1.	Naloge jedra so: preklapljanje med procesi kontrola in nasljavljanje naprav upravljanje s pomnilnikom in procesi scheduling, medprocesna komunikacija, procesiranje izjem in prekinitev Jedro operacijskega sistema, se nalozi ob samem zagonu operacijskega sistema. Datoteka jedra pa se nahaja v korenskem direktoriju.
2.	Regular file Text ali binary data Directory vsebuje druge direktorije ali datoteke Executable file Fdatoteka z nastavljenim "execute bit-om". Vecina ukazov je Executable. Symbolic link Fdatoteka je bljiznica, ki kaze na drugo datoteko Device special file Vmesnik do dolocenega dela hardwera, kot naprimer printer. Pipe Vmesnik do omreznega programa.
3.	Programski proces je izvajanje zaporedja instrukcij na dani mnozici podatkov. Dogajanje s življenjskem obdobju procesa: - proces nekdo tvori - proces se izvaja - proces čaka na izvajanje - proces nekdo uniči. Stanja procesov: - stanje izvajanja - stanje čakanja na izvajanje. Identifikacija procesa: Vsak proces ima svojo identifikacijsko številko »PID«. Ob rojstvu je »otrok« kopija »starševskega« procesa, loči se le po PID-u. Proces »oče« prepozna in tako lahko vpliva nanj. Nadzorovanje procesov v UNIX-u: - pregledamo jih z ukazom ps - ubijemo jih z “kill PID” ali “killall imeproces”
4.	UKAZI ZA DELO Z IMENIKI IN DATOTEKAMI: - izpis vsebine imenika: ls - kopiranje datotek: cp ime_datoteke - Premikanje datotek: mv - Brisanje imenikov in datotek: rmdir (rm –r * → odstrani vse datoteke in vse poddirektorije); rm ime_datoteke; - Pregled vsebine datoteke: more ime_datoteke; pg ime_datoteke; cat ime_datoteke - Iskanje datotek: $find /| grep *.txt ls – list, cp – copy, mv – move, rm – remove, cat – izpis na enkran
5.	Program: postopek+podatki; Programski proces: izvajanje programa; življenjska doba procesa: življenje,smrt;

6.	DIR, COPY, MOVE, DEL-RD, MORE.-EDIT
7.	*A:* *MD BACK_UP* *CD BACK_UP* *C:* *COPY *.EXE A:* *COPY *.DLL A:* *DIR A:* *DEL *.DLL*
8.	*MOVE *.EXE A:*
9.	Vsebina Beri.bat: ECHO OFF FOR %%Z IN (X.TXT) DO TYPE %%Z ECHO Izpisujem datoteko %%Z
10.	Prevajalnik preveja program, napisan v visokem programskem jeziku v program v strojnem jezik. Interpreter interpetira ukaz za ukazom iz programa napisanega v visokem programskem jezik v mnozico ukazov v storjnem jeziku. (Prevajalnik pretvori kodo visjega programskega jezika v instrukcije, kar pomeni, da ko zazenes preveden program se vedno izvajajo diektne strojne instrukcije katerih je veliko manj, kot ce se program interpretira. Ce pa se program interpretira mora interpreter vsakic ob izvajanju opravit delo prevajalnika(syntax checking, ...) in izvajati instrukcije, ki so produkt interpretacije to pomeni da interpreter v bistvu za vsako izvajanje kompajla in pozene.)
11.	Povezovalniki povezujejo program s knjiznicami. Nalagalniki nalozijo program in knjiznice, ki so potrebne za delovanje programa in razhroscevalnik(debugger) omogoca pregled delovanja programa, in spreminjanje poteka delovanja.
12.	Poznamo: navadno izbiranje(razvit qsort), izbiranje, zamenjave(bubble), zlivanja(navadno, naravno). |nevem ce je zlivanje za sotritanje| Primeri:
13.	Rekurzivno: ponovljivi del, pogoj zaust., klicanje metode do pogoja. Iterativno: sekvenca stavkov, pogojni (brezpogojni) skok. Rekurzivno: bolj naravno, krajša koda, bolj intuitivna, hitrost izvajanja+, potrošnja pomn+. Primer rekurzivnega: PROCEDURE PONOVI(VAR X:INTEGER); BEGIN INC(X); WRITELN(X); IF X< 100 THEN PONOVI(X); END; Primer iterativnega: WHILE X>= 100 DO BEGIN INC(X); WRITELN(X); END;
14.	Objekt je podatkovni tip, sestavljen iz večih lastnosti in metod. Lastnosti takega programiranja: dedovanje, enkapsulacija, polimorfizem. Primer objekta v pascalu:

	Type objekt = object *n: integer;* *procedure vnesi(nn: integer);* *function vrni: integer;* *end;* Primer razreda v Cju: *class razred {* *int n;* *void vnesi(int nn);* *int vrni();* *}*
15.	Pri pascalu: s pomočjo oznake private pred definicijo lastnosti/metode. Pri Cju: tudi s pomočjo oznake private pred definicijo lastnosti/metode.
16.	Pri pascalu: definicijo obekta podamo na način *type objekt = object(podedovani_objekt);* Pri Cju: definicijo obekta podamo na način *class objekt:podedovani_objekt;*
17.	Konstruktor omogoča izvedbo določenih ukazov pri inicializaciji objekta. Potrebujemo ga, kadar objekt vsebuje navidezne metode.
18.	Destructor omogoča izvedbo določenih ukazov ob sprostitvi pomnilnika objekta (če ne želimo izvesti nobenega ukaza ga pustimo praznega). Potrebujemo ga, kadar objekt vsebuje navidezne metode.
19.	Konstruktor uporabimo, če uporabljamo navidezne metode (kliče se ob inicializaciji razreda). Deklaracija: *class razred {* *int n;* *razred() { n = 0; }; // to je konstruktor* *};* Pravila o konstruktorjih: ne morejo vračati nobene vrednosti, lahko imajo parametre, se ne podedujejo, ne morejo biti virtualni, če jih ne definiramo mi, jih definira C++ sam, kličejo se avtomatsko ob sprostitvi pomnilnika razreda.
20.	Destruktor uporabimo, če uporabljamo navidezne metode (kliče se ob sprostitvi pomnilnika razreda). Deklaracija: *class razred {* *int n;* *razred() { n = 0; }; // to je konstruktor* *~razred() { n = 99; }; // to je destruktor* *};* Pravila o destruktorjih: ne morejo vračati nobene vrednosti, lahko imajo parametre, se ne podedujejo, lahko so virtualni, če jih ne definiramo mi, jih definira C sam, kličejo se avtomatsko ob sprostitvi pomnilnika razreda
21.	Vsebina VMT tabele: nisem ziher, tole je direkt iz zvezka - število navideznih metod - 4 (??????)

- offset DMT tabele
- rezervirano
- število dinamičnih metod
- naslov DMT starša
- x
- x
- indeksi navideznih procedur
- naslovi navideznih procedur

jest sm najdu pa takle primer (zivc)

Virtual method table layout

Offset	Type	Description
-76	*Pointer*	pointer to virtual method table (or **nil**)
-72	*Pointer*	pointer to interface table (or **nil**)
-68	*Pointer*	pointer to Automation information table (or **nil**)
-64	*Pointer*	pointer to instance initialization table (or **nil**)
-60	*Pointer*	pointer to type information table (or **nil**)
-56	*Pointer*	pointer to field definition table (or **nil**)
-52	*Pointer*	pointer to method definition table (or **nil**)
-48	*Pointer*	pointer to dynamic method table (or **nil**)
-44	*Pointer*	pointer to short string containing class name

-40	*Cardinal*	instance size in bytes
-36	*Pointer*	pointer to a pointer to ancestor class (or **nil**)
-32	*Pointer*	pointer to entry point of *SafecallException* method (or **nil**)
-28	*Pointer*	entry point of *AfterConstruction* method
-24	*Pointer*	entry point of *BeforeDestruction* method
-20	*Pointer*	entry point of *Dispatch* method
-16	*Pointer*	entry point of *DefaultHandler* method
-12	*Pointer*	entry point of *NewInstance* method
-8	*Pointer*	entry point of *FreeInstance* method
-4	*Pointer*	entry point of *Destroy* destructor
0	*Pointer*	entry point of first user-defined virtual method
4	*Pointer*	entry point of second user-defined virtual method
. . .	. . .	. . .

S pomočjo VMT tabele se navidezne funkcije povežejo z objektom (pozno povezovanje).

22.	Datotečna org. je način shranjevanja in urejanja podatkov v fizičnih datotekah.Vrste: **Neurejena datoteka** - ne obstaja predpis, po karerem so zapisi v dat. urejeni **Urejena datoteka** - zapisi urejeni po vrednosti ključa(pomembna tudi razvrščenost znotraj ključa, ker je tudi ključ lahko iz več elementov) **Razpršena datoteka** - zapisi urejeni po vrednosti ključa
23.	kriteriji: 1. poraba pomnilniškega prostora 2. čas za dostop do poljubnega zapisa v datoteki 3. čas za dostop do logično naslednjega zapisa 4. čas, potreben za dodajanje, brisanje, spreminjanje zapisa 5. čas, potreben za branje vseh blokov fizične dat. v logični zapis
24.	Podatki o diskovnem pomnilniku Dodelitvena tabela Datotečni seznami Fizične datoteke
25.	podatkovni element – je najmanjsi del podatkov zapis – (zapis s konstantno dolzino, zapis z spremelnjivo dolzino) v datoteki.. tip zapisa – z tipom zapisa predstavimo strukturo zapisa in vrsto ter obliko podatkovnih elementov, ki sestavljajo zapis. logicna datoteka – mnozico zapisov istega tipa imenujemo logicna tatoteka
26.	Logični zapisi – zapisi logičnih datotek Fizični zapisi – s pomočjo fiz.zapisov se v bloke fiz.datoteke shranjujejo log.zapisi • Primerjava strukture: struktura enaka, vendar pri fiz. zapisih razširjena z meta podatki kot so kazalci in števci ponovljivosti podatkovih elementov v zapisih. obsega le del podatkovih elementov logičnega zapisa (logičnih zapis je predstavljen z dvema ali več fizičnimi zapisi). Preslikava log. v fiz zapis: 1. vsi log. zapisi v eni fizični datoteki, ki ne vsebuje zapisov drugih fiz. datotek 2. vsi log. zapisi v eni fizični datoteki, ki pa vsebuje zapise drugih fiz. Datotek 3. en logični zapis v večih fizičnih, ki ne vsebujejo zapise drugih log. datotek 4. en logični zapis v večih fizičnih, ki pa vsebujejo zapise drugih log. datotek
27.	iskanje, dodajanje, spreminjanje, brisanje. Izvedba v **NEUREJENI DAT**: ISKANJE - Dostop do zapisov v neurejenih datotekah poteka običajno s pomočjo kazalcev (naslovov), ki so shranjeni v zapisih drugih datotek. V primerih, ko nastopa neurejena datoteka samostojno, iščemo zapise s pomočjo zaporednega iskanja. Princip zaporednega iskanja temelji na preiskovanju fizičnih blokov datoteke enega za drugim v skaldu z njihovim logičnim zaporedjem, dokler ne naletimo na polje z zapisom, ki ustreza iskalnemu pogoju. Iskanje lahko na tem metu prekinemo ali pa nadaljujemo z iskanjem naslednjega zapisa ob istem iskalnem pogoju vse do konca datoteke. Rezultate iskanja je lahko nič, eden ali več zapisov. BRISANJE - Če gre za gosto datoteko, se brisanje zapisov izvaja s prepisovanjem vsebine stare datoteke v novo datoteko. Tak postopek se uporablja v primeru, ko je

	potrebno hkrati izbrisati večje število zapisov. Pri brisanju posameznih zapisov je prepisovanje preveč zamudno, v primeru vezanih zapisov pa sploh neuporabno. V slednjem primeru izbrišemo zapis tako, da označimo polje, v katerem se nahaja kote prosto polje. DODAJANJE - Najhitrejši način dodajanja zapisov je njihovo vpisovanje v prosta polja v zadnjem bloku datoteke. Če je zadnji blok popolnoma zaseden, je potrebno datoteko razširitei, zapis pa se vpiše v prvo polje dodanega bloka.
28.	**ZAPOREDNA s FIZIČNIM ZAPOREDJEM** GOSTA dat. - Pri njej se izvaja dodajanje in brisanje zapisov s prepisovanjem stare datoteke v novo datoteko »ZAMUDNO. Ta vrsta datoteke se v fizičnih podatkovnih bazah uporablja le v primerih, kjer zapis v datoteki le beremo ali jih spreminjajmo. REDKA DATOTEKA omogoča hitro dodajanje oziroma brisanje zaposv. Ob kreiranju datoteke in začetnem vpiso zapisov, ohranimo v vsakme bloku nekaj praznih polj. Le-ta in tista, ki so se izpraznila pri brisanju zapisov, izkoriščamo za shranjevanje kasneje dodanih zapisov. Ob dodajanju je zaradi ohranitve pravilnega zaporedja potrebno v bloku že shranjene zapise po potrebi premakniti v sosednja polja. Ko se kakšen blok datoteke povsem napolni, je potrebno izvesti reorganizacijo datoteke. Kreiramo novo fizično datoteko, ki vsebuje vse zapise stare datoteke, pri tem pa so v fizičnih blokih spet predvidena prazna polja za dodajanje novih zapisov. Reorganizacija je razmeroma dolgotrajen postopek, zato se izvede v času ko podatkovna baza ni aktivna. Da se izvede pravočasno, je treba zasedenost blokov tekoče nadzorovati. Ker se pri dodajanju zapisi praviloma selijo iz polja v polje, je takšna datoteka primerna le za nevezane zapise.
29.	**ZAPOREDNA s POVEZAVO BLOKOV s KAZALCI:** DODAJANJE - 1. Poišči fizični blok, v katerega bi sodil zapis glede na vrednost svojega ključa. Ta blok imenujemo ciljni blok. 2. Če obstaja v ciljnem bloku kakšno prosto polje, vrini zapis na ustrezno mesto v bloku, pri tem pa po potrebi premakni obstoječe zapise v sosednja polja tega bloka. 3. Če so v ciljnem bloku zasedena vsa polja, dodeli datoteki nov fizični blok, ki ga vrinemo za ciljni blok, ter ustrezno spremeni povezovalne kazalce. Zapise ciljnega bloka vključno z dodanim zapisom razdelimo med ciljni blok in novo dodani blok datoteke, tako da sta oba enakomerno zasedena. BRISANJE – 1. Poišči fizični blok (ciljni blok), v katerem se nahaja zapis, ki ga nameravamo izbristi iz datoteke. **2.** Polje, v katerem se nahaja zapis, označimo kot prosto polje. **3.** Če je v ciljnem bloku manj kot polovica polj zasedenih z zapisi, potem zapise ciljnega bloka in zapise sosednjega bloka porazdelimo enakomerno na oba

<table>
<tr><td></td><td>bloka - t.i. zlivanje blokov. (Sosednji blok je lahko blok, ki se v datoteki logično nahaja pred ciljnim blokom ali pa za njim.) Če je zapisov v obeh blokih toliko, da jih lahko shranimo v en blok, to tudi storimo, iz fizične datoteke pa izločimo odvisni blok z ustrezno spremembo povezovalnih kazalcev.</td></tr>
<tr><td>30.</td><td>ZAPOREDNA s FAKTORIZACIJO:

DODAJANJE -

1. shranimo zapis tipa ostali_podatki v neurejeno datoteko Ostali_podatki
2. dodamo še zapis tipa ključ s kazalcem v zaporedno datoteko Ključ, pri čemer kaže kazalec na shranjeni zapis 'ostali podatki'

BRISANJE – Zadošča, da izbrišemo le ustrezni zapis iz datoteke Ključ, saj 'osali podatki postanejo tako nedostopni. Če želimo v datoteki Ostali_podatki izkoriščati za shranjevanje zapisov tudi polja izbrisanih zapisov, je potrebno tudi polje, v katerem se nahaja zapis 'ostali podatki', označiti kot prosto polje. (počasnejše, vendar pride do boljše izkoriščenosti diskovnega pomnilnika).</td></tr>
<tr><td>31.</td><td>Pri zaporednih datotekah se zapisi logične datoteke preslikajo v zapise fizične datoteke tako, da so v njej urejene po vrednosti ključa. Ker je ključ lahko sestavljen iz več podatkovnih elementov, je pomembna tudi razvrstitev podatkov znotraj ključa. Zapisi znotraj datoteke so urejeni s fizično urejenostjo zapisov, s kazalci ali pa s faktorizacijo.</td></tr>
<tr><td>32.</td><td>
<table>
<tr><td>Boot – kje se nahaja fat, dir, data</td></tr>
<tr><td>FAT</td></tr>
<tr><td>Dir – ime datoteke, naslov prve dodel. Enote</td></tr>
<tr><td>DATA</td></tr>
</table>

Zgradba FAT-a:

<table>
<tr><td colspan="2">Naslov Prve Dodel.Enote</td></tr>
<tr><td>Kazalec 1 (kaže na DATA)</td><td>Kazalec 2 (naslov nasl. D.E.)</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td colspan="2">Naslov naslednje D.E.</td></tr>
</table>

DATA
//////////////////////
Fiz.podatki

Postopek branja: direktno pregledamo zaporedje do najdbe dat.(dobimo naslov), nato gremo v FAT, preberemo v pomnilnik, preberemo drugi del.</td></tr>
<tr><td>33.</td><td>Lahko samo A,B, vendar …</td></tr>
<tr><td>34.</td><td>Preobloženost določa: ime enolično, vrsta, število parametrov. Lahko samo primer B(napaka pri vajaX?). …</td></tr>
<tr><td>35.</td><td>Abstraktne razrede uporabljamo kot osnovo, iz katere nato podedujemo različne objekte, ki imajo vse značilnosti abstraktnega razreda in nekaj svojih dodatnih. Vsak izpeljan razred mora imeti implementirane vse tri funkcije, da ne bo več abstrakten.</td></tr>
</table>

36. Za prijateljske razrede je značilno, da lahko dostopajo do privatnih lastnosti/metod razreda, ki je definiran kot prijatelj. To velja samo v eno smer (odgovor na vprašanje je NE).

37. Za enozložna cela števila (short int) lahko preobložimo operator +. V struct lahko preobložimo operator ++ tako, da zmanjšamo vrednost.

38. Statični član:

```
struct Tobjekt {
        static int a;
        int b; };
int Tobjekt::a;
```

Vsi objekti tega tipa si delijo isti pomnilniški prostor za to lastnost.

Primer:

```
#include <conio.h>
#include <stdio.h>

struct Tobjekt {
        static int a;
         int b;};
int Tobjekt::a;

Tobjekt spr1, spr2;

void main() {
  spr1.a = 10;
  spr1.b = 20;
  spr2.a = 30;
  spr2.b = 40;
  clrscr();
  printf("\n Vrednosti: spr1.a = %d, spr1.b = %d, spr2.a = %d, spr3.b = %d", spr1.a,
spr1.b, spr2.a, spr2.b);
  getch();
};
```

39. V prvem primeru kličemo vse funkcije enako: objekt.metoda, v drugem pa ???

40. Pri *private* dedovanju, postanejo vse metode/lastnosti privatne.
Pri *protected* dedovanju postanejo vse metode/lastnosti privatne razen tistih, ki so bile definirane kot *public* (te postanejo *protected*).
Pri *public* dedovanju vse metode/lastnosti ostanejo iste vrste kot so bile prej.

41. **.) PASCAL**

Aritmetični op.	Logični op.	Bitni op.	Primerjalni op.
+	Not		=
-	And		<>
*	Or		<
/	Xor		>
	Shl		<=
	Srr		>=
			In

42. **C++**

Aritmetični op.	Logični op.	Bitni op.	Primerjalni op.
+	&&	&	<
-	||	^	>
*		|	<=
/			>=
			==
			!=

43. G,J,M.

44. Vrste:
postopkovni (c, pascal, basic) – napišemo zaporedje ukazov ali algoritem za reševanje problema.
nepostopkovni (SQL, Prolog) – postavimo vprašanje, računalnik pregleda svojo bazo in vrne odgovor

45. Algoritem:
Zaželjene last.alg:
Program: postopek + podatki
Povezava:

46. Podatek : poljubna predstavitev s pomočjo simbolov ali analog.količin, ki se ji pripiše nek pomen
Informacija : je pomen, ki ga človek pripiše podatkom z uporabo znanih dogodkov, ki so uporabljeni pri predstavitivi podatkov.
Sporočilo:

Merska enota za količino info: **1 bit**

47. Kod: niz določenih znakov; Kodiranje: opis podatka nekega sveta s predstavitvijo podatka drugega sveta; Enkripcija: skrivanje nekega podatka.

48. Podatek je poljubna predstavitev s pomočjo simbolov in analognih veličin, ki jim je pripisan nek pomen. Vrste podatkov: simboli – diskretni; analogne veličine – analogni podatki. Podatki so v računalniških sistemih predstavljeni v dvojiškem številskem sistemu. Alfanumerične znake predstavljamo s 7-bitno ali 8-bitno ASCII tabelo ali podobnimi tabelami.

49. $1011101_{(2)}$ $11111_{(2)}$ $10000_{(2)}$
$- \ 110111_{(2)}$ $+ \ 1000_{(4)}$ $+ \ 111_{(5)}$
$0100100_{(2)}$ $1011111_{(2)}$ $101111_{(2)}$

50. osnovna ASCII – 49 bitov (7*7)
razširjena ASCII – 56 bitov (8*7)
min. Število bitov – 21 (3*7)