Структури от данни и приложни алгоритми

Лекции 2 ч.

Лабораторни упражнения 1 ч.

Курсова работа

Изпит

Лектор: **Доц. д-р Мариана Горанова**
Катедра „Програмиране и компютърни технологии“, ФКСУ
Технически университет – София
Кабинет: 2302
Електронен адрес: mgor@tu-sofia.bg
URL: pct.tu-sofia.bg/Moodle001

Литература

1. Робърт Седжуик, Алгоритми на C, СофтПрес, София, 2002.
2. П. Наков, П. Добриков, Програмиране=++Алгоритми;, Top Team Co, София, 2002.
3. П. Наков, Основи на компютърните алгоритми, Top Team Co, София, 1998.
4. Т. М. Смит, Програмиране с PASCAL. Принципи и методи, Техника, 1996.
5. Robert Sedgewick, Algorithms in C++, Addison-Wesley Publishing Company, Inc., 1992.
6. Никлаус Вирт, Алгоритми+структури от данни=програми, Техника, София, 1980.

Структури от данни

Определение

информация = данни + съдържание

Структури данни са организирана информация, която може да се опише, създаде и обработи от програмите.

Изразяват съдържанието на конкретното представяне на информацията на физически носители (последователност от битове).

Фундаментални структури данни – базовите типове данни в езиците за програмиране:

1. **Целочислен тип.**
2. **Реален тип.**
3. **Символен тип.**
4. **Низ.**
5. **Масив.**

Структура и организация на работа с данни: функционална спецификация, логическо описание и физическо представяне

Определение

Структури от данни (СД) се наричат набор от едно или няколко имена и множества от данни, към които се осъществява достъп чрез тези имена.

СД е абстрактен обект (едно или няколко имена) и физически обект (величина).

Връзката между абстрактния и физическия обект се осъществява чрез декомпозиране на СД в термините на по-елементарни СД.

Нива на описание на СД

1. Функционална спецификация
2. Логическо описание
3. Физическо представяне

Функционална спецификация – разглежда СД като абстрактен обект (без конкретна реализация).

1. **Нов тип T.**
2. **Операции.**

   $f: X_1 \times X_2 \times \ldots X_n \rightarrow Y_1 X Y_2 X \ldots \times Y_m$

   X_i и Y_i са типове, като поне един е типът T.

   Видове операции:

   а) за достъп – T се явява само от лявата страна на стрелката;
   – връща значение, характеризиращо обектите от тип T;

   б) за модификация – T се явява и от двете страни – създава нов обект от тип T въз основа на обект от тип T и допълнителни елементи;

в) за създаване – T се явява само от дясната страна – създава елемент от тип T въз основа на елементи от други типове или без други типове (наляво от стрелката няма типове).

3. Свойства – за всеки обект t от тип T.

Логическо описание

1. **Декомпозиция на обектите от функционалното определение на по-елементарни обекти.**

   а) съединение (;) – даден тип T се определя чрез съединяване на други типове T_i;

   тип T = (име$_1$: T_1; ....; име$_n$: T_n)

   б) разделяне на вариантите (I) – даден тип T се определя чрез разделяне на варианти и се използва оператор избери и отношение e;

   тип T = (T_1 I T_2 I ... I T_n)

   избери

   | $f_1(t)$ e T_1: действие$_1$,
   | ...
   | $f_n(t)$ e T_n: действие$_n$

в) изброяване (,) – частен случай на б) – даден тип T се определя чрез краен брой възможни решения, зададени като списък от константи $конст_1$, $конст_2$, …, $конст_n$**.**

тип T = ($конст_1$, $конст_2$, …, $конст_n$)

2. **Декомпозиция на операциите на по-елементарни операции.**

   <u>Пълно логическо описание</u> – операциите на функционалното описание се описват като функции, които работят с обекти от тип T и проверяват свойствата на описанието.

Рекурсивно определение

Рекурсия е наличие в определението на обект от същия тип (директна рекурсия) или обект от друг тип, който от своя страна е определен от типа на дадения обект (косвена рекурсия).

При определение на типа T чрез съединение една компонента може да бъде от тип T или от друг тип T_i, който от своя страна е определен чрез типа T (броят на нивата е произволен).

тип T = (име$_1$: T_1; ....; име$_i$: T; име$_n$: T_n)

Логическото описание на СД включва:

1. **Определяне на типа чрез включване на други типове и тип, който сам се определя (рекурсия), с използване на операциите съединение, разделяне на вариантите и изброяване.**
2. **Функции, които оперират върху определения тип.**

Физическо представяне

1. **Метод за разполагане на величините в паметта на компютърната система.**

а) **разделяне на вариантите**

тип T = (T_1 I T_2 I ... I T_n)

Обектът от тип T **се представя чрез два елемента:**

- **индикатор на типа** i: ЦЯЛО, $1 \le i \le n$, **показва на кой тип** T_i **принадлежи обектът; представя се чрез** $[\log_2 n]$ **бита;**
- **обект от тип** T_i**; представя се чрез необходимата памет за типа с максимален размер.**

[x] **– най-малкото цяло число, по-голямо или равно на** x**.**

б) изброяване

тип T = (конст$_1$, конст$_2$, ..., конст$_n$)

Обектът от тип T се представя чрез:

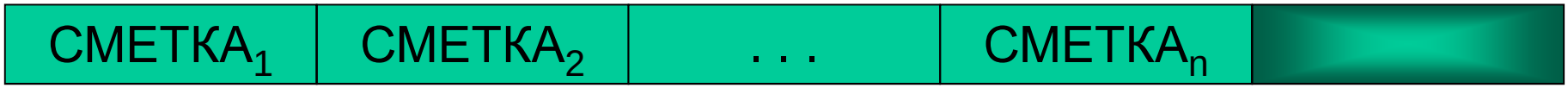
- **индикатор** i: ЦЯЛО**,** $1 \le i \le n$**;**
  представя се чрез $[\log_2 n]$ **бита;**

в) съединение

тип T = (име$_1$: T$_1$; ....; име$_n$: T$_n$)

Обектът от тип T се представя чрез сума от размерите на съответните обекти от тип T$_i$.

- **масив – паметта може да се препълни или се резервира безполезно голямо количество;**



– **списък от свързани информационни блокове с фиксиран размер (всеки блок съдържа данни за следващия елемент в списъка, т.е неговия адрес);**
краят на списъка се представя чрез специален тип ПРАЗНО**.**

2. **Начин за кодиране на операциите чрез език за програмиране.**

Видове класификации

I. Статични структури от данни
   1. Масив
   2. Вектор
   3. Структура
   4. Изброяване
   5. Файл

II. Динамични структури от данни
   1. Списък
   2. Стек
   3. Опашка
   4. Дек
   5. Граф
   6. Дърво
   7. Хеш-таблица

Основни типове данни

I. Числови данни

1. Цели числа

Тип	Размер [байта]	Област
unsigned short	2	0÷65535
short int	2	-32768÷32767
unsigned int	1 дума=4 байта	0÷4294967295
int	1 дума=4 байта	-2147483648÷ 2147483647
unsigned long	4	0÷4294967295
long int	4	-2147483648÷ 2147483647

Проблеми:

а) целочислено деление – промяна на реда в израза;

```
printf ("%d\n", 10*(7/10));          // 0
printf ("%d\n", (10*7)/10);          // 7
```

б) целочислено препълване – при умножение и събиране.

```
int a = 110000, b = 20000, product;
product = a * b / 20000;
printf ("%d\n", product);          // -104748
```

110,000 * 20,000 = 2,200,000,000 > max = 2,147,483,647
(-2,147,483,648 + 2,200,000,000) / 20,000 =
-2,094,967,295 / 20,000 = -104,748

2. Числа с плаваща точка

Тип	Размер [байта]	Област	Точност
float	1 дума=4 байта	+/-3.4E38	7 знака
double	2 думи=8 байта	+/-1.7E308	15 знака
long double	10 байта	+/-1.2E4932	

Проблеми:

а) събиране и изваждане на числа с големи разлики в стойностите – сортиране на числата и извършване на действията от най-малките стойности;

```
float f = 1000000.00f + 0.1f;
printf ("%f\n", f);                    // 1000000.125000

float g = 5000000.02f - 5000000.01f;
printf ("%f\n", g);                    // 0.000000
```

б) сравняване за равенство – проверка за достатъчна близост на стойностите

```
float key = 1.0f;
float sum = 0.0f;
for (int i = 0; i < 10; i++)
   sum += 0.1;
if (key == sum)
   printf ("Числата са равни.\n");
else
   printf ("Числата са различни.\n");
// Числата са различни
```

Проверка за достатъчна близост на стойностите

```
#include <math.h>
#define DELTA 0.0001f          // точност

int equals (float x, float y)
{  if (fabs (x - y) < DELTA)
       return 1;                // достатъчно близки числа
   else
       return 0;                // недостатъчно близки числа
}

if (equals (key, sum))
   printf ("Числата са равни.\n");
else
   printf ("Числата са различни.\n");

// Числата са равни
```

в) грешки от закръгляване – същите като а)

– промяна на типа към тип с по-голяма точност – float $\Rightarrow$ double**;**

– промяна към двоично кодирани променливи;

– промяна от плаваща точка към целочислени променливи – левове $\Rightarrow$ стотинки

г) използване на специални типове данни – Currency **в C#**

II. Символи

Тип	Размер [байта]	Област
char	1	-128÷127
signed char	1	-128÷127
unsigned char	1	0÷255

Проблеми:

Символният литерал е криптиран – използва се константа вместо конкретна стойност на символа

```
// Сравнение за символ ESC (^[ с код 0x1B)
#define ESCAPE 0x1B
char inputChar;
…
if (inputChar == ESCAPE) { …}
```

III. Логически тип

С не разполага с логически тип.
Логическите променливи документират програмата и опростяват сложни проверки.

```
// Създаване на собствен логически тип

typedef int BOOLEAN;
...
BOOLEAN finished;
finished = (index < 0) || (DIM < index);
```

IV. Изброим тип

Данни с изброими стойности е подредено крайно множество от константни стойности, които могат да приемат променливите от този тип. Чрез изброимите типове програмите се четат по-лесно.

Дефиниране

enum идентификатор {списък_от_константни_стойности};

Пример:

```
enum Month {January, February, March, April, May, June, July,
            August, September, October, November, December};
enum Counter {Start, First, Second, Tenth = 10, Eleventh};
enum Boolean {True = 1, False = (!True)};
```

V. Указател

Указател е променлива, чиято стойност е адрес от паметта на компютъра. Осъществява се косвен достъп до стойността, записана на този адрес, за разлика от директния достъп чрез името на променливата.

Дефиниране

тип *идентификатор;

идентификатор – име на указателя;
тип – типа на данните, към които сочи указателят.
NULL – нулева стойност за инициализиране на указател.

Операции

а) адресна операция & – връща адреса на своя операнд;

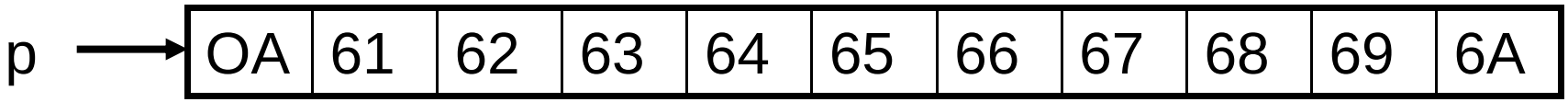
&променлива

б) извличане на стойност * – връща стойността на променливата, разположена на указания адрес.

*указател

Концептуално указателят се състои от две части:

1) **адрес (място в паметта) – 1 дума;**
2) **интерпретация на съдържанието на паметта на този адрес – зависи от базовия тип, към който сочи указателят.**



char *p; | 0A | linefeed

int *p; | 0A | 61 | 62 | 63 | 1667391754

float *p; 4.175956562029 80E+21

short int *p; | 0A | 61 | 24842

Проблеми:

а) използване на void * **– принудително преобразуване до типа на указателя;**

```
link = (NODE *)calloc (1, sizeof (NODE));
```

б) отделяне на памет – чрез sizeof ()**;**

в) връщане на резултат от функция – правило на звездичката: аргумент на функция се предава обратно, ако има * пред аргумента в оператора за присвояване =.

```
// предаване на параметър
void method (int *parameter)
{
   *parameter = SOME_VALUE;
}
```

VI. Масив

Масив е най-фундаменталната структура от данни.

- състои се от елементи от един и същ (**базов**) тип – хомогенна структура;
- структура с пряк достъп – елементите са достъпни чрез **индекс**;
- заема **непрекъсната област** от паметта на компютъра.

Дефиниране – едномерен масив

тип идентификатор [размер];

размер – константен_израз

Памет

общ_брой_байтове = sizeof (тип) * размер

Достъп до елемент на масив

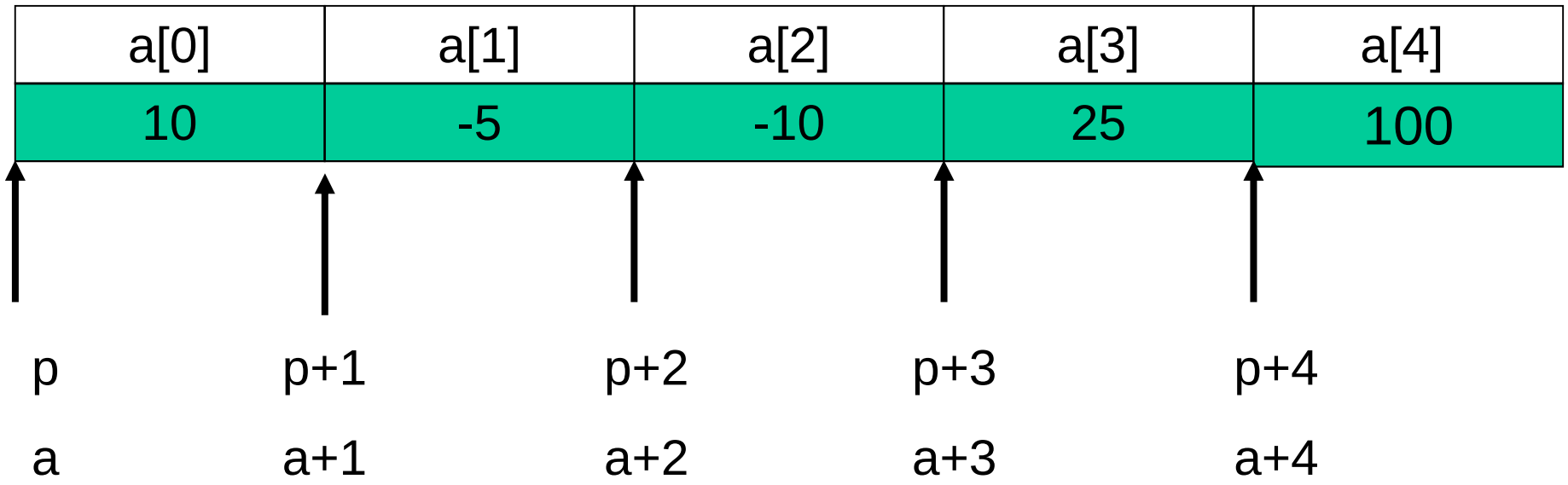
– чрез операция индексиране (директен достъп)

идентификатор [константен_израз]

– чрез операция намиране на стойност по адрес (косвен достъп)

*(идентификатор + константен_израз)

```
int a[5], *p;
p = a;                  /* указателят p сочи към първия елемент  */
a[3] = 25;              /* достъп чрез индекс                    */
*(p+3) = 25;            /* достъп чрез адресна аритметика        */
*(a+3) = 25;            /* достъп чрез адресна аритметика        */
p[3] = 25;              /* указател се индексира като масив      */
```



Дефиниране – многомерен масив

тип идентификатор
    [константен_израз$_1$] […] [константен_израз$_N$];

Памет

общ_брой_байтове = брой_редове*брой_стълбове*sizeof(тип)

Достъп до елемент на масив

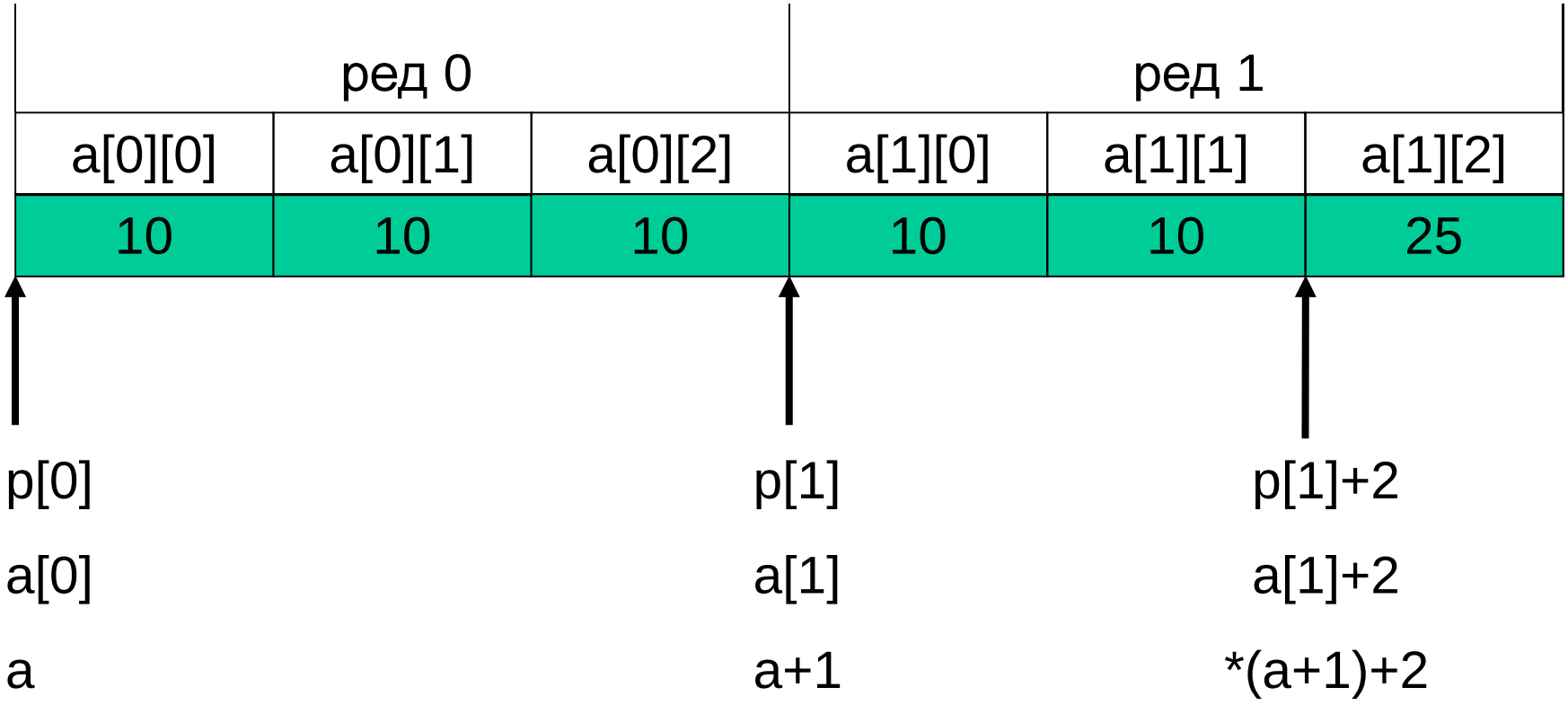
– чрез операция индексиране (директен достъп)

идентификатор [константен_израз$_1$][константен_израз$_2$]

– чрез операция намиране на стойност по адрес (косвен достъп)

((идентификатор + константен_израз$_1$)+константен_израз$_2$)

```
int a[2][3], *p[2];
p[0] = &a[0][0];        /* указателят p[0] сочи към първия ред  */
p[1] = &a[1][0];        /* указателят p[1] сочи към втория ред  */
a[1][2] = 25;           /* достъп чрез индекси                  */
*(a[1]+2) = 25;         /* достъп чрез адресна аритметика       */
*(*(a+1)+2) = 25;       /* достъп чрез адресна аритметика       */
*(p[1]+2) = 25;         /* достъп чрез адресна аритметика       */
```



Проблеми:

а) директен достъп до елемент на масив извън границите на масива – за масиви без определени размери се използва макросът ARRAY_LENGTH:

```
#define ARRAY_LENGTH(x)(sizeof(x)/sizeof(x[0]))
int points[] = {56, 60, 70, 83, 90};
for (int ind = 0; ind < ARRAY_LENGTH (points); ind++)
   ...
```

VII. Низ

Низ е масив от тип char**, който завършва с нулевия символ** '\0'**.**

```
char s[11];          // s може да съхрани 10 символа
#define MSG "Тестване на програма"     // низова константа
```

Проблеми:

а) разлика между указател към низ и масив от символи

```
char *stringPtr;
stringPtr = "Тестване на програма";
```

Операцията присвояване насочва указателя stringPtr **към текстовия низ** "Тестване на програма“**, но не копира съдържанието му в** stringPtr**.**
Операции с низове се извършват чрез strcpy()**,** strcmp()**,** strlen() **и т.н.**

б) деклариране на низ с дължина КОНСТАНТА+1 (\0 за край на низа)

```
#define NAME_LENGTH 30

// Деклариране на низ с дължина NAME_LENGHT+1.
// Инициализиране на низа с нула при деклариране,
// за да не се получи безкраен низ.
char name[NAME_LENGTH+1] = {0};

// Инициализиране на символите в низа, като се
// използва NAME_LENGTH.
for (int i =0; i < NAME_LENGTH; i++)
   name[i] = 'A';

// Копиране на друг низ otherName в name с
// максимална дължина за копиране NAME_LENGTH
// чрез strncpy.
strncpy (name, otherName, NAME_LENGTH);
```

VIII.Структура

Структурата е група от данни (елементи) от различен тип с общо име (идентификатор).

Дефиниране

```
struct идентификатор
{
    тип_на_елемент идентификатор_на_елемент;
    . . .
};
```

Памет

общ_брой_байтове = Σ sizeof (тип_на_елемент$_i$)

Достъп до елемент на структура

– чрез операция точка .

променлива_структура.идентификатор_елемент

– чрез указателна операция ->

указател_към_структура->идентификатор_елемент

Масив от структури

```
struct идентификатор_на_структура
        идентификатор_на_масив [израз];
```

Достъп до елемент на структура в масив

– чрез операция точка .

идентификатор_на_масив[индекс].идентификатор_елемент

– чрез указателна операция ->

*(идентификатор_на_масив+индекс)->идентификатор_елемент

Предимства:

а) изяснява отношенията между данните;

Неструктурирани променливи	Структурирани променливи

int facN;
char name [30];
float uspeh;
facN = 123456;
strcpy (name,
"Иван Георгиев Петров");
uspeh = 5.35;
	struct student
	{ int facN;
	char name [30];
	float uspeh;
	};
	struct student studentFKSU;
	studentFKSU.facN = 123456;
	strcpy (studentFKSU.name,
	"Иван Георгиев Петров");
	studentFKSU.uspeh = 5.35;

б) опростява операциите за група от данни;

Копиране на група от данни	

newFacN = oldFacN;
newName = oldName;
newUspeh = oldUspeh;
	struct student newStudent,
	oldStudent;
	newStudent = oldStudent;

Размяна на две групи от данни	

previousOldFacN=oldFacN;
previousOldName=oldName;
previousOldUspeh=oldUspeh;
oldFacN=newFacN;
oldName=newName;
oldUspeh=newUspeh;
newFacN=previousOldFacN;
newName=previousOldName;
newUspeh=previousOldUspeh;
	struct student newStudent,
	oldStudent,
	previousOldStudent;
	previousOldStudent=oldStudent;
	oldStudent=newStudent;
	newStudent=previousOldStudent;

в) опростява списъка с параметри на функция.

```
// неструктурирани променливи
void printStudent (int facN, char *name, float uspeh);
…
printStudent (facN, name, uspeh);

// използване на структура
void printStudent (struct student s);
…
 printStudent (studentFKSU);
```

Добавянето на нов елемент към структурата (напр. номер на група) не води до промяна на функцията (printStudent**).**

Пример:

```
struct student
{  int facN;
   char name [30];
   float uspeh;
};
struct student list[35], s;
s.facN = 123456;
strcpy (s.name, "Иван Георгиев Петров");
s.uspeh = 5.35;
/* struct student s = {123456, "Иван Георгиев Петров", 5.35}; */
list[0] = s;
list[1].facN = 2345678;
strcpy(list[1].name, "Ана Христова Андреева");
list[1].uspeh = 4.85;
*(list+2)->facN =3456789;
strcpy(*(list+2)->name, "Димитър Александров Димитров");
*(list+2)->uspeh = 5.85;
```

IX. Собствени типове typedef

typedef стар_тип нов_тип;

стар_тип – допустим тип;
нов_тип – ново име на типа.

Модификациите се извършват по-лесно.

```
typedef int BOOLEAN;
typedef  struct student   STUDENT;
typedef double VECTOR[100];
```

Пример: **Сметка на вложител в банка**

Функционална спецификация

1. **Тип** СМЕТКА
2. **Операции**
   - фамилия_вложител: СМЕТКА → НИЗ

     Операция за достъп – връща фамилното име на притежателя на СМЕТКА**.**
   - остатък: СМЕТКА → ЦЯЛО

     Операция за достъп – връща остатъка от СМЕТКА **(сумата в сметката се изразява в стотинки).**
   - кредитор: СМЕТКА → ЛОГ

     Операция за достъп – показва дали притежателят на СМЕТКА **се явява кредитор.**

– разход: СМЕТКА x ЦЯЛО → СМЕТКА

Операция за модификация – връща нова СМЕТКА **при теглене на положителна сума.**

– приход: СМЕТКА x ЦЯЛО → СМЕТКА

Операция за модификация – връща нова СМЕТКА **при внасяне на положителна сума.**

3. Свойства на операциите

с: СМЕТКА
x: ЦЯЛО
кредитор (с) <=> (остатък (с) >= 0)
остатък (разход (с,x)) = остатък (с) - x
остатък (приход (с,x)) = остатък (с) + x
фамилия_вложител (разход (с,x)) =
фамилия_вложител (с)

Логическо описание

1. Декомпозиция на тип СМЕТКА

СМЕТКА =
- номер: ЦЯЛО НАТ
- баланс: ЦЯЛО
- вложител:
  - фамилия: НИЗ
  - възраст: ЦЯЛО НАТ
  - адрес: НИЗ
- дата_откриване:
  - ден: ЦЯЛО НАТ
  - месец: ("януари",...,"декември")
  - година: ЦЯЛО НАТ

```
тип СМЕТКА =
     (номер: ЦЯЛО НАТ;
      баланс: ЦЯЛО;
      вложител:  (фамилия: НИЗ;
                  възраст: ЦЯЛО НАТ;
                  адрес: НИЗ);
      дата_откриване:  (ден: ЦЯЛО НАТ;
                        месец: ("януари",...,"декември");
                        година: ЦЯЛО НАТ)
     )
```

Междинни типове на тип СМЕТКА

```
тип ЛИЧНОСТ = (фамилия: НИЗ;
               възраст: ЦЯЛО НАТ;
               адрес: НИЗ);

тип МЕСЕЦ = ("януари","февруари","март","април","май",
             "юни","юли","август","септември","октомври",
             "ноември","декември");

тип ДАТА = (ден: ЦЯЛО НАТ;
            месец: МЕСЕЦ;
            година: ЦЯЛО НАТ)

тип СМЕТКА = (номер: ЦЯЛО НАТ;
              баланс: ЦЯЛО;
              вложител: ЛИЧНОСТ;
              дата_откриване: ДАТА);
```

Ако за вложител **използваме разделяне на вариантите:**

```
тип ФИРМА = (название: НИЗ;
             капитал: ЦЯЛО НАТ;
             адрес: НИЗ);

тип КЛИЕНТ = (ЛИЧНОСТ I ФИРМА);

тип СМЕТКА = (номер: ЦЯЛО НАТ;
              баланс: ЦЯЛО;
              вложител: КЛИЕНТ;
              дата_откриване: ДАТА);

    избери
      |вложител(с) е ЛИЧНОСТ: действие1,
      |вложител(с) е ФИРМА: действиеn
```

Обекти от тип СМЕТКА**:**

променливи acc, сметка_Иванов: СМЕТКА,
x: ЛИЧНОСТ,
y: КЛИЕНТ;

Константи:

КЛИЕНТ (ЛИЧНОСТ ("Иванов", 250, "Студентски град бл.1"))

ДАТА (18, "декември", 2008)

Оператор за присвояване ←

x ← ЛИЧНОСТ ("Иванов", 250, "Студентски град бл.1");

y ← КЛИЕНТ(x);

acc ← СМЕТКА (11111,1000, y, ДАТА (18, "декември", 2008))

2. Декомпозиция на операциите на по-елементарни операции.

```
функция откриване_сметка: СМЕТКА
  (аргументи n: ЦЯЛО НАТ, p: КЛИЕНТ, d: ДАТА,
                v: ЦЯЛО НАТ)
  I откриване_сметка ← СМЕТКА (n, v, p, d)

функция приход: СМЕТКА
  (аргументи acc: СМЕТКА, x: ЦЯЛО)
  | приход ← acc;
  | баланс (приход) ← баланс (приход) + x

функция разход: СМЕТКА
  (аргументи acc: СМЕТКА, x: ЦЯЛО)
  I разход ← приход (acc, -x)
```

```
функция име_вложител: НИЗ (аргумент acc: СМЕТКА)
  | избери
  |   | вложител(acc) е ЛИЧНОСТ:
  |   |     име_вложител ← фамилия(вложител(acc)),
  |   | вложител(acc) е ФИРМА:
  |   |     име_вложител ← название(вложител(acc))

функция остатък: ЦЯЛО (аргумент acc: СМЕТКА)
  | остатък ← баланс (acc)

функция кредитор: ЛОГ (аргумент acc: СМЕТКА)
  | кредитор ← (остатък (acc)) >= 0
```

Ако за типа СМЕТКА **добавим компонента** гаранция **(друга сметка), то се получава рекурсивно определение:**

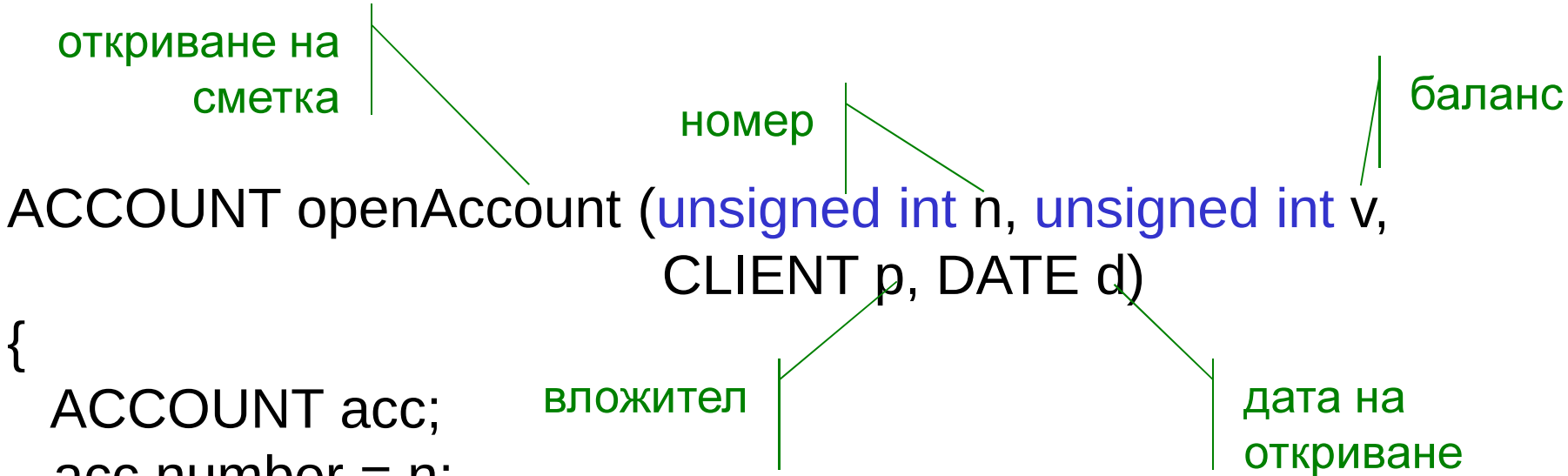
```
тип СМЕТКА = (номер: ЦЯЛО НАТ;
              баланс: ЦЯЛО;
              вложител: КЛИЕНТ;
              гаранция: СМЕТКА;
              дата_откриване: ДАТА);

променлива acc: СМЕТКА;
гаранция (acc) е СМЕТКА
дата_откриване (гаранция (acc)) е ДАТА
година (дата_откриване (гаранция (acc))) е ЦЯЛО НАТ
```

Пример: **Структура данни** СМЕТКА

```
struct person                          // тип ЛИЧНОСТ
{
   char family[10];                    // фамилия
   unsigned int age;                   // възраст
   char address[20];                   // адрес
};
typedef struct person PERSON;
struct firm                            // тип ФИРМА
{
   char title[20];                     // название
   unsigned int capital;               // капитал
   char address[20];                   // адрес
};
typedef struct firm FIRM;
```

```
struct client                          // тип КЛИЕНТ=(ЛИЧНОСТ|ФИРМА)
{
   int indicator;                      // индикатор на варианта 0/1
   void *personOrFirm;                 // указател към ЛИЧНОСТ/ФИРМА
};
typedef struct client CLIENT;
enum month {January, February, March,          // тип МЕСЕЦ
   April, May, June, July, August, September,
   October, November, December};
typedef enum month MONTH;
struct date                                    // тип ДАТА
{
   unsigned int day;                           // ден
   MONTH month;                                // месец
   unsigned int year;                          // година
};
typedef struct date DATE;
```

```
struct account                                  // тип СМЕТКА
{
   unsigned int number;                         // номер
   int balance;                                 // баланс
   CLIENT depositor;                            // вложител
   DATE open;                                   // дата на откриване
};
typedef struct account ACCOUNT;

ACCOUNT openAccount (unsigned int n, unsigned int v,
                                    CLIENT p, DATE d);

ACCOUNT income (ACCOUNT acc, int x);

ACCOUNT expenditure(ACCOUNT acc, int x);

char *depositorName (ACCOUNT acc);

int remainder (ACCOUNT acc);
int creditor (ACCOUNT acc);
```


откриване на сметка
номер
баланс
ACCOUNT openAccount (unsigned int n, unsigned int v,
CLIENT p, DATE d)
{
ACCOUNT acc;
acc.number = n;
acc.balance = v;
acc.depositor = p;
acc.open = d;
return acc;
}
вложител
дата на откриване

```
ACCOUNT list[100];

PERSON p = {"Petrov", 22, "Sofia"};
CLIENT cp = {0, &p};
DATE d0 = {30, November, 2009};

FIRM f = {"ABV", 10000000, "Plovdiv"};
CLIENT cf = {1, &f};
DATE d1 = {5, October, 2009};

list[0] = openAccount(111, 500, cp, d0);
list[1] = openAccount(222, 1000, cf, d1);
```

приход

сметка

внесена сума

```
ACCOUNT income (ACCOUNT acc, int x)
{
  ACCOUNT a;
  a = acc;
  a.balance += x;
  return a;
}
```

разход

сметка

изтеглена сума

```
ACCOUNT expenditure (ACCOUNT acc, int x)
{
  ACCOUNT a;
  a = income (acc, -x);
  return a;
}
```


```
char *depositorName (ACCOUNT acc)
{  char *p;
   switch (acc.depositor.indicator)
   {  case 0:                                  // личност
      {    PERSON *pp = (PERSON *)acc.depositor.personOrFirm;
           p = pp->family;
           break;
      }
      case 1:                                  // фирма
      {    FIRM *pf = (FIRM *)acc.depositor.personOrFirm;
           p = pf->title;
           break;
      }
   }
   return p;
}
```

остатък

сметка

```
int remainder (ACCOUNT acc)
{
   return acc.balance;
}
```

кредитор

сметка

```
int creditor (ACCOUNT acc)
{
   return (remainder (acc)>=0)?1:0;
}
```

Задача: **Структура данни** СМЕТКА

1. **Напишете функция** print(), **която отпечатва дадена** СМЕТКА**.**

```
функция печат: (аргумент acc: СМЕТКА)
  отпечатай номер(acc), баланс(acc)
  избери
      вложител(acc) е ЛИЧНОСТ:
         отпечатай фамилия(вложител(acc)),
                   възраст(вложител(acc)),
                   адрес(вложител(acc)),
      вложител(acc) е ФИРМА:
         отпечатай название(вложител(acc)),
                   капитал(вложител(acc)),
                   адрес(вложител(acc))
  отпечатай ден(дата_откриване(acc))
  избери
      месец(дата_откриване(acc)) е ЯНУАРИ:
         отпечатай януари,
      . . .
      месец(дата_откриване(acc)) е ДЕКЕМВРИ:
        отпечатай декември
 отпечатай година(дата_откриване(acc))
```

2. **Представете сметките на вложители в банка чрез масив.**
3. **Използвайте функциите:**

- openAccount() **за въвеждане на информация за сметките на вложители в банка;**
- income() **за изчисляване на приход;**
- expenditure() **за изчисляване на разход;**
- depositorName() **за получаване име на вложител;**
- remainder() **за изчисляване на остатък;**
- creditor() **за определяне дали вложителят е кредитор;**
- **разпечатайте въведената информация и получените резултати от извикването на горните функции, като използвате функцията** print()**, където е необходимо.**