**КОНСТРУКТОРИ И ДЕСТРУКТУРИ. СЪЗДАВАНЕ И РАЗРУШАВАНЕ НА ОБЕКТИ НА КЛАСОВЕ**

Конструктори –има същото име като класа, към който принадлежи и не притежава тип на връщан резултат. Конструктора е необходим за инициализация при решаване на реални задачи. Конструкторът на един клас се извиква всеки път когато се създава обект от този клас.

Пример 1:

1. #include <iostream>
2. using namespace std;
3. class myclass {
4. int a; //private променлива
5. public:
6. myclass(); // декларира конструктор
7. void show(); // декларира функцията show()
8. };
9. myclass::myclass() //деф. конструктора –той няма тип на връщан резулт!
10. {
11. cout << "In constructor"<<endl; //тяло на конструктора
12. a = 10;
13. }
14. void myclass::show() //дефинира функцията show()
15. {
16. cout << a; //тяло на функцията show() отпечатва a на екрана
17. }
18. int main() //демонстрира класа
19. {
20. myclass ob; //конструктора се извиква при създаване на обекта ob
21. ob.show(); //функц. за отпечатване на а на екарана става чрез констр.
22. return 0;
23. }

Резултат: а която има стойност 10 се инициализира от конструктура

In constructor

10

За глобалните обекти, конструкторът се извиква веднъж – когато започне изпълнението на програмата , за локалните обекти-всеки път, когато се изпълнява конструкцията за деклариране на променлива.

**Деструктор**: противоположен на конструктора

Тази функция се извиква, когато се унищожава обект. Това е наложително, защото при създаването си обекта заделя памет, и тя трябва да се унищожи при освобождаването на обекта. Името на деструктура на един клас е името на класа предшествано от ~

Пример 2: Тази програма демонстрира деструктур

1. #include <iostream>
2. using namespace std;
3. class myclass {
4. int a;
5. public:
6. myclass(); // конструктор
7. ~myclass(); // деструктор
8. void show();
9. };
10. myclass::myclass()
11. {
12. cout << "In constructor\n";
13. a = 10;
14. }
15. myclass::~myclass() //дефинира деструктора
16. {
17. cout << "Destructing...\n"; //тялото на деструктора
18. }
19. void myclass::show()
20. {
21. cout << a << "\n";
22. }
23. int main()
24. {
25. myclass ob;
26. ob.show();
27. return 0;
28. }

Резултат:

In constructor

10

Деструктурът на един клас се извиква при унищожаването на обект от този клас. Локалните обекти се унищожават, когато излязат извън областта на видимост.

Глобалните обекти се унищожават когато завърши изпълнението на програмата.

Пример 3 . Тази програма използва обект от класа timer, за да определи времето между създаването на един обект от класа timer , и неговото унищожаване. Изминалото време се извежда, когато се извика деструктура на обекта. Можете да използвате подобен обект, за да измерите времето за изпълнение на една програма или времето , изразходвано от една функция в даден блок. Само се уверете, че обектът излиза от областта на видимост в момента, в който искате да приключи времевия интервал.

1. #include <iostream>
2. #include <ctime>
3. using namespace std;
4. class timer {
5. clock\_t start;
6. public:
7. timer(); // конструктур
8. ~timer(); //деструктур
9. };
10. timer::timer()
11. {
12. start = clock();
13. }
14. timer::~timer()
15. {
16. clock\_t end;
17. end = clock();
18. cout << "Elapsed time: " << (end-start) / CLOCKS\_PER\_SEC << endl;
19. }
20. int main() //демонстрира констр. и деструкт.
21. {
22. timer ob;
23. char c;
24. cout << "Press a key followed by ENTER: ";
25. cin >> c;
26. return 0;
27. }

Тази програма използва стандартната библиотечна функция clock(), която връща броя на тактовите импулси от началото на изпълнението на програмата. Разделяйки тази стойност на CLOCKS\_PER\_SEC, получавате резултата в секунди

Пример 4. Създайте клас stopwatch, който емулира хронометър. Използвайте конструктор, за да установите началната стойност 0 за измереното време. Добавете член-функцията start() и stop(), които съответно да стартират и да спират таймера. Включете и член функцията show(), която да показва измереното време. Също така използвайте и деструктора, който автоматично да показва измереното време при унищожаване на обекта от тип stopwatch ( за по-лесно извеждайте времето в секунси).

// Stopwatch емулатор

1. #include <iostream>
2. #include <ctime>
3. using namespace std;
4. class stopwatch {
5. double begin, end;
6. public:
7. stopwatch(); //конструктор
8. ~stopwatch(); //деструктор
9. void start();
10. void stop();
11. void show();
12. };
13. stopwatch::stopwatch()
14. {
15. begin = end = 0.0;
16. }
17. stopwatch::~stopwatch()
18. {
19. cout << "Stopwatch object being destroyed...";
20. show();
21. }
22. void stopwatch::start()
23. {
24. begin = (double) clock() / CLOCKS\_PER\_SEC;
25. }
26. void stopwatch::stop()
27. {
28. end = (double) clock() / CLOCKS\_PER\_SEC;
29. }
30. void stopwatch::show()
31. {
32. cout << "Elapsed time: " << end - begin;
33. cout << endl;
34. }
35. int main() //демонстрира
36. {
37. stopwatch watch;
38. long i;
39. watch.start();
40. for(i=0; i<320000; i++) ; // time a for loop
41. watch.stop();
42. watch.show();
43. return 0;
44. }

**КОНСТРУКТУРИ С АРГУМЕНТИ**

Възможно е да се задават аргументи на конструктурите . За тази цел добавете подходящи параметри към декларациите и дефинициите на конструкторите

Пример 5: Демонстрира как конструктура myclass приема един параметър. Стойността, предадена на myclass, се използва за инициализиране на променливата а.

1. #include <iostream>
2. using namespace std;
3. class myclass {
4. int a;
5. public:
6. myclass(int x); // конструктур с аргумент (int х)
7. void show();
8. };
9. myclass::myclass(int x)
10. {
11. cout << "In constructor";
12. a = x;
13. }
14. void myclass::show()
15. {
16. cout << a << endl;
17. }
18. int main()
19. {
20. myclass ob(4);
21. ob.show();
22. return 0;
23. }

Резултат:

In constructor

4

Обърнете внимание как е деклариран обектът ob в main(). Стойността 4, посочена в скобите след ob(), е аргументът, който се предава на myclass за инициализиране на а. За разлика от конструктурите, деструктурите не могат да приемат аргумент, тъй като не съществува механизъм, чрез който да се предават аргументи на обект, който се унищожава.

Пример 6: Тази програма демонстрира конструктор да получава повече от един аргумент

1. #include <iostream>
2. using namespace std;
3. class myclass {
4. int a, b;
5. public:
6. myclass(int x, int y); // конструктур
7. void show();
8. };
9. myclass::myclass(int x, int y)
10. {
11. cout << "In constructor\n";
12. a = x;
13. b = y;
14. }
15. void myclass::show()
16. {
17. cout << a << ' ' << b << "\n";
18. }
19. int main()
20. {
21. myclass ob(4, 7);
22. ob.show();
23. return 0;
24. }

Пример 3: Тази програма показва как да предадете аргумент на конструктора на един производен клас

1. #include <iostream>
2. using namespace std;
3. class base {
4. public:
5. base() { cout << "Constructing base class\n"; }
6. ~base() { cout << "Destructing base class\n"; }
7. };
8. class derived : public base {
9. int j;
10. public:
11. derived(int n) {
12. cout << "Constructing derived class\n";
13. j = n;
14. }
15. ~derived() { cout << "Destructing derived class\n"; }
16. void showj() { cout << j << '\n'; }
17. };
18. int main()
19. {
20. derived o(10);
21. o.showj();
22. return 0;
23. }

**Задача 1**

Създайте клас триъгълник с два конструктора. Първият е без параметри и изисква въвеждане на дължините на страните от потребителя (изисква се проверка на валидност на въведените данни, т.е. дали тези страни могат да са страни на триъгълник). Вторият конструктор създава триъгълник по директно зададени дължини на страни (но отново проверява за валидност на данните). Напишете функции за изчисляване на периметър и лице на така създадения триъгълник.