1

КОНСТРУКТОРИ И ДЕСТРУКТОРИ

1. Инициализиране на обекти
2. Дефиниране на конструктори и деструктори
3. Предефинирани конструктори
4. Подразбиращ се конструктор
5. Конструктори с подразбиращи се параметри
6. Конструктори за присвояване и копиране
7. Конструктори и масиви от обекти
8. Приятелски декларации

2

1. Инициализиране на обекти

- *Най-често инициализацията се свежда до задаване на начални стойности на данните на новосъздадените обекти.*
- *В общия случай обаче, инициализацията може да включва и други действия като заделяне на памет, запомняне на текущото състояние на програмата с цел възстановяване в друг момент, копиране на друг обект.*

3

1. Инициализиране на обекти

- *Синтактичната конструкция за предварителни действия се нарича конструктор,*
- *а тази за заключителните действия - деструктор.*

4

2. Дефиниране на конструктори и деструктори

- **Конструкторът и деструкторът са методи на съответния клас.**
- **Конструкторът е метод, който се извиква автоматично при дефинирането на обект след резервирането на памет за неговите компоненти.**
- ***Деструкторът се изпълнява автоматично непосредствено преди освобождаването на паметта, която е заета от обекта.***

5

2. Дефиниране на конструктори и деструктори

```
class Alfa
{
 public:
    Alfa( int, int);  //Декларация на конструктор
    void disp( void );
 private:
    int x;
    int y;
};
```


6

2. Дефиниране на конструктори и деструктори

```
class String
{
 public:
    String( char * );
    String( int );
 private:
    int len;
    char *str;
};
```

2. Дефиниране на конструктори и деструктори

7

Всяка дефиниция на обект поражда неявно обръщение към конструктор на класа, при което се прави пълен контрол на типовете.

```
int lenSt = 1024;

// грешка, няма аргументи
String myString;

 // грешка, несъответствие в типовете
String inBuf( &lenSt );

// грешка, повече аргументи
String search ("rosebud", 7);
```


2. Дефиниране на конструктори и деструктори

8

Съществува явен и съкратен начин за предаване на аргументи на конструктор:

явен

```
String searchWord = String("rose");
```

съкратен

```
String commonWord("the");
String inBuf = 1024;
```

използването на new изисква явно предаване

```
String *ptrBuf = new String (1024);
```


2. Дефиниране на конструктори и деструктори

9

```
class Person
{
   public:
      Person( void );
      void display( void );
      ~Person( void )
         { delete name; }
   private:
      char *name;
      int age;
};
```


3. Предефинирани конструктори

10

- Един и същ клас може да има няколко конструктора.
- Всички те трябва да имат едно и също име (името на класа), но трябва да се различават по брой и/или тип на параметрите си.
- Такива конструктори се наричат предефинирани (overloading constructors).
- При създаването на обекти се изпълнява само един от предефинираните конструктори в зависимост от броя и/или типа на предаваните фактически параметри.
- Предефинираните конструктори дават възможност различните обекти на един и същ клас да бъдат инициализирани по различен начин.

3. Предефинирани конструктори

11

```
class Person
{
   public:
      Person( void );      // Конструктор 1
      Person( char *, int ); // Конструктор 2
      void display( void );
      ~Person( void )
         { delete name; }
   private:
      char *name;
      int age;
};
```


4. Подразбиращ се конструктор

12

- Всеки клас има един подразбиращ се конструктор, който се създава автоматично, ако не са дефинирани други конструктори.
- time obj = anti;// obj и anti са обекти от клас time
- Тук следва да се разгледат два случая:
  - В първия случай, класът *time* няма конструктори. Присвояването означава побитово копиране на членовете на обекта *anti* в съответните членове-данни на обекта *obj*.
  - При втория случай, класът *time* има конструктори. В този случай няма да има обръщение към някой от тези конструктори, а отново ще се изпълни подразбиращият се копиращ конструктор.

13

4. Подразбиращ се конструктор

- Подразбиращите се конструктори на класовете могат да бъдат предефинирани.
- За да се предефинира подразбиращият се конструктор на един клас е необходимо да бъде дефиниран конструктор без параметри.

```
Person :: Person( void);
```


14

5. Конструктори с подразбиращи се параметри

```
class Time
{
     Time( int a, int b=5, int c=10 );
};
```

- Подразбиращите се параметри трябва да бъдат в края на списъка с параметрите на конструктора.
- При обръщение към конструктора подразбиращите се параметри могат да бъдат пропуснати и тогава те получават подразбиращите се стойности, например:

```
Time obj(3);   // b и c са пропуснати
Time obj(3,12);     // c е пропуснат
```


15

5. Конструктори с подразбиращи се параметри

Недопустимо е в един и същи клас да бъдат дефинирани подразбиращ се конструктор (конструктор без параметри) и конструктор с един подразбиращ се параметър.

16

5. Конструктори с подразбиращи се параметри

```
class Beta
{
   public:
     Beta( void )  // Подразбиращ се конструктор
        { r=0; }
     Beta( float x=0.5) // Констр. с 1 подразбиращ се пар.
        { r=x; }
   private:
        float r;
};

   void main( void )
   {
     Beta obj(2.5);   // Ще се изпълни вторият конструктор
     Beta obj2;    // ГРЕШКА!!
   }
```


17

6. Конструктори за присвояване и копиране

- Конструкторът за присвояване на даден клас е конструктор, който има един параметър от тип псевдоним на обект от същия клас.

```
Alfa :: Alfa( Alfa & )
```

- Ако в един клас не е дефиниран конструктор за присвояване, компилаторът автоматично създава такъв конструктор, при изпълнението на който компонентите на новосъздадения обект се инициализират със стойностите на компонентите на обекта, намиращ се от дясната страна на знака за присвояване.
- *Автоматично създаваният от компилатора конструктор за присвояване е конструктор за копиране*.

18

7. Конструктори и масиви от обекти

- Ако класът, към който принадлежат елементите-обекти на масива, има конструктор, то той задължително трябва да има конструктор без параметри.
- Статично дефиниран масив от обекти се инициализира по два начина:
  - ✓ неявно - чрез извикване на конструктора по подразбиране за всеки обект-елемент на масива;
  - ✓ явно - чрез инициализиращ списък в дефиницията на масива.
- При първия начин ако няма подразбиращ се конструктор, компилаторът ще сигнализира за грешка.

7. Конструктори и масиви от обекти

19

Използване на втория начин за инициализиране:

```
Array vector[20] = {<инициализиращ_списък> };

<инициализиращ_списък> = <стойност1>, <стойност2>, .........

<инициализиращ_списък> = <обръщение_към_конструктор>(......),

<инициализиращ_списък> = <обр._към_констр.>(......),  стойност1, ..
```

- Стойностите се разглеждат като параметър на конструктор.
- Следователно *в класа трябва да има дефиниран конструктор с един параметър или с един явен и останалите подразбиращи се параметри*. Този конструктор ще бъде извикан неявно.

7. Конструктори и масиви от обекти

20

```
class Array
{
   public:
      Array( void )
           { k=1; c='~';}
      Array( int i, char r = '%')
           { k=i; c=r; }
      void print( void )
      { cout << "k= " << k << "c= " << c << "\n"; }
   private:
      int k;
      char c;
};
```


8. Приятелски декларации

21

- **Скриването на информация понякога налага големи ограничения;**
- **Използването на приятелски декларации е начин за преодоляване на тези ограничения;**
- **Приятелите на даден клас имат достъп до всички негови компоненти (включително до private компонентите), т.е. всички компоненти на класа имат статут на public по отношение на неговите приятели.**

8. Приятелски декларации

22

```
class X
{
   friend int R::f1( X * );   // Метод-приятел на класа X
   friend void Z::f3( X & );// Метод-приятел на класа X
   friend float f( int, X & );// Функция-приятел на класа X
   friend class Y;    // Клас-приятел на класа X

     public:
        ...................
     private:
        int value;
};
```


8. Приятелски декларации

23

```
Screen& isEqual( Screen &, Window & );

class Screen
{
   friend Screen& isEqual( Screen &, Window & );
   // ............
};
class Window
{
   friend Screen& isEqual( Screen &, Window & );
   // ...........
};
```


8. Приятелски декларации

24

```
class Screen
{
   public:
      Screen& copy( Window &);
      // ............
};

class Window
{
   friend Screen& Screen :: copy (Window &);
   public:
      // ...................
};
```

8. Приятелски декларации

25

Един клас също може да се декларира като приятелски за друг клас:

```
class Screen
{
  friend class Window;
  public:
  ……..
  ……..
};
```

Така методите на Window имат достъп до всички членове на клас Screen.

8. Приятелски декларации

26

```
class Alfa
{
   friend void swap ( Alfa &, Beta &);   // Функция-приятел
     public:
     Alfa( void )
        { cout << "Value = ? "; cin >> value; }
     void disp ( void )
        { cout << "value = " << value; }

   private:
        int value;
};
```


8. Приятелски декларации

27

```
class Beta
{
   friend void swap ( Alfa &, Beta &); // Функция-приятел

   public:
      Beta( void )
      { cout << "X = "; cin >> x; cout << "Y="; cin >> y; }
      void disp ( void )
         { cout << "X = " << x << "Y = " << y; }

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