

Тема: Перегрузка операций

- Оформление перегрузки операций
- Правила перегрузки операций
- Выбор типа функции-операции
(функция-член или функция-друг)
- Модификация класса ***String***

Правила перегрузки операций

- Для перегрузки операции ***op*** надо определить функцию с именем ***operatorop***.
- При перегрузке операции нельзя изменить ее приоритет, ассоциативность и количество связанных ею операндов
- Для вызова используется символ операции:

Пример. Пусть для типа ***String*** функция ***concat*** заменена перегруженной операцией “+”. Тогда конкатенация строк ***s1*** и ***s2*** будет записываться как ***s1+s2***

Некоторые особенности

- Можно перегружать следующие операции:
+ - * / % ^ & | ~ ! = < > += -= *= /= %= ^=
&= |= << >> >>= <<= == != <= >= && || ++
-- ->*, -> [] () new new[] delete delete[]
- Следующие операции не могут быть перегружены : “ :: ” , “ . ” , “ .* ” , “ ?: ”
- Для перегруженных операций “ ++ ” и “ -- ” пре- и постфиксная формы неразличимы
- Операции “ = ” , “ [] ” , “ () ” , “ -> ” должны быть методами

Использование перегруженных операций

- Операнды передаются в функцию перегруженной операции
 - как вызывающий объект и аргумент (если операция является функцией-членом класса)
 $s1+s2$ эквивалентно $s1.operator+(s2)$
 - как аргументы (если операция не является функцией-членом класса)
 $s1+s2$ эквивалентно $operator+(s1, s2)$
- Возвращаемое значение функции – значение выражения

Перегрузка операций

```
const int max_string_length = 128;
class String {
    public:
        String operator= ( char* ); // вместо set_to()
        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 );
    private:
        // объект типа String – это последовательность из не более чем
        // max_string_length символов, оканчивающаяся символом NULL
        char text [ max_string_length+1 ];
};
```

Обратите внимание на:

- Тип операндов для операции “ + ”:
 - ***a+b***, а не ***&a+&b***
 - хотя бы один аргумент перегруженной операции должен иметь тип класса
- Неэффективность передачи операндов при вызове операции “ + ”: по значению, а не через указатель
- Присваивание переменных разных и одинаковых типов
 - String firstname, name;***
 - firstname = “Bilbo”;*** // использует `operator=(char*)`
 - name = firstname;*** // использует умолчание
- Возвращаемое значение перегруженной операции присваивания определяется выбором синтаксиса
 - a=b; if((a=b) == c) {...}***

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;  
}
```

Функция `operator+(String, String)`

```
#include "mystring.h"
#include <string.h> // для стандартных строковых функций
#include <stdio.h>
#include <stdlib.h> // для exit()

// можно реализовать в виде метода
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()

// можно реализовать в виде метода

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()

// нельзя реализовать в виде метода

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
}
```

РЕЗЮМЕ

В C++ можно определять функции-операции для разрабатываемых типов данных

- Позволяет писать изящный простой и понятный код
- Нельзя изменить приоритет, число операндов или ассоциативность определяемой операции
- Нельзя придумать новые операции
- Нельзя переопределить операции встроенных типов
- Хотя бы один аргумент должен быть типа класса
- Не всегда реализуемы в виде методов

Упражнение

- Изменить в классе *String* методы `is_the_same_as` и `is_different_from` на операции “`==`” и “`!=`”. Прототипы функций имеют вид:

```
int operator!=( String );
```

```
int operator==( String );
```

- Протестируйте следующий синтаксис:

```

#include "mystring.h"
#include <stdio.h>
void main(int, char *[]) {
    String h1, h2, w;
    h1 = "hello";
    h2 = "hello";
    w = "world";

    printf("\n String == String ");
    if ( h1 == h2 && !( h1 == w ) )
        printf("works.\n");
    else
        printf("doesn't work.\n");

    printf("\n String != String ");
    if ( !(h1 != h2) && h1 != w )
        printf("works.\n");
    else
        printf("doesn't work.\n");

    printf("\n String == char_* ");
    if( h1=="hello" && !( h1=="world") )
        printf("works.\n");
}

```

```

    else
        printf("doesn't work.\n");

printf("\n String != char_*");
    if ( !(h1 != "hello") && h1 != "world" )
        printf("works.\n");
    else
        printf("doesn't work.\n");

printf("\n char_* == String ");
    if ( "hello" == h1 && !( "world" == h1) )
        printf("works.\n");
    else
        printf("doesn't work.\n");

printf("\n char_* != String ");
    if ( !("hello" != h1) && "world" != h1 )
        printf("works.\n");
    else
        printf("doesn't work.\n");
}

```