Тема: Константы

- Правила использования констант
- Константы и аргументы функций
- Постоянные методы
- Операция индексации
- Оптимизация класса ***String***

Константы

- Объявляются с ключевым словом **const**.
- Должны инициализироваться
- Значение константы не может быть изменено
- Ключевое слово **const** значимо при перегрузке
- Могут передаваться функциям только
  - для формальных аргументов, передаваемых по значению
  - для формальных аргументов, объявленных как ссылки или указатели на константу
- Постоянные объекты могут вызывать только постоянные методы.

Ссылки на константы

```
#include "mystring.h"
void f1( String& );
void f2( const String& );

void main( int, char** ) {
    String  s ;  s = "hello world" ;
    const String  cs = s ;
    f1(s) ;      // правильно
    f1(cs) ;     // неправильно
    f2(s) ;      // правильно
    f2(cs) ;     // правильно
}
void f1( String& str )          // может изменять аргумент
{ str = "new value"; }
 void f2( const String& str ) // не может изменять аргумент
 { str.print(); }
```

Указатели на константы

```
#include "mystring.h"
void f1( String* );
void f2( const String* );
void main( int, char** ) {
    String  s ;  s = "hello world" ;
    const String  cs = s ;
    f1(&s) ;      // правильно
    f1(&cs) ;     // неправильно
    f2(&s) ;      // правильно
    f2(&cs) ;     // правильно
}
void f1( String* str )            // может изменять (*str)
{ *str = "new value"; }
 void f2( const String* str )  // не может изменять (*str)
 { str->print(); }
```

Константы и вызов по значению

```
#include "mystring.h"
void f1( String );
void f2( const String );

void main( int, char** ) {
    String  s ;  s = "hello world" ;
    const String  cs = s ;
    f1(s) ;      // правильно
    f1(cs) ;     // правильно
    f2(s) ;      // правильно
    f2(cs) ;     // правильно
}
void f1( String str )          // может изменять str, но не оригинал
{ str = "new value"; }
 void f2( const String str )  // не может изменять str
 { str.print(); }
```

Постоянные методы

```
#include "mystring.h"
class Employee {
    public:
     void  set_name( const String& );
     String get_name( ) const ;
    private:
     String name;
     float salary ;
};
void Employee::set_name( const String& s ) {
    name = s;
 }
String Employee::get_name( ) const {
    return name;
 }
```

Вызов постоянных методов

```
#include "mystring.h"
#include "employee.h"

void main( int, char** ) {
    String  jones, smith ;
    jones = "Jones" ;
    smith = "Smith" ;
    Employee  e ;
    e.set_name(jones) ;          // правильно
    const Employee  ce = e;
    e.set_name(smith) ;
    ce.set_name(smith) ;         // не правильно
    ce.get_name().print() ;      // правильно
    e.get_name().print() ;       // правильно
}
```

Операция индексации **operator[ ]**

- Доступ к элементу по его номеру используется для многих типов данных
- Для операции индексации надо обеспечить синтаксис (на примере типа ***String***)

```
String firstname ;
firstname = “bilbo” ;
char ch ;
ch = firstname[0] ;
firstname[0]  = ‘B’;
const String name =      firstname ;
ch = name[0] ;
```

- и запретить синтаксис

```
name[0]  = ‘B’;
```

- ***Должна быть методом***

Оптимизация класса *String*

```
const int max_string_length = 128;
class String {
    public:
     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 ];
};
```

Обновление методов

```
#include "mystring.h"
String& String::operator= ( const char* )
    { // код тела прежний }
int String::lenth( ) const
    { // код тела прежний }
int String::read( )
    { // код тела прежний }
void String::print( ) const
    { // код тела прежний }
String String::substring (int start, int len) const
    { // код тела прежний }
String operator+ (const String&, const String& )
     { // код тела прежний }
String operator+ (const String&, const char* )
    { // код тела прежний }
String operator+ (const char*, const String& )
    { // код тела прежний }
```

Новые методы **operator[ ]**

```
#include "mystring.h"
#include <stdio.h>
char& String::operator[ ]( int i ) {
    if( length() <= i || i < 0 ) {
      fprintf(stderr, "illegal index (%d) for string \"%s\".\n", i , text );
      exit(1);
    }
    return text[i];
}
const char& String::operator[ ]( int i ) const {
    if( length() <= i || i < 0 ) {
      fprintf(stderr, "illegal index (%d) for string \"%s\".\n", i , text );
      exit(1);
    }
    return text[i] ;
}
```

Использование *String*

```
#include "mystring.h"

void main( int, char** ) {
    String  s ;
    s = "Bilbo Baggins";
    const String  cs = s ;
    char ch ;
    // теперь определен синтаксис
    cs + s ;   "Hello, " + cs ;   cs + "." ;
    cs.length() ;  cs.print() ;   cs.substring(0,6) ;
    ch = cs[0] ;
    // и запрещен синтаксис
    cs = " " ;
    cs.read() ;      cs[0] = ch ;
}
```

РЕЗЮМЕ

- В **C++** можно тонко регулировать права доступа к данным с помощью описателя **const**
- Описатель **const** лишает объект свойства быть ***lvalue***
- При вызове функции константа может быть в аргументе, если передача аргумента происходит
  - по значению: **void** *func*(*type*);
  - по ссылке на константу: **void** *func*(**const** *type*&);
  - по указателю на константу: **void** *func*(**const** *type**);
- Для постоянного объекта можно вызывать только постоянные методы класса
- Использование **const** расширяет область применимости функции

Упражнение

- Оптимизируйте методы в классе *String*, используя ссылки и константы
- Измените методы для операций “**==**” и “**!=**” так, чтобы они могли использоваться с постоянными объектами типа ***String.***
- Протестируйте следующий синтаксис:

```
#include “mystring.h"

#include <stdio.h>
void main(int, char *[ ]) {
    String a;
    a = "hello world";
    const  String b = a;
    if ( a == b  )     printf(“part 1 works.\n");
    if ( b == a  )     printf(“part 2 works.\n");
    if ( !( a != b  )   printf(“part 3 works.\n");
    if ( !( b != a  )   printf(“part 4 works.\n");
}
```