ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

ПЕРЕГРУЗКА ОПЕРАЦИЙ

НЕЛЬЗЯ перегружать

* ?: :: # ## sizeof

ПРАВИЛА перегрузки

- при перегрузке операций сохраняются количество аргументов, приоритеты операций и правила ассоциации (справа налево или слева направо), используемые в стандартных типах данных;
- для стандартных типов данных переопределять операции нельзя;
- функции-операции не могут иметь аргументов по умолчанию;
- функции-операции наследуются (за исключением =);
- функции-операции не могут определяться как static.

тип operator операция (список параметров)
{тело функции}

Перегрузка унарных операций

```
class monstr
{
    …
    monstr & operator ++() {++health; return
   *this;}
};
monstr Vasia;
cout « (++Vasia).get_health();
```

Перегрузка унарных операций

```
class monstr
{
    …
    friend monstr & operator ++( monstr &M);
};
monstr& operator ++(monstr &M)
    {++M.health; return M;}
```

Перегрузка унарных операций

```
void change_health(int he){health = he;}

monstr& operator ++(monstr &M)
{
    int h = M.get_health(); h++;
    M.change_health(h);
    return M;
}
```

Перегрузка унарных операций

class monstrj

- monstr operator ++(int){
- monstr MCnhis); health++;
- return M;
- }
- }:
- monstr Vasia:
- cout « (Vas1a++).get_health();

Перегрузка операций **new** и **delete**

- им не требуется передавать параметр типа класса;
- первым параметром функциям new и new[] должен передаваться размер объекта типа size_t;
- они должны определяться с типом возвращаемого значения void*, даже если return возвращает указатель на другие типы;
- Операция delete должна иметь тип возврата void и первый аргумент типа void*;
- операции выделения и освобождения памяти являются статическими элементами класса.

Перегрузка операций **new** и **delete**

```
class Obj {...};
class pObj
{
    …
    private:
        Obj *p;
};
new
     Obj *p = new pObj;
sizeof(pObj)
```

Перегрузка операций **new** и **delete**

```
class pObj
{
    public:
        static void * operator new(size_t size);
    private:
        union
        {
            Obj *p;          // Указатель на объект
            pObj *next;  // Указатель на следующую свободную ячейку
        };
    static const int BLOCK_SIZE; // Размер блока
    static pObj *headOfFree;  // Заголовок списка свободных ячеек:
};
```

Перегрузка операций **new** и **delete**

```
void * pObj::operator new(size_t size)
{
    // Перенаправить запросы неверного количества памяти
    // стандартной операции new:
    if (size != sizeof(pObj)) return ::operator new(size);
    pObj *p = headOfFree; // Указатель на первую свободную ячейку
    // Переместить указатель списка свободных ячеек:
    if (p) headOfFree = p -> next;
```

Перегрузка операций **new** и **delete**

```
    // Если свободной памяти нет. выделяем очередной блок:
    else
    {
       pObj *newblock = static_cast<pObj*>
          (::operator new(BLOCK_SIZE *       sizeof(pObj)));
    // Вее ячейки свободны, кроме первой (она будет занята), связываем
   их:
    for (int i = 1: i< BLOCKJIZE - 1; ++i)
          newblock[i].next = &newblock[i + 1 ] ;
    newblock[BLOCK_SIZE - 1].next = 0:
    // Устанавливаем начало списка свободных ячеек:
    headOfFree = &newblock[1];
    p = newblock;
    }
return p;  // Возвращаем указатель на выделенную память
}
```

Перегрузка операций **new** и **delete**

```
pObj *pObj::headOfFree; // Устанавливается в 0 по умолчанию
const int pObj::BLOCK_SIZE = 1024;
```

Перегрузка операции приведения типа

Operator имя нового типа ();

Пример:

```
monstr::operator int(){return health;}
…
monstr Vasia; cout « int(Vasia);
```

Перегрузка операции вызова функции

```
class if_greater
{
    public:
    int operator () (int a, int b) const
     {
        return a > b;
    }
};
```

Перегрузка операции вызова функции

Пример:

```
if_greater x;
cout « x(1. 5) « endl;              // Результат - 0
cout « if_greater()(5. 1) « endl;   // Результат -
   1
```

Перегрузка операции индексирования []

```
#1nclucle <iostream.h>
#1nclude <stdlib.h>
class Vect
{
    public:
        explicit Vect(int n = 10);
        Vect(const int a[]. int n); //инициализация массивом
        ~Vect() { delete [] p; }
        int& operator [] (int i);
        void Print();
    …
    private:
        int* p;
        int size;
};
```

Перегрузка операции индексирования []

```
Vect::Vect(int n) : size(n)
{
    p = new int[size];
}
Vect::Vect(const int a[]. int n) : size(n)
{
    p = new int[size];
    for (int i = 0; i < size; i++) p[i] = a[i];
}
```

Перегрузка операции индексирования []

```
// Перегрузка операции индексирования:
int& Vect::operator [] (int i)
{
    if(i < 0 II i >= size)
    {
        cout « "Неверный индекс (i = " « i « ")" « endl;
        cout « "Завершение программы" « endl;
        exit(0);
    }
    return p[i];
}
```

Перегрузка операции индексирования []

```
void Vect::Print()
{
	for (int i = 0; i < size; i++) cout « p[i] « " ";
	cout « endl;
}
int main()
{
	int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
	Vect  a(arr, 10);
	a.Print();
	cout « a[5] « end!;
	cout « a[12] « endl;
	return 0;
}
```

Перегрузка операции индексирования []

Результат работы программы:

1 2 3 4 5 6 7 8 9 10

6

Неверный индекс (i = 12)

Завершение программы

Указатели на элементы классов

.* и ->*

Формат указателя на метод класса:

возвр тип (имя класса::*имя указателя)(параметры);

Например:

```
int get_health() {return health;}
int get_ammo() {return ammo;}
int (monstr:: *pget)();

void fun(int (monstr:: *pget)())
{
    (*this.*pget)(); // Вызов функции через операцию .*
    (this->*pget)(); // Вызов функции через операцию ->*
}
```

Указатели на элементы классов

Пример:

```
// Присваивание значения указателю:
pget « & monstr::get_health;
monstr Vasia. *p;
p = new monstr;
// Вызов функции через операцию .* :
int Vasin_health = (Vasia.*pget)();
// Вызов функции через операцию ->* :
int p_health = (p->*pget)();
```

Указатели на элементы классов

Правила использования указателей на методы классов:

- Указателю на метод можно присваивать только адрес методов, имеющих соответствующий заголовок.
- Нельзя определить указатель на статический метод класса.
- Нельзя преобразовать указатель на метод в указатель на обычную функцию, не являющуюся элементом класса.

Указатели на элементы классов

Формат указателя на поле класса:

Тип данных (имя класса::*имя указателя);

&имя_класса: :имя_поля;// Поле должно быть public

int (monstr::*phealth) » &monstr::health;

cout « Vasia.*phealth; // Обращение через операцию .*

cout « p->*phealth; // Обращение через операцию ->*

Рекомендации по составу класса

Класс должен содержать:

- скрытые поля (private)
- функции:
  - конструкторы, определяющие, как инициализируются объекты класса;
  - набор методов, реализующих свойства класса;
  - набор операций, позволяющих копировать, присваивать, сравнивать объекты и производить с ними другие действия, требующиеся по сути класса;
  - Класс исключений, используемый для сообщений об ошибках с помощью генерации исключительных ситуаций.