Тема: Ссылки

- Что такое ссылка?
- Использование ссылок в языке **C++**
- Ссылки, как аргументы функций
- Ссылки, как возвращаемые значения
- Операция индексации

Что такое ссылка?

- Ссылка – альтернативное имя переменной.
- Ссылки введены в язык **C++** для расширения возможностей передачи параметров в функции ***по адресу***.

  Ссылка – адрес, разыменовываемый в момент использования
- Ссылка принципиально и формально отличается от указателя
  - со ссылкой не связано распределение памяти, а указатель – это переменная в памяти, хранящая адрес
  - синтаксис использования ссылки идентичен синтаксису использования простой переменной, а указатель используется с операциями “ * ”, “–>”

Использование ссылок

- Ссылка объявляется следующим образом:

  typename & linkname

- Локальная ссылка должна инициализироваться

Пример.

```
int i;
int& j = i;
j=7;
```

- **НО!**

  Хотя ссылки введены как обще средство языка **C++**, использовать их следует *только для объявления формальных аргументов функций*, науке не известен другой контекст полезного использования ссылок

Ссылки - аргументы функций

- Использование ссылки в качестве формального аргумента позволяет изменить значение фактического параметра
- При вызове функции для фактического аргумента сохраняется внешний вид обращения как при передачи значения

  Пример.

```
void set_to_five( int& i ) { i = 5; }
void main( int, char** ) {
  int  number = 78;
  set_to_five( number ) ;  // number теперь равно 5
}
```

- Для больших объектов данных вызов по ссылке быстрее, чем вызов по значению

Аргументы-ссылки

```
const int max_string_length = 128;
class String {
    public:
      String operator= ( char* );
      int  length( );
      int  read( );  // чтение из stdin
      void  print( );     // вывод в  stdout
      String substring (int start, int len);
      // возвращает подстроку не изменяя оригинала
      friend String operator+ ( String& , String& );
      friend String operator+ ( String&, char* );
      friend String operator+ ( char*, String& );
      // эффективность как при использовании указателей
    private:
      // объект типа String – это последовательность из не более чем
      // max_string_length символов, оканчивающаяся  символом NULL
      char text [ max_string_length+1 ];
};
```

Функция **operator+(** *String&, String&* **)**

```
#include "mystring.h"
#include <string.h>  // для стандартных строковых функций
#include <stdio.h>
#include <stdlib.h>  // для exit()
// все изменения – в заголовке функции
String operator+( String& s1, String& s2 ) {
    if( s1.length()+s2.length() > max_string_length ) {
      fprintf( stderr, "string too large: \"%s %s\".\n",
      s1.text, s2.text );
      exit(1);
    }
    String both;
    strcpy( both.text, s1.text );
    strcat( both.text, s2.text );
    return both;
}
```

Функция **operator+(** *String&* **, char*** **)**

```
#include "mystring.h"
#include <string.h>  // для стандартных строковых функций
#include <stdio.h>
#include <stdlib.h>  // для exit()
// все изменения – в заголовке функции

String operator+( String& s1 , char* s2 )  {
    if( s1.length()+strlen(s2) > max_string_length ) {
     fprintf( stderr, "string too large: \"%s %s\".\n",
     s1.text, s2 );
     exit(1);
    }
    String both;
    strcpy( both.text, s1.text );
    strcat( both.text, s2 );
    return both;
}
```

Функция **operator+(char* , *String&*)**

```
#include "mystring.h"
#include <string.h>  // для стандартных строковых функций
#include <stdio.h>
#include <stdlib.h>  // для exit()
// все изменения – в заголовке функции

String operator+( char* s1 , String& s2 )  {
    if( strlen(s1) + s2.length() > max_string_length ) {
      fprintf( stderr, "string too large: \"%s %s\".\n",
      s1 , s2.text );
      exit(1);
    }
    String both;
    strcpy( both.text, s1);
    strcat( both.text, s2.text );
    return both;
}
```

Использование операций класса

```
#include "mystring.h"

void main( int, char** ) {
    String  first, last, full;

    first = "Bilbo";       last = "Baggins";

    // оба присваивания вызывают метод   operator=(char*)
    full = first + "  " + last;

    // сперва вызывается друг  operator+(String& , char*)
    // а затем друг   operator+(String& , String& )
    ( "Name is: " + full ).print( );

    // вызывается друг operator+( char* , String& )
    // а затем метод print( ) для анонимного объекта
}
```

Ссылки – возвращаемые значения функций

- Возвращаемое значение не копируется $\Rightarrow$ нельзя возвращать ссылку на локальный объект.
- Для больших объектов возврат ссылки быстрее, чем возврат значения
- Т.к. ссылка – ***lvalue***, то функция – средство изменения значения.

Пример.

```
int& max( int& i, int& j ) { if( i > j ) return i; else return j; }
void main( int, char** ) {
    int  x = 42, y = 7500, z;
    z= max(x,y);  // z теперь равно 7500
    if( max(z,1) = 1 )  z++ ;
    else z-- ;
}
```

Возвращение ссылки

```
const int max_string_length = 128;
class String {
    public:
      String& operator= ( char* ); // теперь эффективно
      int  length( );
      int  read( );  // чтение из stdin
      void  print( );     // вывод в  stdout
      String substring (int start, int len);
      // возвращает подстроку не изменяя оригинала
      friend String operator+ ( String& , String& );
      friend String operator+ ( String&, char* );
      friend String operator+ ( char*, String& );
      // эффективность как при использовании указателей
    private:
      // объект типа String – это последовательность из не более чем
      // max_string_length символов, оканчивающаяся  символом NULL
      char text [ max_string_length+1 ];
};
```

Определение функции **operator=**

```
#include "mystring.h"
#include <string.h>
  // использование:  s = "Hello World"
String& String::operator=( char* s ) {
    strncpy( text, s, max_string_length );
    text [max_string_length] = '\0';
    return *this;
 }
```

Операция индексации

Пусть

String s;

char *c;*

s = “hello”;

- Вариант **char operator[ ](int)**
  - можно *c = s[0];*
  - нельзя *s[0] = ‘H’;*
- Вариант **char& operator[ ](int)**
  - можно *c = s[0];*
  - можно *s[0] = ‘H’;*

Добавление в *String*

```
const int max_string_length = 128;
class String {
    public:
      String& operator= ( char* );
      int  length( );
      int  read( );  // чтение из stdin
      void  print( );     // вывод в  stdout
      char& operator[ ] ( int );
      String substring (int start, int len);
      // возвращает подстроку не изменяя оригинала
      friend String operator+ ( String& , String& );
      friend String operator+ ( String&, char* );
      friend String operator+ ( char*, String& );
    private:
      // объект типа String – это последовательность из не более чем
      // max_string_length символов, оканчивающаяся  символом NULL
      char text [ max_string_length+1 ];
};
```

Определение **operator[ ]**

```
#include "mystring.h"
#include <stdio.h>
#include <stdlib.h>
  // использование  s[k]  эквивалентно
  // вызову s.operator[ ]( k )
char& String::operator[ ]( int i ) {
    if( i< 0 || i >= length() ) {
     fprintf ( stderr, "Ошибка индекса (%d) в
          строке \"%s\". \n", i, text );
     exit(1);
    }
    return text[i];
}
```

Использование индексации

```
#include "mystring.h"
#include <ctype.h>
#include <stdio.h>
void main( int, char** ) {
    String  s;
    printf( "Введите строку :" );
    s.read();
    for( int i = 0 ; i < s.length() ; i++ )  s[i] = toupper( s[i] );
    s.print();
}
```

РЕЗЮМЕ

- Ссылки используются в функциях как аргументы и/или результаты
- Вызов аргумента по ссылке
  - выглядит как вызов по значению
  - обеспечивает эффективность
  - допускает изменение аргумента
- Возвращение результата по ссылке
  - возможно для нелокальных данных
  - обеспечивает эффективность
  - превращает вызов функции в ***lvalue***

Упражнение

- Изменить в классе *String* методы операций “**==**” и “**!=**” так, чтобы они использовали вызов по ссылке. Прототипы функций имеют вид:

  int opetor!=(*String*&);

  int opetor==(*String*&);

- Перекомпилируйте тестовую программу из предыдущего упражнения.
- Надо ли менять код тестовой программы?