ПРЕДЕФИНИРАНЕ НА ОПЕРАЦИИ

1. *Псевдоними*
2. *Указател this*
3. *Основни ограничения при предефиниране на операции*
4. *Дефиниране на нови операции*
5. *Примери за предефиниране на операции*

1. Псевдоними (алтернативни имена)

- Един и същ обект (променлива) може да се означава с няколко имена.
- Тази възможност се използва при предаване параметри на функции.
- Псевдонимите се използват в случаите, когато е необходимо да се избегне копирането на стойностите на аргументите или резултатите на функциите.

1.Псевдоними (алтернативни имена)

- Ако един псевдоним и неговият инициализатор са от един и същ тип, те представляват две имена на един и същ обект.
- Това означава, че всички операции, които се прилагат върху променливата, ще се отнасят до псевдонима и обратно.

1.1. Псевдоним като параметър на функция

```
void swap ( int &x, int &y )
{
    int buf;
    buf = x;
    x = y;
    y = buf;
}
```


1.1. Псевдоним като параметър на функция

- Когато се променя стойността на псевдоним, то в действителност се променя стойността на самата променлива;
- Ако псевдонимът е параметър на функция, промяна се стойността на променливата, с която се извиква функцията;
- При предаване на обект чрез псевдоним, той не се копира и отпада необходимостта от неговото конструиране и разрушаване;
- Псевдоним като параметър на функция се използва за модифицируеми параметри.

1.1. Псевдоним като параметър на функция

```
class Point
{
    public:
        Point( void );
        Point( float, float );
        ~Point( void );
        void print( void );
        void neg( Point & );
    private:
        float x;
        float y;
};
```

1.2. Константен псевдоним като параметър на функция

```
class Point
{
    public:
        Point( void );
        Point( float, float );
        ~Point( void );
        void print( void );
        float dist( Point, Point );
    private:
        float x;
        float y;
};
```


1.2. Константен псевдоним като параметър на функция

- За да се избегне копирането на обекта и извикването на деструктора при прекъсване изпълнението на функцията, е необходимо обектите да се предадат чрез псевдоними.
- Тъй като обектите не трябва да бъдат променени от функцията, то се използват *константни псевдоними*.

1.2. Константен псевдоним като параметър на функция

```
class Point
{
    public:
        Point( void );
        Point( float, float );
        ~Point( void );
        void print( void );
        float dist( const Point &, const Point & );
    private:
        float x;
        float y;
};
```


1.3. Псевдоним като резултат на функция

- Функция може да върне псевдоним като резултат ;
- След като функцията прекъсне действието си, обектът, към който сочи псевдонимът, трябва все още да съществува;
- Функция, която връща псевдоним като резултат, може да се използва и от лявата страна на оператора за присвояване;

1.3. Псевдоним като резултат на функция

```
class Point
{
    public:
        Point( void );
        Point( float, float );
        ~Point( void );
        void print( void );
        float &X( void );
        float &Y( void );

    private:
        float x;
        float y;
};
```


2. Указател this

- Обектът става активен, когато се извика някой от неговите методи;
- Указателят ***this*** се инициализира автоматично да сочи към обекта, който е активен или, с други думи, указва към обекта, който го е създал;
- this е създаден, за да се осигури връзката между съответния обект и методите на класа.
- По този начин едно-единствено копие на методите на класа се използва от всички обекти на същия клас.

2. Указател this

```
class DList
{
   public:
        void append( DList * );
   private:
        DList *prev, *next;
};
```


3. Основни ограничения при предефиниране на операции

- **Под *обобщени операции* се разбират такива операции, които се изпълняват за обекта като цяло, а не за отделните му членове.**
- **Каква операция ще се изпълнява - стандартна или обобщена, компилаторът разпознава единствено от контекста.**
- **Всички символи за операции в C++ може да бъдат предефинирани с изключение на**

  ::, sizeof, ?:, . и .*

3. Основни ограничения при предефиниране на операции

- **Не е възможно да се променят основните характеристики на операторите.**
- **Някои от свойствата на оригиналните оператори могат да се загубят след тяхното предефиниране.**
- **Първоначалната семантика на операторите не може да се променя.**
- **За съкратените комбинирани операции, като +=, -=, *= и др. при предефиниране не се запазват зависимостите между съставящите ги операции.**

3. Основни ограничения при предефиниране на операции

- **Предефинирането на операторите се осъществява чрез *операторни функции*, които трябва да бъдат или методи на класовете, или да имат поне един параметър от тип class.**
- **Един и същ оператор може да бъде предефиниран многократно чрез множество операторни функции, които трябва да се различават по типа и (или) броя на параметрите си.**

4. Дефиниране на нови операции

- **Предефинирането на операторите се осъществява чрез специален вид функции, наречени *операторни функции*;**
- **Параметри на функцията са операндите на операцията, а резултатът от нея би могъл да се връща като върната стойност;**
- **Операция, декларирана като член на класа, има един аргумент по-малко от съответната "приятелска" декларация, тъй като обектът, за който се извиква, се подразбира като първи аргумент на операцията-член.**

4. Дефиниране на нови операции

```
class Beta
{
    public:
         Beta& operator+( const Beta & );
         Beta& operator-( const Beta &);
    private:
         ....................
};
```

4.1. Формат на бинарна операция

Като метод на класа

Записът **X @ Y** ще се интерпретира като

```
X.operator@(Y)
```

- Всеки метод на клас има за първи неявен параметър указателя *this* към обекта, който активира метода и представлява първия операнд.
- Вторият операнд се предава като параметър на операторната функция.
- Връщаният резултат на операторната функция може да бъде обект от типа на обектите операнди и там да се съхранява резултатът от операцията.

4.1. Формат на бинарна операция

Като метод на класа

```
class A
{
    public:
        A operator-( const A & ); // x-y
        A operator*( const A & ); // x*y
};
```


4.1. Формат на бинарна операция

Като приятелска функция

Записът **X @ Y** ще се интерпретира като

```
operator@(X,Y)
```

- Параметрите на функцията са обектите-операнди на операцията.
- Операторната функция е приятелска на класа на операндите и резултатът от операцията може директно да се запише в друг обект от същия клас.

4.1. Формат на бинарна операция

Като приятелска функция

```
class B
{
    friend B operator-( const B &, const B & );
    friend B operator*( const B &, const B & );

     public:
        ……..
};
```


4.2. Формат на унарна операция

Като метод на класа

Записът **@X** ще се интерпретира като

```
X.operator@()
```

```
class A
{
 public:
     A operator-( void );// -x
     ……..
};
```


4.2. Формат на унарна операция

Като приятелска функция

Записът **@X** ще се интерпретира като

```
operator@(X)
```

```
class B
{
     friend B operator-( const B & );
     public:
     ……..
};
```

4. Дефиниране на нови операции

Правило:

Всяка операция, която е член на класа, има един аргумент по-малко от съответната "приятелска" операция. Това е така, защото първият операнд на операцията-член е обекта, с който тя се извиква.

5. Примери за предефиниране на операции

```
const int ARRAY_SIZE = 24;
class IntArray
{
    public:
        IntArray( int sz = ARRAY_SIZE );
        IntArray( const IntArray& );
        ~IntArray()     { delete ia; }
        void loadArray( void );
        void showArray( void );
        int getSize( void )   { return size; }
        ..............................
    private:
        int size;
        int *ia;
};
```


5. Примери за предефиниране на операции

```
class IntArray
{
   public:
      ..............................
      IntArray& operator = ( const IntArray & );
      int& operator [] ( int );
      IntArray& operator ++ ( void );
      IntArray operator + ( const IntArray & );
      void operator += ( int );
      IntArray& operator += ( int );
      ……..
   private:
      ..............................
};
```


5.1. Дефиниции на конструкторите на класа

```
IntArray :: IntArray(const IntArray &oldIa)
{
     size = oldIa.size;
     ia = new int[size];
     for ( int i=0; i<size; ++i)
                  ia[i] = oldIa.ia[i];
}
```


5.2. Дефиниции на инструменталните методи на класа

```
void IntArray :: showArray( void )
{
  cout << "Разпечатване на масива\n";
  for ( int i=0; i<size; ++i)
    cout << "ia[" << i << "] = " << ia[i] << endl;
}
```


5.3. Предефиниране на операции += И -=

Като метод на класа

```
class IntArray
{
 public:
     // операция += (увеличаване с константа)
     void operator += ( int );

     // операция -= (намаляване с константа)
     void operator -= ( int );
     ……..
};
```

5.3. Предефиниране на операции += И -=

Като метод на класа

```
IntArray myArray;
myArray.loadArray();
myArray.showArray();
myArray += 5;
myArray.showArray();
myArray -= 10;
myArray.showArray();
```


5.3. Предефиниране на операции += И -=

Като метод на класа

```
class IntArray
{
 public:
    void operator += ( IntArray & );     // +=
    void operator -= ( IntArray & );     // -=
    ……..
};
```


5.3. Предефиниране на операции += И -=

Като приятелска функция

```
class IntArray
{
    friend void operator += ( IntArray&, int);
    public:
        ……..
    private:
        ……..
};
```


5.3. Предефиниране на операции += И -=

Като приятелска функция

```
void operator += ( IntArray &inAr, int a )
{
    for (int i=0; i< inAr.size; i++)
                inAr.ia[i] = inAr.ia[i] + a;
}
```


5.4. Предефиниране на операции + И -

Като метод на класа

```
class IntArray
{
 public:
    // операция + (събиране на два масива)
    IntArray operator + ( const IntArray & );

    // операция - (изваждане на масиви)
    IntArray operator - ( const IntArray & );
    ……..
};
```


5.4. Предефиниране на операции + И -

Като метод на класа

```
IntArray Obj1;
IntArray Obj2( Obj1.getSize() );
IntArray Obj3( Obj1.getSize() );
Obj3 = Obj2 + Obj1;
Obj3.showArray();
```

5.4. Предефиниране на операции + И -

Като приятелска функция

```
class IntArray
{
     friend IntArray operator+
          ( const IntArray &, const IntArray & );
     public:
          ……….
     private:
          ……….
};
```


5.4. Предефиниране на операции + И -

Като приятелска функция

```
IntArray operator + ( const IntArray &A, const IntArray &B )
{
     if ( A.size != B.size){
          cout << "Различни размери\n"; getch(); exit(0);
     }
     IntArray newAr(A.size); // създаване на нов обект
     for (int i=0; i<A.size; i++)
          newAr.ia[i] = A.ia[i] + B.ia[i];
     return newAr;
}
```


5.5. Предефиниране на операции ++ И --

Като метод на класа

```
class IntArray
{
    public:
        IntArray &operator ++ ( void ); // операция ++

        IntArray &operator -- ( void ); // операция --
        ……..
};
```


5.5. Предефиниране на операции ++ И --

Като метод на класа

```
IntArray Obj;
……..
Obj++;
Obj.showArray();
```


5.5. Предефиниране на операции ++ И --

Като приятелска функция

```
class IntArray
{
     friend IntArray operator ++ ( IntArray & );
     public:
          ……….
     private:
          ……….
};
```


5.5. Предефиниране на операции ++ И --

Като приятелска функция

```
IntArray operator ++ ( IntArray &oldArray )
{
     for (int i=0; i<oldArray.size; i++ )
               oldArray.ia[i]++;
     return oldArray;
}
```

5.6. Предефиниране на операция индексиране [ ]

Задължително като метод на класа

```
class IntArray
{
 public:
    // операция []
    int &operator [] ( int );
    ……..
};
```


5.7. *Предефиниране на операция присвояване =*

Основните разлики между присвояване и инициализиране са следните:

- **инициализирането може да бъде по-общо отколкото поелементното присвояване;**
- **за присвояването на обекти и за инициализирането на обекти чрез присвояване се използват по подразбиране различни служебни функции.**

5.7. Предефиниране на операция присвояване =

В редица случаи стандартната операция = просто няма да работи така, както се очаква.

```
void fun( void )
{
        IntArray obj1(20);
        IntArray obj2(20);
        obj1 = obj2;
}
```

Служебната функция-операция

```
IntArray& IntArray :: operator= ( const IntArray & );
```


5.7. Предефиниране на операция присвояване =

- **При повторно дефинирана от програмиста операция се очаква, че нейните операнди са инициализирани.**
- **Затова е необходимо наличието на дефиниран от програмиста копиращ конструктор:**

```
IntArray :: IntArray( const IntArray & )
```


5.7. Предефиниране на операция присвояване =

```
IntArray& IntArray :: operator = ( const IntArray &oldAr )
{
   delete ia;            // освобождава заетата от ia памет
   size = oldAr.size;    // изравнява размерите
   ia = new int[size];   // заделя нова област памет

   // копира
   for ( i=0; i<size; i++ ) ia[i]=oldAr.ia[i];
   return *this;
}
```