BPPC OPR

1) a) Vztah ukazatelové aritmetiky (ukazatelů) a polí (uveďte ekvivalentní zápisy), Ukazatelová aritmetika.
Nadeklarujte jednorozměrné pole prvků long velikosti 50 prvků. Zapište hodnotu 23 na třicátou pozici oběma způsoby
3b

b) Typy dědění a přístupová práva ke členům v odvozených třídách
3b

2) Napište funkci, která vloží do prvního řetězce druhý řetězec na danou pozici. Funkce má mít tři parametry a nemá vracet žádnou hodnotu. Prvním parametrem je řetězec, do kterého se bude vkládat. Druhým parametrem je vkládaný řetězec. Třetím parametrem je pozice, na kterou se má vkládat. Nadeklarujte oba řetězce. Nadeklarujte a nadefinujte tuto funkci a naznačte volání funkce s těmito řetězci jako parametry.

funkci jsou předány dva řetězce a pozice v prvním řetězci, kam se má vložit řetězec druhý (řetězce jsou alokovány dynamicky pro skutečnou délku řetězce). Ošetřete chybové stavy (nemožnost připojení mimo rozsah řetězce, …) . naznačte a popište volání (včetně definicí a ukončení proměnných). funkce nevrací žádnou hodnotu.
13b

3) napište třídu :TSstring tak aby šlo přeložit a fungoval následující program:
TString obsahuje dynamicky alokované pole charů pro řetězec.

```
{
TString  a(„abc“), b=a,c; // konstruktor z řetězce, kopykonstruktor, implicitní konstruktor
c = a + b; // přiřazení (=) spojených řetězců (+)
a = a;
int i1 = c.Delka();   // vrací délku řetězce
int i2 = VyskytZnaku(a,’b’); // vrátí počet výskytů daného znaku v řetězci
char znak = c[i-1]; // vrátí znak na dané pozici, při indexaci mimo řetězec vrací odkaz na globální proměnnou
c[2] = ‘e’;  // index musí fungovat i pro přiřazení
int j = a > b;
cout << a << “ to je a “;
}
```

20b

4) co se vypíše po spuštění tohoto programu. Tištěný text napište k příslušnému řádku funkce main, kterého se týká:

```
class A {
public:
A(void) {cout << 'a';}
virtual ~A(void) {cout << 'b';}
void f(void) {cout << 'c';}
virtual fv(void) {cout << 'd';}
};

class B:public A {
A a;
public:
B(void) {cout << 'e';}
virtual ~B(void) {cout << 'f';}
void f(void) {cout << 'g';fv();}
virtual fv(void) {cout << 'h';}
};

class C:public A {
B a;
public:
C(void) {cout << 'i';}
virtual ~C(void) {cout << 'j';}
void f(void) {cout << 'k';fv();}
};


int main ()
{
 A *a;
 B b;
 B *c = (B*)new C;
 a = &b;
 a -> f();
 a -> fv();
 c -> f();
 c -> fv();
 delete c;
 b.f();
 b.fv();
}
```

5) Vytvořte program který načte binární soubor jehož název je zadán jako parametr z příkazového řádku při spuštění programu. Program vytvoří statistiku četnosti jednotlivých bajtů ze souboru a výslednou tabulku vytiskne na konzolu seřazenou dle velikosti. Nejčastěji se vyskytující hodnota bude zobrazena jako první. (10b)

Označte (pomocí ✓ na konci řádku) ty řádky, které považujete z hlediska překladu a funkčnosti programu za správné.

```
#include<stdio.h>
#include<stdlib.h>
#define POCET 256  /* pocet prvku statistiky */
struct TPrvek
        {
        char znak;
        int pocet;
        };
int setrid(struct TPrvek *aPrvky)
        {
        struct TPrvek tmp;
        int i, zmena = 0;
```

1	`for(i=0;i<POCET;i++)`
2	`for(i=1;i<POCET;i++)`
3	`for(i=0;i<POCET-1;i++)`

4	`if((aPrvky[i].pocet) < (aPrvky[i+1].pocet))`
5	`if((aPrvky[i].pocet) > (aPrvky[i+1]->pocet))`
6	`if((aPrvky[i]->pocet) < (aPrvky[i+1]->pocet))`

```
          {
```

7	`tmp.pocet = aPrvky[i].pocet; tmp.znak = aPrvky[i].znak;`
	`aPrvky[i].pocet = aPrvky[i+1].pocet; aPrvky[i].znak = aPrvky[i+1].znak;`
	`aPrvky[i+1].pocet = tmp.pocet; aPrvky[i+1].znak = tmp.znak;`

8	`tmp.pocet = aPrvky[i]->pocet; tmp.znak = aPrvky[i]->znak;`
	`aPrvky[i]->pocet = aPrvky[i+1]->pocet; aPrvky[i]->znak = aPrvky[i+1]->znak;`
	`aPrvky[i+1]->pocet = tmp.pocet; aPrvky[i+1]->znak = tmp.znak;`

9	`tmp->pocet = aPrvky[i]->pocet; tmp->znak = aPrvky[i]->znak;`
	`aPrvky[i]->pocet = aPrvky[i+1]->pocet; aPrvky[i]->znak = aPrvky[i+1]->znak;`
	`aPrvky[i+1]->pocet = tmp->pocet; aPrvky[i+1]->znak = tmp->znak;`

```
          zmena = 1;
          }
        return(zmena);
        } /* setrid() */
int main(int argn, char *argv[])
        {
        int i, znak;
        FILE *fsrc;
        struct TPrvek *prvky;
```

10	`if(argn != 2) return(1);`
11	`if(argn = 2) return(1);`
12	`if(argn < 2) return(1);`

13	`if((fsrc = fopen(argv[1], "rb")) == NULL) return(2);`
14	`if(fsrc = fopen(argv[0], "rt") != NULL) return(2);`
15	`if((fsrc = fopen(argv[1], "at")) == NULL) return(2);`

16	`prvky = (struct TPrvek *) malloc(POCET * sizeof(struct TPrvek));`
17	`prvky = (struct TPrvek *) malloc(POCET * sizeof(TPrvek));`
18	`prvky = (struct TPrvek *) malloc(POCET * sizeof(TPrvek) + 1);`

19	`if(prvky == NULL)`
20	`if(prvky == EOF)`
21	`if(prvky != NULL)`

```
          { fclose(fsrc); return(3); }
        for(i = 0; i < POCET; i++)
```

22	`{ (&prvky[i])->znak = i; (&prvky[i])->pocet = 0; }`
23	`{ prvky[i]::znak = i; prvky[i]::pocet = 0; }`
24	`{ prvky[i]->znak = i; prvky[i]->pocet = 0; }`

25	`while((znak = getc(fsrc)) != EOF) if(znak < POCET) prvky[znak].pocet++;`
26	`while(znak = getc(fsrc) != EOF)  if(znak < POCET) prvky[znak]::pocet++;`
27	`while(znak = (getc(fsrc) != EOF)) if(znak < POCET) prvky[znak]->pocet++;`

```
        while(setrid(prvky));
        for(i = 0; i < POCET; i++)
```

28	`printf("%c se vyskytl %d\n",prvky[i].znak, prvky[i].pocet);`
29	`printf("%c se vyskytl %d\n",prvky[i]->znak, prvky[i]->pocet);`
30	`printf("%c se vyskytl %d\n",prvky[i]::znak, prvky[i]::pocet);`

```
        free(prvky);
        fclose(fsrc);
        return(0);
        } /* main() */
```