- [2 body]

Je definovaný typ:
typedef enum {AUTO, VLAK, LIETADLO, AUTOBUS} DOPRAVA;
Definujte, čo je to kardinalita dátového typu. Podľa uvedenej definície uveďte, aká je kardinalita typu DOPRAVA?

Definícia: Kardinalita dátového typu T je počet rôznych hodnôt prislúchajúcich typu T.

Kardinalita typu DOPRAVA je: 4

- [2 body]

Je definovaný typ:
typedef enum {CERVENA, ZELENA, ZLUTA} BARVY;
Definujte, čo je to kardinalita dátového typu. Podľa uvedenej definície uveďte, aká je kardinalita typu BARVY?

Definícia: Kardinalita dátového typu T je počet rôznych hodnôt prislúchajúcich typu T.

Kardinalita typu BARVY je: 3

- [2 body]

Je daný nasledujúci kód:

```
signed int z, x =7, y = 22;
z = x++ - y/5;
```

aká bude hodnota premennej „z“ po jeho prevedení?

Hodnota premennej „z“ bude rovná 3.

- [2 body]

Je definované inicializované pole:
int a [10] = {1, 2, 3, [8] = 8, 10};
Akú hodnotu bude mať prvok a [10]?

Nie je definované.

- [2 body]

Je definované inicializované pole:
int a [10] = {1, 2, 3, [7] = 10, 9, 8};
Akú hodnotu bude mať prvok a [9]?

Prvok a[9] bude mať hodnotu 8.

- [2 body]

Čo vykoná daná konštrukcia kódu?:

While ((int c = getchar ()) != EOF) putchar(c);

Táto konštrukcia skopíruje štandardný vstup na štandardný výstup.

- [2 body]

Je daný nasledujúci kód:

```
#include <stdio.h>
struct osoba{int vek; double vaha; int vyska;};
typedef struct osoba CLOVEK;
int main(void)
{ CLOVEK c4 = {vek = 30}; CLOVEK c3 = {vyska = 175}; c4 = c3;
printf(“%d“, c4.vek);
return 0; }
```

Čo sa zobrazí na štandardný výstup po jeho prevedení?

Zobrazí sa 0.

- [2 body]

Je daný nasledujúci kód:

```
#include <stdio.h>
struct osoba{int vek; double vaha; int vyska;};
typedef struct osoba Clovek;
int main(void)
{ Clovek c4 = {.vek = 25, .vyska = 165}; Clovek c3 = {.vek = 35};
c4 = c3; printf(“%d“, c4.vyska); return 0; }
```

Čo sa zobrazí na štandardný výstup po jeho prevedení?

Zobrazí sa 0.

- [2 body]

Vysvetli pojem dekompozícia.

- **dekompozícia** = technika programovania, kedy z jednoduchších (elementárnych) častí kódu (podprogramov) vytvárame všeobecnejšie časti kódu, riešiace zložitejšie problémy. Dekomponovať znamená chápať zložité problémy v podobe kolekcie jednoduchších pod problémov. Tiež to znamená nájsť v zložitom probléme také hierarchické usporiadanie, ktoré umožnia zapísať zložité akcie pomocou akcií jednoduchších.

- [2 body]

Vysvetli pojem hromada.

- **hromada(heap)** = je pamäťový priestor, ktorý je výhradne v rukách programátora. Programátor si v pamäti heap dokáže rezervovať presnú veľkosť pamäte podľa jeho požiadaviek (funkcia malloc). Na konci každého programu by mal pomocou (free) zabratú časť pamäti uvoľniť.

- [2 body]

Je daný nasledujúci kód:

unsigned int a = 1, b = 2, c;

c = a & b || a && b;

Akú hodnotu bude mať premenná “c“ po prevedení tohto kódu?

Premenná „c“ bude mať hodnotu 1.

- [2 body]

Jaká bude hodnota proměnné sum po provedení následujícího kódu:
kód:

```
int sum = 1 ;
	for ( int i = 0 ; i < 10 ; i ++ ){
		switch ( i ){
			case 1 :
			case 3 :
```

```
                case 7 : sum ++ ;
                default : continue ;
                case 5 : break ;
        }
        break ;
}
```

Riešenie: 3

- [4 body]

Definujte premennú štruktúrovaného typu pre uloženie údajov pre 500 študentov. O každom študentovi sa bude ukladať: priezvisko (maximálne 15 znakov), miesto bydliska (maximálne 30 znakov), ulica (maximálne 30 znakov), popisné číslo, štipendium (v celých korunách). Miesto bydliska, ulicu a číslo popisné definujte ako vnorenú štruktúru. Inicializujte údaj prvého študenta hodnotami o svojej osobe.

```
typedef struct
 {
        char misto[31] ;
        char ulice[31] ;
        int cislo ;
        } Tadresa ;
typedef struct
 {
        char prijmeni[16] ;
        Tadresa adresa ;
        int stipendium ;
        } Tstudent ;
Tstudent studenti[500] = { { .prijmeni = "Adamec,
.adresa.misto = "Brno", .adresa.ulice = "Vinohrady",
.adresa.cislo = 25, .stipendium 6125 }, } ;
// také správně
Tstudent studenti[500] = { { .prijmeni = "Adamec, .adresa = {
.misto = "Brno", .ulice = "Vinohrady", .cislo = 25 },
.stipendium 6125 }, } ;
```

- [4 body]

Definujte premennú štruktúrovaného typu pre uloženie údajov pre 50 študentov. O každom študentovi sa bude ukladať: priezvisko (max 20

znakov), deň narodenia, mesiac narodenia (max 8 znakov), rok narodenia, štipendium (v celých korunách). Dátum narodenia definujte ako vnorenú štruktúru. Inicializujte údaje prvého študenta hodnotami o svojej osobe.

```
Typedef struct{
Int den;
Char mesiac[9];
Int rok;
}narodeniny;

Typedef struct{
Char priezvisko[21];
Int stipendium;
}student;

student[0].priezvisko = „novak“;
student[0].stipendium = 1000;
student[0].narodenie.den = 3;
student[0].narodenie.mesiac = „marec“;
student[0].narodenie.rok = 2000;
```

- [4 body]

Aká bude hodnota premennej “sum“ po prevedení nasledujúceho kódu?

```
int sum = 0;
for (int i = 0; i < 10; i++){
        switch(i){
                case 1: case 4: case 7: sum++;
                default: continue;
                case 5: break;
        }
break;
}
```

Premenná “sum“ bude mať hodnotu 2.

- [4 body]

Aká bude hodnota premennej “sum“ po prevedení nasledujúceho kódu?

```
int sum = 1;
for (int i = 0; i < 10; i++){
	switch(i){
		case 1: case3: case 4: case 7: sum++;
		default: continue;
		case 5: break;
	}
break;
}
```

Premenná “sum“ bude mať hodnotu 4.

- [2 body]

Vysvetlite princíp binárneho vyhľadávania:

Vyhľadávaný kľúč sa porovná s kľúčom položky, ktorá je umiestnená v polovici vyhľadávaného poľa. Ak dôjde ku zhode, končí vyhľadávanie úspešne. Ak je vyhľadávaný kľúč menšie, postupuje sa porovnávaním prostredného prvku v ľavej polovici pôvodného poľa, ak je vyhľadávaný kľúč väčší postupuje sa porovnávaním prostredného prvku v pravej polovici pôvodného poľa. Vyhľadávanie končí neúspechom v prípade, že vyhľadateľná časť poľa je prázdna (jej ľavý index je väčší ako pravý).

- [2 body]

Je daný nasledujúci kód:

```
Signed int z, x = 7, y = 18;
Z = x++ - y/5;
```

Aká bude hodnota premennej z po jej prevedení?

Riešenie: Z= 4

- [2 body]

Čo najviac zjednodušte zadanú funkciu tak, aby sa previedlo čo najmenej operácií. Upravená funkcia musí vracať totožné logické hodnoty, ako funkcia zadaná:

```
Int test(int x, int y, int z)
{
If (x<=0)
        If (y<0) {return 1;}
                Else
                        if (z>=0){return 1;} else {return 0;}
        else
        if (z=>0){return 1;}else {return 0;}
}
```

Riešenie: (not sure)

```
If (x<=0 &&( y < 0 || y > 0) && z>= 0 ){return 1;}
Else {return 0};
```

- [2 body]

Definujte Floydovu metódu:

Floydova-Steinbergovo metóda distribúcie chyby je algoritmus slúžiaci k zmene farebnej charakteristiky obrazu pri zachovaní pôvodnej vizuálnej informácie v čo najväčšej miere. Tento algoritmus je bežne používaný na úpravu obrázkov. Napríklad pri prevádzaní do formátu GIF dôjde týmto prevodom k obmedzeniu maximálneho počtu farieb na 256. Algoritmus spracováva vstupné pixely zľava doprava, zhora nadol. Pri zaokrúhlenie vstupné intenzity vzniká chyba, túto chybu si algoritmus zapamätá a použije ju k modifikácii hodnôt ďalších pixelov. Chyba sa distribuuje iba medzi susednými pixelmi, ktoré ešte neboli upravované, tzn. že sa upravuje vždy nasledujúci pixel na riadku a pixely na ďalšom riadku.
V niektorých prípadoch sa smer distribúcie chyby mení po každom riadku "cik-cak" (zig-zag) alebo má špeciálny trajektóriu, napr. Hilbertova krivka.

- [4 body]

Aká bude hodnota premennej “sum“ po prevedení nasledujúceho kódu?

```
int sum = 1;
for (int i = 0; i < 10; i++){
	switch(i){
		case 1; case 4; case 7; sum++;
		default: continue;
		case 5: break;
	}
break;
}
```

Premenná “sum“ bude mať hodnotu 3.

- [5 bodov]

Je definovaný typ:
typedef float (*Pcena)(double, int);
Slovne popíšte tento definovaný typ.

“Pcena“ je ukazateľ (pointer) na funkciu s dvoma parametrami špecifikovanými na predávanie hodnotou, ktorá vracia hodnotu typu float.

- [3 bodov]

Vysvetlite pojmy: „ukazovateľ“, „referencia“, a „dereferencia“. Použjte operátory „referencie“ a „dereferencie“ na vhodných operandoch a výsledky uložte do premenných vhodných typov (deklarujte premenné vhodných typov).

- **ukazovateľ** = dátový typ, ktorého hodnota sevyjadruje umiestnenie/adresu dát v pamäti
- **referencia** = hodnota pre nepriamy prístup k premennej
  použitie operátoru referencie =
  int a = 5; int *b = &a;
- **dereferencia** = prístup k hodnote premennej cez jej referenciu
  použitie operátoru dereferencie =
  int c = *b

- [5 bodov]

Je definovaný typ:

typedef double (*Pcena)(double, int);

Slovne popíšte tento definovaný typ.

“Pcena“ je ukazovateľ (pointer) na funkciu s dvoma parametrami špecifikovanými na predávanie hodnotou, ktorá vracia hodnotu typu double.

- [4 body]

Uveďte výhody a nevýhody modulárneho programovania (aspoň 4 výhody). Každú výhodu charakterizujte.

Výhody:

- **zrozumiteľnosť** = modul sa zaoberá iba jedným problémom
- **spoľahlivosť** = jasne definovaná jednotka je lepšie testovateľná
- **udržateľnosť** = malé časti sa lepšie udržujú, menia, upravujú
- **znovu použiteľnosť** = efektívne využitie zdrojov
- **možnosť súbežného vývoja** = viac modelov môže byť vyvíjaných súbežne

Nevýhody:

- **problémy vyplývajúce z integrácie modulov** = testovanie komunikácie modulov
- **konkretizácia modulov** = možné chyby vyplývajúce z neznalosti implementačných detailov

---

- [4 body]

Definujte export a inport modulu:

- **export modulu** = množina funkcii, dátových typov, identifikátorov, ktoré modul poskytuje
- **inport modulu** = množina funkcii, dátových typov, identifikátorov, ktoré modul vyžaduje

---

- [4 body]

Uveďte, ktoré programové konštrukcie sú z hľadiska tvorby modelov vhodné v hlavičkovom súbore (.h), a ktoré v zdrojovom kóde súboru (.c).

- **hlavičkový súbor** (aspoň 4 konštrukcie): prototypy funkcií, deklarácia dátových typov, všeobecné makrá, spätné deklarácie globálnych premenných
- **zdrojový súbor** (aspoň 2 konštrukcie): definícia funkcií, deklarácie globálnych premenných

---

- [8 bodov]

Upravte nasledujúcu funkciu tak, aby zobrazila všetky prvky nad vedľajšou diagonálou štvorcovej matice radu n (po riadkoch v pôvodnom poradí) uložené v dvojrozmernom poli, danom parametrom array.

```
void printAboveSideDiag (int n, int array[n][n]){
        for (int i = 1; i < n; i++){
                for (int j = 1; j < i-1; j++)
                        printf ("%d  ", array[i][j]); //!!tento príkaz neupravujte!!
                        printf ( "\n");
        }
return;
}
```

- [8 bodov]

Upravte nasledujúcu funkciu tak, aby zobrazila všetky prvky pod vedľajšou diagonálou štvorcovej matice radu n (po riadkoch v pôvodnom poradí, tj. v oboch smeroch v smere rastúcich indexov) uložené v dvojrozmernom poli, danom parametrom array.

```
void printUnderSideDiag (int n, int array[n][n]){
\\parameter n je usporiadanie matice
        for (int i = n-1; i = n; i++){
                for (int j = i; j > n; j++)
                        printf ("%d  ", array[i][j]); //!!tento príkaz neupravujte!!
                        printf ( "\n");
        }
return;
}
```

- [8 bodov]

Upravte nasledujúcu funkciu tak, aby zobrazila všetky prvky na hlavnej diagonále a na vedľajšej diagonále štvorcovej matice usporiadania n

v poradí zľava doprava, zhora dole uložené v dvojrozmernom poli daným parametrom array.

```
void printMainSideDiag (int n, int array[n][n]){
\\parameter n je usporiadanie matice
        for (int i = 1; i < n - 1; i ++){
                if (i == n)
                        printf (“%d  “, array[i][i]);
                else
                        printf (“%d %d \n“, array[i][i < n/2 ? i: n-i-1]), array[i][i < n/2 ? n-
i-1 :i])
}
```

- [8 bodov]

Upravte nasledujúcu funkciu tak, aby zobrazila prvky pod hlavnou diagonálou štvorcové matice usporiadania n (po riadkoch v pôvodnom poradí) uložené v dvojrozmernom poli danej parametrom array. Úpravu vykonajte s ohľadom na maximálnu efektivitu programu. Úpravu vyznačte priamo v programe.

```
#include <stdio.h>
void printUnderDiag ( int n, int array[n][n] ){
        for (int i = 0; i < n - 1; i ++){
                for ( int j = 1 ; j < i ; j ++ )
                        printf ( "%d ", array[i][j] ) ; // !! tento příkaz neupravujte !!
                printf ( "\n" ) ;
        }
 }
```

- [8 bodov]

Upravte nasledujúcu funkciu tak, aby zobrazila všetky prvky na hlavnej diagonále štvorcovej matice usporiadania n po riadkoch v opačnom poradí uložené v dvojrozmernom poli daným parametrom array.

```
void printMainDiagRev (int n, int array[n][n]){
\\parameter n je usporiadanie matice
        for (int i = 1; i <= n + 1; i--){
                printf (“%d  “, array[n - 1][n - 1]);
```

```
}
```

- [8 bodov]

Upravte nasledujúce 2 funkcie tak, aby každá zobrazila všetky prvky na hlavnej diagonále štvorcovej matice usporiadania n po riadkoch v opačnom poradí uložené v dvojrozmernom poli daným parametrom array.

```
void printMainDiagRev_A (int n, int array[n][n]){
                n-  >=  i--
        for (int i = 0; i < n ; i++)
                printf ("%d  \n", array[i][i]);
}
void printMainDiagRev_B (int n, int array[n][n]){
        for (int i = 0; i < n ; i++)
                printf ("%d  \n", array[i][i]);
}
```

[n-i-1][n-i-

- [10 bodov]

Je daný dátový typ TMatrix, ktorý umožňuje uložiť dvojrozmerné pole. Je daný prototyp funkcie "initUpperTriangular", ktorá vracia inicializovanú hodnotu typu TMatrix obsahujúcu hornú trojuholníkovú maticu.

typedef struct { int rows, cols, **matrix; } TMatrix;

```
TMatrix initUpperTriangular(unsigned int n, int *errCode){
        *errCode = 0;
        TMatrix m = { .rows = n, .cols = n };
        m.matrix = malloc(n*sizeof(int *));
        if (m.matrix == NULL){ *errCode = -1; return m; }
        for ( int r = 0; r < n; r++){
                m.matrix[r] = malloc((n – r) *sizeof(int));     // (n-r) – dĺžka riadku
                if (m.matrix[r] == NULL) { *errCode = r + 1; return m; }
        }
return m;
}
```

- [10 bodov]

Je daný dátový typ:

```
typedef struct{
        int rows, cols;   //rozmery
        int **matrix;    //2D pole
} Tmatrix;
```

Implementujte funkciu “initMatrix“, ktorá alokuje pole a vracia inicializovanú hodnotu typu TMatrix.

```
TMatrix initMatrix (int rows, int cols, int *errCode){
        *errCode = 0;
        TMatrix m = { .rows = rows, .cols = cols };
        m.matrix = malloc(rows * sizeof(int *));
        if (m.matrix == NULL){ *errCode = -1; return m; }
        for ( int r = 0; r < n; r++){
                m.matrix[r] = malloc(cols *sizeof(int));     // (n-r) – dĺžka riadku
                if (m.matrix[r] == NULL) { *errCode = r + 1; return m; }
        }
return m;
}
```

- [10 bodov]

Je dané:

```
typedef struct {...; int pay;} tdata;
struct item { tdata data; titem *next;};
typedef struct item titem;
typedef struct { titem *head; ... } tlist;
```

Definujte funkciu “listFindMin“, ktorá vracia ukazateľ na položku s minimálnou hodnotou zložky “pay“ v lineárnom zozname (danom parametrom funkcie). V prípade, že je zoznam prázdny, funkcia vracia NULL.

```
titem * listFindMin (tlist *list){
        titem *tmp = list->head;
        titem *minitem = tmp;
        while (tmp != NULL){
                if (tmp->data.pay < minitem->data.pay)
                        minitem = tmp;
```

```
            tmp = tmp->next;
        }
return minitem;
}
```

- [10 bodov]

Je dané:

```
Typedef struct{
        Char name[11];
        Char surname[16];
        Int pay;
}tdata;
Struct item{
        tdata data;
        titem *next;
};
Typesef struct{
        titem *head;
        titem *tail;
} tlist;
Int sumFamilyPay )const tlist *list, char *surname);
```

Definujte funkciu sumFamilyPay podľa zadaného prototypu, ktorá vracia súčet platov (položka pay) všetkých osôb so zhodným zadaným priezvisko surname. Problémy s pretekaním hodnôt typu int nie je potreba v tejto úlohe riešiť. Môžete využiť systémovú funkciu strcmp.

Riešenie: extrémne sa mi to nechce robiť srry

- [10 bodov]

Je dané:

```
typedef struct {...; int pay;} tdata;
struct item { tdata data; titem *next;};
typedef struct item titem;
typedef struct { titem *head; ... } tlist;
```

Definujte funkciu “listFindMin“, ktorá vracia ukazateľ na položku s maximálnou hodnotou zložky “pay“ v lineárnom zozname (danom parametrom funkcie). V prípade, že je zoznam prázdny, funkcia vracia NULL.

```
titem * listFindMax (tlist *list){
	titem *tmp = list->head;
	titem *maxitem = tmp;
	while (tmp != NULL){
		if (tmp->data.pay > maxitem->data.pay)
			maxitem = tmp;
		tmp = tmp->next;
	}
return maxitem;
}
```

- [12 bodov]

Čo sa zobrazí po prevedení nasledujúceho programu? Uveďte presne výsledok, ktorý sa zobrazí na štandardný výstup.

```
unsigned long vypocet (unsigned long a, unsigned long b){
if (a*b == 0) return 0; //namiesto A
if (a < b) {
	a^ = b; b^ = a; a^ = b; //namiesto B
}
return (a % b> 0) ? vypocet(b, a % b):b; //namiesto C
}
int main(void){
	unsigned long x = 35;
	unsigned long y = 10;
	printf(*%lu*, vypocet(x, y));
return 0;
}
```

Zobrazí sa číslo 5.
Význam funkcie výpočet: Funkcia vypočíta najväčší spoločný deliteľ dvoch prirodzených čísel.
Význam kódu v mieste A: Koncová podmienka rekurzie, pokiaľ bude niektorá z premenných nulová, funkcia vráti nulu.
Význam kódu v mieste B: swap, alebo výmena hodnôt premenných “a“ a “b“, aby bolo pred rekurzívnym volaním zaručené, že a>b.
Význam kódu v mieste C: Rekurzívne volanie funkcie “vypocet“.

- [10 bodov]

Čo sa zobrazí po vykonaní nasledujúceho programu? Uveďte presne výsledok, ktorý sa zobrazí na štandardné výstup. Popíšte, aký význam má kód v miestach označených ako A a B.

```
#include <stdio.h>
void zobrazeni ( unsigned long long cislo, unsigned short arg_bitu ){
        if ( arg_bitu > 1 ) // Místo A
                zobrazeni ( cislo >> 1, arg_bitu - 1 ) ; // Místo B
 // operátor >> provádí bitový posuv prvníhoho operandu opočet bitů
 // (daný druhým operandem) vpravo
 // vyprázděné bity zleva jsou vyplněny nulami
        printf ( "%llu", cislo & 1 ) ;
 // operátor & provádí logický součin nad jednotlivými
 // bity operandů
 }
int main ( void ){
        unsigned short pom = 542 ;
        zobrazeni ( pom, sizeof ( pom ) * 8 ) ;
 // předpokládejte, že velikost typu short je 2 Byte
 return 0 ;

 }
```

542: 10 0001 1110
Zobrazí sa: 0000001000011110
Namiesto A: podmienka ukončenia rekurzie.
Namiesto B: rekurzívne volanie funkcie myPrint.

- [12 bodov]

Čo sa zobrazí po prevedení nasledujúceho programu? Uveďte presne výsledok, ktorý sa zobrazí na štandardný výstup.

```
 void zobrazenie(unsigned long long cislo, unsigned short arg_bitu)
{    // namiesto A
        if (arg_bitu > 2)
                zobrazenie (cislo >> 2, arg_bitu – 2);
        printf(*%llu% *cislo & 1, (cislo >> 1)&1);
}
int main (void){
        unsigned short pom = 121;
        zobrazenie (pom, sizeof (pom *4);       //namiesto B
// predpokladajte, že veľkost typu short je 2 bajty
return 0;
}
```

Zobrazí sa: 10 11 01 10

Namiesto A: hlavička funkcie zobrazení, sú špecifikované 2 formálne parametre pre predávanie hodnotou.
Namiesto B: volanie funkcie zobrazení s 2 argumentami.

- [10 bodov]

Čo sa zobrazí po prevedení nasledujúceho programu? Uveďte presne výsledok, ktorý sa zobrazí na štandardný výstup.

```
#include <stdio.h>
void myPrint (int n){
        printf ("%d", n % 2);
        if (n > 0)  // namiesto A
                myPrint (n – 1);  // namiesto B
        printf ("%d", n / 2);
        return;
}
int main (void){ int count = 4; myPrint (count); return 0; }
```

Zobrazí sa: 0101000112
Namiesto A: podmienka ukončenia rekurzie.
Namiesto B: rekurzívne volanie funkcie myPrint.

- [10 bodov]

Čo sa zobrazí po prevedení nasledujúceho programu? Uveďte presne výsledok, ktorý sa zobrazí na štandardný výstup.

```
#include <stdio.h>
void myPrint (int n){
        printf ("%d", n / 2);
        if (n > 0)  // namiesto A
                myPrint (n – 1);  // namiesto B
        printf ("%d", n );
        return;
}
int main (void){ int count = 4; myPrint (count); return 0; }
```

Zobrazí sa: 2110001234
Namiesto A: podmienka ukončenia rekurzie.
Namiesto B: rekurzívne volanie funkcie myPrint.

- [12 bodov]

Analyzujte nasledujúcu funkciu:

```
unsigned int vypocet(unsigned ind a, unsigned int b, bool c){
	if (a < b){ a = b – a; b = b – a; a = b + a; }
	unsigned int x = a, y = 0;
	//namiesto A		invariant: a >= b, x = a, y = 0
	if (b > 0)
		while (x >= b){
		// namiesto B		invariant: a >= b, b > 0, x >= b, a = x + y ´b
			x = x – b; y = y + 1;
		}
	return c ? x : y;
}
```

Hodnota funkcie pri volaní výpočet (10, 42, false): 4
Hodnota funkcie pri volaní výpočet (100, 42, false): 16
Čo funkcia vykonáva: Prevádza celočíselné delenie a zbytok po delení čísel “a“ a “b“ (váčší dleno menší). Vracia podiel alebo zbytok podľa hodnoty v parametry “c“.

- [12 bodov]

Čo sa zobrazí po prevedení nasledujúceho programu? Uveďte presne výsledok, ktorý sa zobrazí na štandardný výstup.

```
unsigned long vypocet (unsigned long a, unsigned long b){
if (a*b == 0) return 0; //namiesto A
if (a < b) {
	a^ = b; b^ = a; a^ = b; //namiesto B
}
// operátor ^ predstavuje XOR
return (a % b > 0) ? vypocet(b, a % b):b; //namiesto C
}
//operátor % je operácia modulo
int main(void){
	unsigned long x = 204; unsigned long y = 114;
	printf(*%lu*, vypocet(x, y));
return 0;
}
```

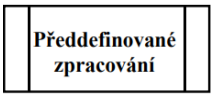
Zobrazí sa: 6.

Význam funkcie výpočet: Funkcia vypočíta najväčší spoločný deliteľ dvoch prirodzených čísel.
Význam kódu v mieste A: Koncová podmienka rekurzie, pokiaľ bude niektorá z premenných nulová, funkcia vráti nulu.
Význam kódu v mieste B: swap, alebo výmena hodnôt premenných “a“ a “b“, aby bolo pred rekurzívnym volaním zaručené, že a>b.
Význam kódu v mieste C: Rekurzívne volanie funkcie “vypocet“.

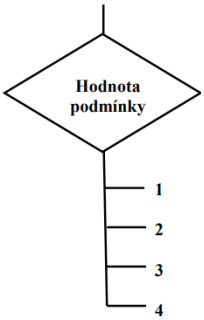
- [6 bodov]

Uveďte základnú riadiacu štruktúru v programovaní (3 štruktúry). Každú štruktúru stručne charakterizujte, pre každú štruktúru uveďte minimálne 2 príklady (príkazy), ktorými sa dá danú štruktúru v programovaní realizovať a jeden príklad symbolu, ktorým sa dá príkaz štruktúry vyjadriť vo vývojovom diagrame.

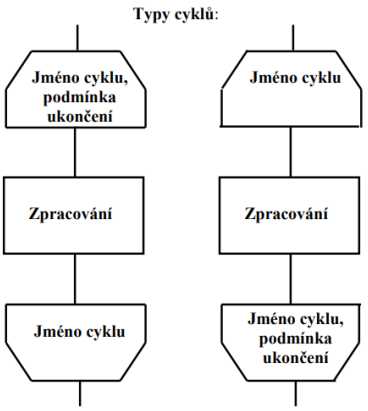
- **sekvencia** = všetky príkazy sa postupne prevedú jeden po druhom v danom poradí, napr. priradenie, volanie funkcii



- **selekcia** = v závislosti na splnenie podmienky sa určitý príkaz buď prevedie alebo neprevedie, napr. vetvenie if - then, if - then - else, switch



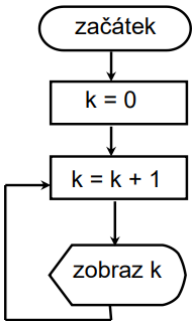
- **iterácia** = vzávislosti na splnenie podmienky sa časť programu môže vykonať viackrát, napr. cykly for, while, do - while



- [6 bodov]

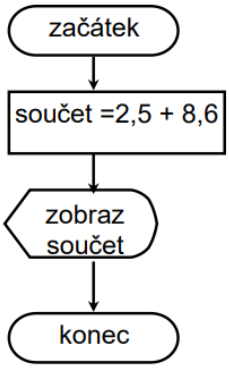
Stručne charakterizujte základné vlastnosti algoritmov (1-2 vety pre každú vlastnosť). Uveďte príklady algoritmov, ktorý nie je konečný a príklad algoritmu, ktorý nie je hromadný.

- **konečnosť (rezultatívnosť)** = má zaručiť vyriešenie úlohy po konečnom počte krokov

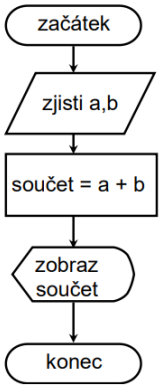


nie je konečný

- **hromadnosť** = jediným algoritmom sa dá riešiť veľkú škálu úloh rovnakého druhu

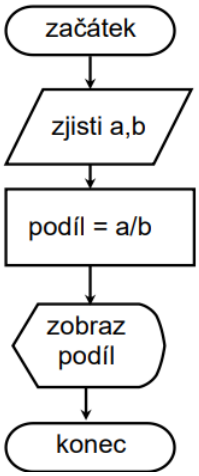


nie je hromadný



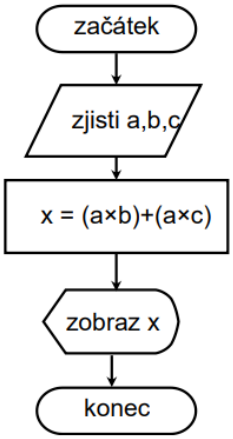
je hromadný

- **determinovanosť** = algoritmus je zadaný vo forme konečného počtu jednoznačných pravidiel



nie je determinovaný

- **efektívnosť** = na správny priebeh programu nemá žiadny vplyv, zaisťuje iba to, aby program trval čo najkratšiu dobu



nie je efektívny (efektívny by bol ak by sme urobili toto: ax(b+c))

- [2 body]

Vysvetlite, čo predstavuje inštrumentácia programov.

Inštrumentácia = dynamická analýza, pred spustením kódu je kód mierne obohatený o riadiace/kontrolné prvky (tzv. sondy) pre účel ladenia.

- program je inštrumentovaný
- program sa spustí normálnym spôsobom
- spúšťané sondy informujú monitor o aktuálnych udalostiach (napr. prístupov do pamäte).
- po skončení programu valgrind informuje súhrnom udalostí.

- [2 body]

Vysvetlite 2 základné typy analýz programov. Pri každej uveďte príklad aspoň jednej techniky ladenia programu.

- **statická analýza**: skúma zdrojové kódy bez ich spustenia, metóda code review. Napr. code review, (yellow/rubber duck debugging)
- **dynamická analýza**: skúma priebeh programu. Napr. krokovanie, ladiace výpisy/záznamy

---

- [2 body]

Je daný nasledujúci kód:

```
unsigned int a = 1, b = 2, c;
c = a & b || a && b;
```

Akú hodnotu bude mať premenná “c“ po prevedení tohto kódu?

a & b = 1 & 2
01
10
00 = 0
(a & b)||b = 0||2 = false||true = true = 1
(a & b||b) && a = 1 && 1 = true && true = 1

Premenná “c“ bude mať hodnotu 1.

- [4 body]

Aká bude hodnota premennej “sum“ po prevedení nasledujúceho kódu?

```
int sum = 1;
for  ( int i = 0; i < 10; i++) {
        switch (i) {
        case 1:  case 4:  case 7:  sum++;
        default:  continue;
        case 3:  break;
        }
break;
}
```

i = 0, sum = 1:
continue;
i = 1, sum = 1:
sum++; continue;
i = 2, sum = 2:
continue;
i = 3, sum = 2:
break;

Premenná “sum“ bude mať hodnotu 2.

- [6 bodov]

Je dané pole “a“. Akú hodnotu bude mať prvok a[10]?

a[10] = premenná c{1,2,3,[8]=8,4};

Nie je definované. Do poľa sa dá pristupovať cez a[0] až a[9].

- [6 bodov]

Aký je princíp práce so zásobníkom?

Dynamická alokácia na zásobníka = do priestoru zásobníka sa ukladajú všetky lokálne premenné a parametre funkcií. Pri každom zavolaní funkcie sa automaticky vyhradí priestor pre lokálne premenné a parametre, takže programátor sa o to nemusí vôbec starať. Táto pamäť sa automaticky uvoľní v okamihu ukončenia funkcie. Všetky lokálne premenné a parametre, ktoré sme doteraz používali, boli alokované
na zásobníku.

- [6 bodov]

Počet výplat různých od “payValue“:

```
typedef struct {
        char name[11];
        char surname[16];
        int pay;
} tdata;
typedef struct item titem;
struct item {
        tdata data;
        titem *next;
};
typedef struct {
        titem *head;
        titem *tail;
} tlist;
int countDifferentPay(const *list, int payValue);
```

```
int countDifferentPay(const titem *list, int payValue) {
  int count = 0;
  for (titem *tmp = list->head; tmp != list->tail; tmp = tmp->next)
    if(tmp->data.pay != payValue)
      count++;
  return count;
}
```

- [8 bodov]

Je dané:

```
typedef struct {
        char name[11];
        char surname[16];
        int pay;
} tdata;
typedef struct item titem;
struct item {
        tdata data;
        titem *next;
};
typedef struct {
```

```
        titem *head;
        titem *tail;
} tlist;
int countDifferentPay(const tlist *list, int payValue);
```

Definujte funkciu “countItemGreaterPay“, ktorá vracia počet položiek lineárneho zoznamu (daného prvým parametrom funkcie), v ktorých je hodnota zložky “pay“ väčšia ako hodnota daná paramerom “payValue“.

```
int countItemGreaterPay (const tlist *list, int payValue) {
  int count = 0;
  for (titem *tmp = list->head; tmp != NULL; tmp = tmp->next)
    if(tmp->data.pay != payValue)
      count++;
  return count;
}
```

- [4 body]

Princíp Eratosthénova sita a najmenšia možná zložitosť algoritmu.

Eratosthénovo sito je jednoduchý algoritmus pre nájdenie všetkých prvočísiel menších ako zadaná horná hranica.

Algoritmus funguje "preosievaním" zoznamu čísel - na začiatku zoznam obsahuje všetky čísla v danom rozsahu (2, 3, 4, ..., zadané maximum). Potom sa opakovane prvé číslo zo zoznamu vyberie, toto číslo je prvočíslom; zo zoznamu sa potom odstráni všetky násobky tohto čísla (čo sú čísla zložená). Tak sa pokračuje dovtedy, než je zo zoznamu odstránené posledné číslo (alebo vo chvíli, keď je ako prvočíslo označené číslo vyššie ako odmocnina najvyššieho čísla - v takej chvíli už všetky zostávajúce čísla sú nutne prvočíslami). Časová zložitosť tohto algoritmu je O (N * log (log N)), kde N je horná hranica rozsahu. Zložitosť je semi-logaritmická. (O(N*log*N))

- [2 body]

Je daný nasledujúci kód:

```
Char tmp1[20] = „Jupiter“;
Char tmp2[20];
Tmp2= tmp1;
Strcat(tmp1,tmps); //spojenie retazcov
```

Printf(„%s“, tmp1);

Čo kód vypíše?

Riešenie: JupiterJupiter

- [4 body]

Analyzujte nasledujúci fragment programu a za predpokladu, že 0.3<x<10.7, uveďte platné výstupné podmienky vzhľadom k premennej x pre príkazy bloku S1 a pre príkazy bloku S3:

```
float x;
// inicializácia x
// v tomto mieste platí pre x: 0.3<x<10.7
if (x < 10.0) {
// príkazy bloku S1
}
else if (x < 10.1) {
// príkazy bloku S2
}
else //príkazy bloku S3
```

Vstupná podmienka bloku S1: 0.3 < x < 10.0
Vstupná podmienka bloku S2: 10.1 <= x < 10.7

- [4 body]

Analyzujte nasledujúci fragment programu a za predpokladu, že 0.5<=x<=10.8, uveďte platné výstupné podmienky vzhľadom k premennej x pre príkazy bloku S1 a pre príkazy bloku S3:

```
float x;
// inicializácia x
// v tomto mieste platí pre x: 0.5<=x<=10.8
if (x < 10.0) {
// príkazy bloku S1
}
else if (x < 10.5) {
// príkazy bloku S2
}
else //príkazy bloku S3
```

Vstupná podmienka bloku S1: 0.5 < x < 10.0
Vstupná podmienka bloku S2: 10.5 < x <= 10.8

- [4 body]

Analyzujte nasledujúci fragment programu a za predpokladu, že -10.0 <= x <= 2.0, uveďte platné výstupné podmienky vzhľadom k premennej x pre príkazy bloku S1 a pre príkazy bloku S2:

```
float x;
// inicializácia x
// v tomto mieste platí pre x: -10.0 <= x <= 2.0
if (x <= 0.0) {
// príkazy bloku S1
}
else if (x >= 2.0) {
// príkazy bloku S2
}
```

Vstupná podmienka bloku S1: -10.0 <= x <= 0.0
Vstupná podmienka bloku S2: x = 2.0

- [3 body]

Stručne vysvetlite (2-3 vety) princíp manipulácie s dátami abstraktného dátového typu zásobník (LIFO).

Pre zásobník platí, že dáta uložené do zásobníku ako posledné, budú čítané, ako prvé. Pre manipuláciu s uloženými dátovými položkami sa udržuje tzv. ukazovateľ zásobníku, ktorý udáva relatívnu adresu poslednej pridanej položky, tzv. vrchol zásobníku.

- [10 bodov]

Je dané:

```
typedef struct {char name[11]; char surname[16]; int pay;} tdata;
typedef struct item titem;
struct item {tdata data; titem *next;};
typedef struct {titem *head; titem *tail;} tlist;
```

Definujte funkciu “countItemDifferentPay“, ktorá vracia počet položiek zoznamu (daného prvým parametrom funkcie), v ktorej je hodnota

zložky “pay“ rôzna od hodnoty danej parametrom “payValue“. Predpokladajte, že zoznam obsahuje minimálne jednu položku.

```
int countItemDifferentPay (const tlist *list, int payValue){
        int count = 0; // inicializácia počítadla
        for (titem *tmp = list->head; tmp != NULL; tmp = tmp->next)
                if (tmp=>data.pay != payValue)
                        count++;
        return count;
}
```

- [6 bodov]

Stručne vysvetlite základné vlastnosti metód riadenia (1-2 vety pre každú vlastnosť):

- **sekvenčnosť** = je vlastnosť, ktorá vyjadruje že riadiaci algoritmus pracuje so vstupnými údajmi a aj s dátovými medziproduktami v tom poradí, v akom sú lineárne usporiadané v dátovej štruktúre.
- **časová zložitosť** = označuje mieru času potrebnú na realizáciu daného algoritmu. Pri radiacich algoritmoch ju označujeme, ako funkciu počtu n radených prvkov.
- **priestorová** zložitosť = označuje mieru priestoru potrebnú k realizácii daného algoritmu. Pri radiacich algoritmoch ju označujeme, ako funkciu počtu n radených prvkov.
- **prirodzenosť** = je vlastnosť algoritmu, ktorá vyjadruje, že doba potrebná k radeniu jej zoradenej množiny údajov je menšia, ako doba pre zoradenie náhodne usporiadanej množiny a tá je menšia než doba pre zoradenie opačne zoradenej množiny údajov.
- **stabilita** = je vlastnosť radenia, ktorá vyjadruje, že algoritmus zachováva vzájomné poradie údaje so zhodnými kľúčmi.
- **in situ** = metóda pracuje in situ znamená, že metóda nepožaduje výrazne väčší priestor pre radenie, než zaberá pôvodná zoradená štruktúra

---

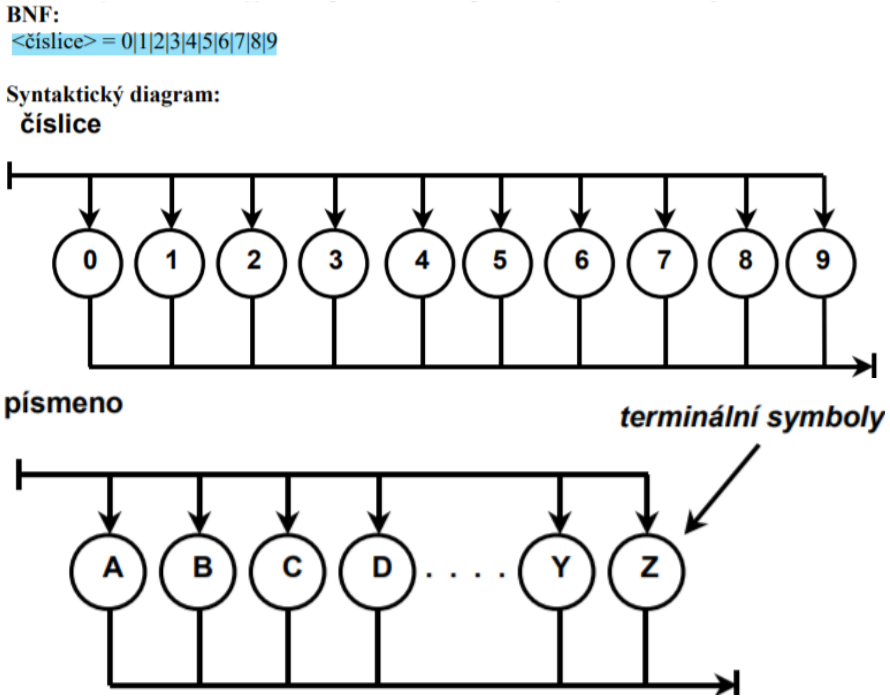
- [2 body]

Vysvetlite pojmy syntax a sémantika programovacích jazykov.

- **syntax** = súbor pravidiel udávajúcich možné konštrukcie programu. Popisuje formálnu štruktúru programu. Definuje kľúčové slová, identifikátory, čísla a ďalšie programové entity a určuje spôsob, ako ich možno kombinovať. Na základe syntaktických pravidiel je možné posúdiť, či určitý text je alebo nie je korektným zápisom programu v danom jazyku.
- **syntax číslice/písmena =** BNF(Backus Naurova Forma)

BNF:
<číslice> = 0|1|2|3|4|5|6|7|8|9

Syntaktický diagram:



- **sémantika** = určuje logický význam jednotlivých výrazov jazyka. Zo sémantiky vyplýva, aký má daná konštrukcia význam.

- [2 body]

Definujte pojem zarážka pri sekvenčnom vyhľadávaní v poli so zarážkou. Popíšte v čom spočíva jej výhoda.

- **zarážka** = je hľadaná položka umiestnená na konci poľa. Potom pri priechode poľom pri prehľadávaní nie je nutné opakovane testovať na koniec poľa (ukončenie poľa zaisťuje zarážka). Vďaka tejto úprave sa dá algoritmus zrýchliť.

- [2 body]

Čo je výsledkom aplikácie operátoru sizeof?

Výsledkom aplikácie operátoru sizeof je veľkosť jeho operandu v bytoch.

- [2 body]

Popíšte výhody a nevýhody testovania a formálnej verifikácie programov

- **testovanie** = + : rýchle, jednoduché, výpočetne nenáročné; - : nepotvrdí absenciu chyby
- **formálna verifikácia** = + : potvrdí (ne-)prítomnosť chyby (vzhľadom na špecifikáciu programu), - : výpočetne náročné

- [2 body]

Je daná definícia:

int mat[n][n]

Akú postupnosť časovej zložitosti má algoritmus určený na výpis všetkých hodnôt po obvode štvorcovej matice usporiadania n , uloženej v poli "mat"? Čím je daná veľkosť vstupných dát?

Usporiadanie zložitosti: lineárne
Veľkosť dát je daná: 4(n-1)

- [2 body]

Je daný nasledujúci kód:

```
float f = 0;
for (int i = 0; i < 10; i++) f = f + 0.1;
if (f == 1.0f);   //XX
```

Vysvetlite prečo podmienka na riadku označeným XX nemusí byť splnená.

Typ float je iba aproximáciou reálnych čísel. 10x prevedené sčítanie hodnôt 0.1 sa iba približuje hodnote 1.0, ostrá rovnosť v podmienke (na riadku označenom XX) teda nemusí platiť.

- [2 body]

Je daný nasledujúci kód:

```
float f = 0.0f;
for (int i = 0; i < 10; i++) f = f + 0.1;
if (f == 1.0f);   printf("výsledok je %6.4f", f);   //XX
```

Čo by sa zobrazilo po jeho prevedení? Odpoveď zdôvodnite.

Typ float je iba aproximáciou reálnych čísel. 10x prevedené sčítanie hodnôt 0.1 sa iba približuje hodnote 1.0, ostrá rovnosť v podmienke (na riadku označenom XX) teda nemusí platiť.

- [4 body]

Je daný nasledujúci kód:

```
void test (int a, int b){
        if (a = b)
                printf(“a je rovné b\n“);       //XX
        else
                printf(“a nie je rovné b\n“);
}
```

Aká by musela platiť vstupná podmienka (pre parametre) tejto funkcie, aby sa vždy vykonal príkaz na riadku označenom XX?

riešenie: b = 0

- [4 body]

Uveďte príklady dátových štruktúr. Pri každej štruktúry uveďte deklaráciu premennej pre uvedený typ dátovej štruktúry.

- **homogénne**: pole, napr.
  int pole [10];
- **heterogénne**: záznam, napr.
  struct miera {int vyska; float vaha;} jana, pavel;
- **statická**: pole, napr.
  float teplota [10][12][31];
- **dynamická**: lineárny zoznam, napr.
  typedef struct { char surname[15]; int pay;}tdata;
  typedef struct item titem;
  struct item {tdata data; titem *next;};
  typedef struct { titem *head; } tlist
  tlist list;

- [4 body]

Uveďte minimálne 4 výrazy, ktoré sa budú líšiť druhom operátorov. Nie je nutné deklarovať použité premenné.

Výraz s unárnym a binárnym operátorom: ++i * a / b
Výraz s logickým operátorom: a && b || c
Výraz s ternárnym operátorom: (a > b)? a : b
Výraz s relačným operátorom: x < z

- [4 body]

Čím sa líšia unárne, binárne a ternárne operátory? Ku každému druhu uveďte príklad (stačí jeden z každého druhu) a stručný popis operátoru jazyka C.

Čím sa líšia: Rozdiel je v počte operandov (unárny 1, binárny 2, terárny 3).

- **unárny**: - negácia, ~ inverzia, sizeof() počet bajtov výrazu alebo typu,...
- **binárny**: + súčet, == ekvivalencia, = priradenie, „ , “ čiarka odovzdanie hodnoty...
- **ternárny**: c ?a :b – podmienený výraz, kde výsledná hodnota (a alebo b) závisí na hodnote c

---

- [6 bodov]

Je dané:

int x;

Napíšte výraz, ktorý nadobúda hodnoty true, keď hodnota uložená v premennej x nie je deliteľná žiadnym číslom z množiny {3, 5, 7} (použite iba multiplikatívne a logické operátory!). Nepoužívajte žiadne príkazy (ani if – else).

((x % 3 != 0) && (x % 5 != 0) && (x % 7 != 0))
alebo
((x % 3) && (x % 5) && (x % 7))

- [6 bodov]

Je daná definícia:

unsigned int r; //inicializácia r

Napíšte výraz vyjadrujúci či je rok uložený v premennej “r“ prestupný (true) alebo nie je prestupný (false). Prestupné roky, podľa Gregoriánskeho kalendára, sú roky deliteľné 4, avšak roky deliteľné 100 sú prestupné len vtedy, pokiaľ sú deliteľné aj 400 (nepoužívajte ternárny operátor).

```
((r % 4 == 0) && ((r % 100 > 0) || (r % 400 == 0)))
alebo
(!(r % 4) &&((r % 100) || !(rok % 400)))
```