда)

- Преобразование во встроенный тип – операция приведения типа

  operator <*type_name*> **()**

Пример

- ***Пример, как не надо***

```
String::operator char* ( ) {  return text; }
```

Опасно!

```
String s ;
char very_big_array [1000] ;
strcpy( s , very_big_array ) ;      // будет худо !
```

- ***Пример, как надо***

```
String::operator const char* () const {  return text; }
```

Безопасно

```
String s ;
char very_big_array [1000] ;
strcpy(s , very_big_array ) ; // ошибка компилятора
```

Пересмотренный класс *String*

```
const int max_string_length = 128;
class String {
    public:
     String() ;
     String( const char* ) ;
     String( const String& ) ;
     ~String() ;
     operator const char*( ) const;
     String& operator= ( const String& );
     int  length( ) const ;
     int  read( );              // чтение из stdin
     void  print( ) const; // вывод в  stdout
     char& operator[ ] ( int );
     const char& operator[ ] ( int ) const;
     String substring (int start, int len) const;
     friend String operator+ (const String&, const String& );
    private:
     char text [ max_string_length+1 ];
};
```

Еще раз о неоднозначностях

```
#include "mystring.h"
#include <stdio.h>
String operator– ( const String& , const String& );
void main( int, char*[ ] ) {
    FILE* pf ;
    String filename = "\tmp\test" ;
    pf = fopen( filename , "w+" ) ; //   теперь можно
    strcpy( filename  , "abc" ) ;     //   запрещено
    String name = "Bilbo Baggins";
    String firstname ;
    firstname = name – "Baggins" ; // теперь неоднозначно

}
```

Вместо преобразования типа

- Перегрузка операции преобразования типа требует большой осторожности
  - включает для типа механизм неявных преобразований по правилам **C/C++**
  - может порождать при компиляции неоднозначности
- Использование обычного метода класса
  - не так красиво, с т.з. синтаксиса
  - но безопасно с т.з. неявных преобразований
- Например, если требуется иметь возможность использовать объект типа *String* там, где требуется **char*** , то это можно сделать так:

Пример для класса *String*

```
#include <stdio.h>
const int max_string_length = 128;
class String {
    public:
     // …
      const char* as_c_string ( ) const { return text; };
      // …
    private:
      char text [ max_string_length+1 ];
};
void main( int, char*[ ] ) {
    FILE* pf ;
   String filename = "\tmp\test" ;
    pf = fopen( filename.as_c_string() , "w+" ) ;
    strcpy( filename.as_c_string() , "abc" ) ;     //  запрещено
}
```

Преобразование указателей и ссылок

- Преобразования типа объекта *преобразует объект*
  - создает временный объект
  - не может нарушить целостности самого объекта

```
float f_var = 3.14;
printf("%d", (int) f_var);
```

- Преобразование указателя или ссылки *не преобразует объект*
  - создается временный указатель (а не объект)
  - изменяется трактовка содержимого объекта (памяти)
  - противоречит принципу инкапсуляции (требуется знать внутреннее представление объекта в памяти)

```
printf("%d", *((int*) &f_var) );
printf("%d", (int&) f_var);
```

РЕЗЮМЕ

- Средства приведения типа необходимы в языках со строгой типизацией
- Позволяет уменьшить набор методов класса, но за счет повышение косвенности вызовов
- В классах **C++** приведение типа управляется
  - конструктором с одним параметром
  - операцией приведения типа
  - приведение типа можно запретить
- Активное включение средств приведения типа в классы может приводить к неоднозначности
- Неоднозначность может быть снята указанием явных преобразований типов