11. ООП

I Основни принципи на ООП.

1. Капсулиране на данните
2. Полиморфизъм
3. Наследяване
4. Абстракция на данните
5. Композиция

1. Капсулиране на данните - за даден обект данните и операциите над тях са в една структура, наречена клас. Директния достъп е ограничен. Достъпа до данните на обекта се осъществява чрез неговите методи.

2. Наследяване - същността на наследяването е, че всеки подклас придобива автоматично (наследява) всички атрибути и операции на съответния супер-клас.

3. Полиморфизъм - дава възможност различни операции да имат едно и също име. Той позволява един метод на класа да реагира различно, в зависимост от използвания обект.

4. Абстракция на данните - абстракцията опростява сложната реалност чрез моделиране на класове, подходящи за разрешаването на определен проблем. Абстракцията представлява модел, изглед или нещо друго общо представяне на действителен обект.

5. Композиция - обектите могат да се слеят от други обекти, тези обекти могат да се скрият от други класове или да са достъпни чрез интерфейси.

II Класове. Дефиниране на класове. Област на класовете. Обекти

Клас - абстрактно описание на структурата от подобни обекти. Обекта наследява всички операции и атрибути на класа, на който принадлежи. Обекта е екземпляр на класа

1. Декларация клас

class име {

етикет за разрешение 1:

член 1;

...

етикет за разрешение 2:

член n;

} име_обект

- private - само в рамките на класа
- protected - членовете могат да се викат от функции или от приятелски класове
- public - отвсякъде, където се вижда класа

2. Област на класовете

- класовете могат да се дефинират на различни нива в програмата

3. Обекти - след като класа е дефиниран, могат да бъдат създавани негови екземпляри, наричани обекти.

а) операции с обектите:

- достъп до компонентите на обектите.

обект. данна

обект. име_член_фя()

обект. име_на_член_фя(параметри)

- присвояване на обекти

III. Конструктори.

1. Същност

Конструкторите инициализират обектите те са член-функции на класовете

2. Дефиниране на конструктор

- Конструктора се дефинира, като една член-функция се нарече с името на класа. Тази функция-конструктор се извиква тогаз самото т., когато се създава нов екземпляр на класа (когато се декларира нов обект от този клас)

3. Особености на конструкторите

- в 1 клас може да е дефиниран конструктор по-малко и да са декларирани няколко.
- извиква се автоматично при създаване на обекти.
- името на конструктура съвпада с името на класа

4. Видове конструктори

- предефинирани
- подразбиращи се
- с подразбиращи се параметри
- за присвояване и копиране

IV Указатели към обекти. Масиви от обекти. Динамични обекти.

1. Указатели към обекти на класове

cat p;

cat * ptr = &p; задължително ; ptr и ще се инициализират с адреса на обекта p

(*ptr).get_hitnar()

2. Масиви и обекти

Масивът е крайна редица от фиксиран брой елементи, които са от един и същи тип. Към всеки елемент от редицата е възможен пряк достъп, който се осъществява чрез индекс.

а) дефиниция на масив от обекти
- елементите на масив може да са обекти, но от 1 и същ клас
• дефиниране без инициализация
T име на масив [размер]
• дефиниране с инициализация
T име на масив [размер] = { стойност 1, стойност 2, };

3. Динамични обекти
Стек: област за временно съхранение на данните
Хийп: по-постоянна област за съхранение на данни; дълготрайна памет ↓ използва се при работа с т.нар. динамични структури от данни.

* създаването и разрушаването на динамични обекти в C++ се осъществява с операторите **new** и **delete**

- масив: int* arr = new int[5]
delete [] arr;

- динамични масиви: с променлива дължина; в динамичната памет.

V. Деструктори: използва се за освобождаване на динамичната памет след разрушаването на обекта
член-функция.
извиква се при разрушаване на обект с delete

2. Създаване и разрушаване на обекти на класове
При създаване чрез дефиниция автом. разруш. при зав. изп. функц.
При създаване чрез функции за динам. памет - ръчно

VI Оператори. Предефиниране на оператори.
(1) - позиция (префикс, инфикс, постфикс)
- приоритет
- асоциативност - определя реда изпълнение → ←

② Предефиниране: чрез дефиниране на операторни функции.

тип оператор знак (параметри)
предефинираното – по 2 к-ва : функция-приятел
или ф-я.

VII. Производни класове. Дефиниране. Наследяване и достъп до наследените компоненти.

1. Наследяване. Производни класове.
едно общо понятие в ООП

Чрез механизма на наследяването се създава нов клас от съществуващ клас.
съществуващия клас се нарича базов (основен) клас.
създаденият – производен.
наследява се 1 или повече класове; (просто 1-кно Насл)
(множествено насл)

2. Дефиниране на производни класове.

class <произв_клас> : <атрибут> <базов клас>
{ <тяло на произв-клас>
};

Множеството от компонентите на един производен клас се състои от компонентите на базовия клас и от новодефинираните компоненти в самия производен клас.

3. Наследяване и достъп до наследените компоненти.

атрибут за клас	компонентите на основен клас, определени като	наследява се като
public	public private protected	public private protected
private	public private protected	private private private
protected	public private protected	protected private protected

Тема 12 СДА.

I. Основи на анализа на алгоритми: Асимптотична нотация на сложността. Основни рекурентни формули. Примери за анализ на алгоритми

1. Време на изпълнение: зависи от входа.
2. Видове анализ: най-лош случай
   средно време
   най-добър случай.

3. Нотации.
   а) O-нотация (горна граница)
   б) Ω-нотация (долна граница)
   в) Θ-нотация (tight bounds)
   г) o-нотация
   д) ω-нотация.

4. Основни рекурентни формули.

а) Формула 1: при рекурентна прогр., която изпълнява цикъл по входните данни и за да елиминира 1 от тях. $C_n = \frac{n(n+1)}{2}$

б) Формула 2: при рекурсивна програма, която разделя входа си на две половини на една стъпка. $C_n = \lg N$

в) Формула 3 при рекурсивни програми, които разделят входа на две половинки, но освен това може би трябва да изследват всеки елемент от входа. $C_n = 2N$

г) Формула 4. при рекурсивни прогр. които трябва да изследват линейно входа преди, по време и след разделянето на 2. $C_n \approx N \lg N$

д) Формула 5. входа на 2 и освен това правят константна друга обработка $C_n = 2N$

ф 1 $C_n = \frac{n(n+1)}{2}$ ↓ изм. на входа елиминир 1

ф 2 $C_n = \lg N$ ↓ деление вход на две половини

ф 3 $C_n = 2N$ деление вх. на 2 полов. и се изследва всеки елемент на входа.

ф 4. $C_n \approx N \lg N$ на 2 и изм. преди, след и по време делението

ф 5. $C_n = 2N$ изм. входа на 2 и др. обработка.

II. Абстрактни типове данни. Интерфейс и реализация

1. Абстрактни типове от данни. – позволяват да се пишат програми от високо ниво на абстракция.

За да се разработи ново ниво на абстракция е необходимо
- да дефинираме абстрактните типове и операциите върху
- да представим данните чрез определени структури
- да реализираме операциите

def. Абстрактен тип данни (АТД) е тип данни, достъпен само чрез интерфейс. Програма, която използва АТД се нарича клиент, а програма, която определя типа данни – реализация.

III. Свързани списъци. Обработка на списъци.

1. Свързани списъци.

Свързания списък е множество от елементи (items), където всеки елемент е част от възел (node), който също съдържа връзка (link) към възел.

(Когато искаме да обходим последователно съвкупност от елементи, ги организираме в свързан списък
- едно свързан
- двусвързан

- изтриване на възел   x.next = x.next.next
- вмъкване на възел   t.next = x.next   x.next = t

IV. Структура от данни стек. Реализация

def Стек е абстрактен тип данни, който има две основни операции: push (вмъкване) на нов елемент и изтриване (pop) на елемента, който последен е вмъкнат.
Стека е крайна редица от елементи от един и същи тип.
Края на стека се нарича "връх на стека"
First In Last Out (FILO)

- динамичен стек | - свързан стек
- статичен стек | - статичен стек

V Структура от данни опашка. Реализация

First In First Out (FIFO) опашка е абстрактен тип данни, който се състои от две основни операции:
- вмъкване на нов елемент (put)
- премахване на елемента, който е вмъкнат най-рано

Опашката е крайна редица от елементи от един и същ тип. - статична опашка
- свързана опашка

VI Дървета. Типове дървета.

Дърветата са математическа абстракция, която играе централна роля при дизайна и анализа на алгоритми. Дърветата могат да бъдат използвани за описването на динамичните свойства на алгоритмите, също и за построяването на други динамични.

Типове дървета:
- дървета
- дървета с корен
- подредени дървета
- m-арни дървета и двоични дървета.

Дърво е непразна съвкупност от възли и връзки между тях, които удовлетворяват различни изисквания

Възел е проста обект, който може да съдържа информация

Ребро е връзка м/у 2 възела.

Път в дърво е списък от различни възли, свързани последователно с ребра.

Ако съществува > 1 път м/у някой възли, -> **цикъл**.

Дърво с корен е дърво в което сме избрали възел за корен.

Дърво с корен е възел, свързан с множество от други дървета.

Дърво е възел, който е свързан с редица подобни дървета.

Е точно 1 път м/у корена и всеки друг възел в дървото

Родител - възел над възела.
Възли без деца - листа (терминални възли)
Поддърво с корен, с указана подредба на децата
N-арно дърво - всеки възел с определен брой деца.
Двоично дърво - поддърво, състоящо се от 2 типа възли - възли без деца, вътрешни с 2 деца

VII Сортиране. Елементарни методи за сортиране.
- чрез селекция (selectionsort)
- чрез вмъкване (insertionsort)
- метод на мехурчето (bubblesort).

VIII Quicksort - метод за сортиране чрез разделяй и владей.
Масива се разделя на 2 части, на базата на някакъв елемент, и след това 2-те части се сортират независимо.

Реализация на метода Сортиране чрез селекция

```
static void selection (ITEM[] a, int l, int r) {
    for (int i=l; i<r; i++) {
        int min=i;
        for(int j=i+1; j<=r; j++)
        if(less(a[j], a[min])) min=j;
        exch(a,i,min);
    }
}
```

Сортиране чрез вмъкване:

```
static void insertion (ITEM[] a, int l, int r) {
    int i;
    for (i=r; i > l; i--) compExch (a, i-1, i);
    for (i = l+2; i<=r; i++) {
      int j = i; ITEM v = a[i];
      while (less (v, a[j-1])) {
          a[j] = a[j-1]; j--;
      }
      a[j] = v;
    }
}
```

Метод на мехурчето:

```
static void bubble (ITEM[] a, int l, int r) {
    for (int i = l; i<r; i++) {
    for (int j = r; j>i ; j--) {
      compExch (a, j-1, j);
}
```

Quicksort.

```
static void quicksort (ITEM[] a, int l, int r) {
      if (r<=l) return;
      int i = partition (a, l, r);
      quicksort (a, l, i-1);
      quicksort (a, i+1, r);
}

static int partition (ITEM[] a, int l, int r) {
      int i = l-1, j = r; ITEM v = a[r];
    for (;;) {
```

```
        while (less(a[++i], v));
        while (less(v, a[--j])) ; if (j == l) break;
        if (i >= j) break;
        exch(a, i, j);
    }
    exch(a, i, r);
    return i;
}
```