ПРЕДЕФИНИРАНЕ НА ОПЕРАЦИИ

1. Цел на упражнението

Усвояване синтактичните и семантични особености при предефиниране на стандартните операции с класове и тяхното използване.

2. Основни ограничения при предефиниране на операции

От системна гледна точка класовете са много по-подходящи за моделиране на реалния процес. Чрез класовете се постига сближаване на нивото, на което се проектира системата, с това, на което тя се реализира програмно. Това означава постигане на пряко съответствие между концепциите и реализацията им, което в крайна сметка води до повишаване ефективността на самата система, както и на процесите на разработването и поддържането й. Това налага необходимостта между представителите на класовете - обектите, да може да се изпълняват обобщени операции. Под ***обобщени операции*** се разбират такива операции, които се изпълняват за обекта като цяло, а не за отделните му членове.

В C++ се допуска символите за операциите, вградени в езика, да се използват повторно при дефиниране на нови операции. Този процес се нарича ***предефиниране на операциите***. Каква операция ще се изпълнява - стандартна или обобщена, компилаторът разпознава единствено от контекста.

Всички символи за операции в C++ могат да бъдат предефинирани с изключение на ::, sizeof, ?:, . и .*.

При използване на символите на операциите при предефиниране са установени следните ограничения, които произтичат от основното предназначение на символите:

- Не е възможно да се променят основните характеристики на операторите. Такива характеристики са техният приоритет, броят на операндите и правилата за асоциативност. *Следователно, операторите* **++** *и* **–** *са винаги унарни (едноопérандни), докато операторите* + - & * *могат да бъдат предефинирани като унарни и като бинарни (двуоперандни).*
- 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. Например **+=** няма да произтича от операция **+** и последващо присвояване. Как ще действа новата операция **+=** зависи от това, какво е дефинирал програмистът.
- 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.

3. Дефиниране на нови операции

Предефинирането на операторите се осъществява чрез специален вид функции, наречени ***операторни функции***. Името на една операторна функция се състои от ключовата дума ***operator*** и мнемоничното означение на оператора, който се дефинира чрез нея. Най-добре е операторните функции да бъдат реализирани като методи на съответния клас. 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.

Синтаксисът на дефиницията на операторна функция е следния:

```
<тип_резултат> <име_клас>::operator<операция>(<параметри>)
        { <тяло_на_функцията> }
```

3.1. Формат на бинарна (двуоперандна) операция

```
<операнд1> @ <операнд2>
```

Предполага се, че операндите са обекти от един и същ клас.

Първата реализация на операцията може да бъде като метод на обекта **<операнд1>** с един явен параметър – обекта - **<операнд2>**:

```
<операнд1>.<операция> (<операнд2>)
    ↓           ↓           ↓
  Обект      МетодОбект
```

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. Така в операторната функция се получава директен достъп до двата обекта. Връщаният резултат на операторната функция може да бъде обект от типа на обектите операнди и там да се съхранява резултатът от операцията.

3.2. Формат на унарна (еднооперандна) операция

За унарна операция разсъжденията са аналогични. Тя може да се реализира като метод без параметри. Отново се използва указателят ***this***.

```
<операнд>.<операция> ()
    ↓           ↓
  Обект       Метод
```

4. Задачи за изпълнение

Задача 1:

Декларацията и дефинициите на методите на класа **Box** е представена във файла `H:\SKELET\EX5_1.CPP`.

Да се създаде проект с име `EX51`.

Файлът `EX5_1.CPP` да се добави към създадения проект.

Проектът да се компилира и изпълни.

Да се анализират получените резултати.

Задача 2:

За класа **Box** от **Задача 1** да се предефинират оператори за свиване размерите на правоъгълника с определена константа и определен брой пъти.

Във функцията **main** да се добавят оператори за тестване на предефинираните операции.

Задача 3:

Декларацията и дефинициите на методите на класа **NewBox** е представена във файла `H:\SKELET\EX5_2.CPP`.

Да се създаде проект с име `EX52`.

Файлът `EX5_2.CPP` да се добави към създадения проект.

Проектът да се компилира и изпълни.

Да се анализират получените резултати.

Задача 4:

Да се преобразува структурата на класа **NewBox** от **Задача 3** в двусвързан списък. Да се преобразуват по съответен начин методите на класа. Да се дефинират методи за добавяне и изтриване на елемент. Да се използва указател **this**. Функцията `main` да се преработи, като се предвидят подходящи оператори за тестване на създадените методи.

Задача 5:

Декларацията на клас **IntArray** е представена във файла `H:\SKELET\EX5_3.CPP`.

Да се създаде проект с име `EX53`.

Файлът `EX5_3.CPP` да се добави към създадения проект.

За класа **IntArray** да се дефинират функции-членове за предефиниране на следните операции: [] - индексиране; ++ - инкрементиране и + - събиране на два масива.

Направените дефиниции да се тестват за различни обекти.

Задача 6:

Декларацията на клас **IntList** е представена във файла H:\SKELET\EX5_4.CPP.

Да се създаде проект с име EX54.

Файлът EX5_4.CPP да се добави към създадения проект.

Указания: Дефинирането на **val** и **next** като членове на класа може да създаде неприятности при неговото използване:

```
class IntList {
      public:
            IntList( val = ??? );
      private:
            int val;
            IntList *next;
};
```

Тази декларация на **IntList** поражда нялколко проблема, причина за които е смесването на обект и член на класа. Например, при тази декларация на **IntList** не може да се създаде празен списък. Въпросителните знаци в конструктора подчертават този проблем. Не съществува стойност за **val**, с която да означим празен списък. Други проблеми се появяват при дефинирането на **insert()** и **remove()**. Причината е в двойнствената интерпретация на обектите от клас **IntList** – като цял списък и като негов първи елемент. За да ги разграничим можем да декларираме два класа – **IntList** и **IntItem**. **IntItem** е скрит клас. Само клас **IntList** може да създава и обработва обекти от клас **IntItem**

За класа **IntList** да се дефинират декларираните конструктори и следните функции-членове:

- **display** - за извеждане елементите на списъка;
- **insert** - за добавяне на нов елемент в началото на списъка;
- **append** - за добавяне на нов елемент в края на списъка;
- **remove** - за изтриване на целия списък.

Направените дефиниции да се тестват за различни обекти.