Задачи за задължителна самоподготовка
по
Увод в програмирането

Калин Георгиев
kalin@fmi.uni-sofia.bg

6 декември 2018 г.

Съдържание

Някои от задачите по-долу са решени в сборника [1]*Магдалина Тодорова, Петър Армянов, Дафина Петкова, Калин Георгиев, "Сборник от задачи по програмиране на C++. Първа част. Увод в програмирането"*. За задачите от сборника е посочена номерацията им от сборника.

1 Увод, основи и примери

1.1 Основни примери

1.1. Превърнете рожденната си дата шестнадесетична, в осмична и в двоична бройни системи.

1.2. Как бихте кодирали вашето име само с числа? Измислете собствено представяне на символни константи чрез редици от числа и запишете името си в това представяне.

Разгледайте стандартната ASCii таблица (`http://www.asciitable.com/`) и запишете името си чрез серия от ASCii кодове.

1.2 Променливи, вход и изход, логически и аритметични операции, условен оператор

1.3. Задача 1.6.[1] Да се напише програма, която по зададени навършени години намира приблизително броя на дните, часовете, минутите и секундите, които е живял човек до навършване на зададените години.

1.4. Задача 1.7.[1] Да се напише програма, която намира лицето на триъгълник по дадени: а) дължини на страна и височина към нея; б) три страни.

1.5. Задача 2.7.[1] Да се напише програма, която въвежда координатите на точка от равнина и извежда на кой квадрант принадлежи тя. Да се разгледат случаите, когато точката принадлежи на някоя от координатните оси или съвпада с центъра на координатната система.

1.6. Задача 1.14.[1] Да се запише булев израз, който да има стойност истина, ако посоченото условие е вярно и стойност - лъжа, в противен случай:

а) цялото число p се дели на 4 или на 7;

б) уравнението $ax^2 + bx + c = 0 (a \neq 0)$ няма реални корени;

в) точка с координати (a, b) лежи във вътрешността на кръг с радиус 5 и център (0, 1); г) точка с координати (a, b) лежи извън кръга с център (c, d) и радиус f;

г) точка принадлежи на частта от кръга с център (0, 0) и радиус 5 в трети квадрант;

д) точка принадлежи на венеца с център (0, 0) и радиуси 5 и 10;

е) x принадлежи на отсечката [0, 1];

ж) x е равно на max {a, b, c};

з) x е различно от max { a, b, c};

и) поне една от булевите променливи x и y има стойност true;

к) и двете булеви променливи x и y имат стойност true;

л) нито едно от числата a, b и c не е положително;

м) цифрата 7 влиза в записа на положителното трицифрено число p;

н) цифрите на трицифреното число m са различни;

о) поне две от цифрите на трицифреното число m са равни помежду си;

п) цифрите на трицифреното естествено число x образуват строго растяща или строго намаляваща редица;

р) десетичните записи на трицифрените естествени числа x и y са симетрични;

с) естественото число x, за което се знае, че е по-малко от 23, е просто.

1.7. Задача 2.12.[1] Да се напише програма, която проверява дали дадена година е високосна.

1.3 Цикли

1.8. Задача 1.20.[1] Да се напише програма, която по въведени от клавиатурата цели числа x и k ($k \geq 1$) намира и извежда на екрана k-тата цифра на x. Броенето да е отдясно наляво.

1.9. Задача 2.40.[1] Да се напише програма, която (чрез цикъл for) намира сумата на всяко трето цяло число, започвайки от 2 и ненадминавайки n (т.е. сумата 2 + 5 + 8 + 11 + ...).

1.10. Задача 2.44.[1] Дадено е естествено число n ($n \geq 1$). Да се напише програма, която намира броя на тези елементи от серията числа $i^3+13\times i\times n+n^3$, $i = 1, 2, ..., n$, които са кратни на 5 или на 9.

1.11. За въведени от клавиатурата естествени числа n и k, да се провери и изпише на екрана дали n е точна степен на числото k.

Упътване: Разделете променливата n на променливата k "колкото пъти е възможно" и проверете дали n достига единица или някое друго число след края на процеса. Използвайте добре подбрано условие за for цикъл, оператора % за намиране на остатък при целочислено деление, и оператора за целочислено деление /.

1.4 Машини с неограничени регистри

Дефиницията на Машина с неогрничени регистри по-долу е взаимствана от учебника [2]*А. Дичев, И. Сосков, "Теория на програмите", Издателство на СУ, София, 1998.*

"Машина с неограничени регистри" (или МНР) наричаме абстрактна машина, разполагаща с неограничена памет. Паметта на машината се представя с безкрайна редица от естествени числа $m[0], m[1], ...$, където $m[i] \in \mathcal{N}$. Елементите $m[i]$ на редицата наричаме "клетки" на паметта на машината, а числото i наричаме "адрес" на клетката $m[i]$.

МНР разполага с набор от инструцкии за работа с паметта. Всяка инструкция получава един или повече параметри (операнди) и може да предизвика промяна в стойността на някоя от клетките на паметта. Инструкциите на МНР за работа с паметта са:

1) `ZERO n`: Записва стойността 0 в клетката с адрес n

2) `INC n`: Увеличава с единица стойността, записана в клетката с адрес n

3) `MOVE x y`: Присвоява на клетката с адрес y стойността на клетката с адрес x

"Програма" за МНР наричаме всяка последователност от инструкции на МНР и съответните им операнди. Всяка инструкция от програмата индексираме с поредния ѝ номер. Изпълнението на програмата започва от първата инструкция и преминава през всички инструкции последователно, освен в някои случаи, опиани по-долу. Изпълнението на програмата се прекратвя след изпълнението на последната ѝ инструкция. Например, след изпълнението на следната програма:

```
0: ZERO 0
1: ZERO 1
2: ZERO 2
3: INC 1
4: INC 2
5: INC 2
```

Първите три клетки на машината ще имат стойност 0, 1, 2, независимо от началните им стойности.

Освен инструкциите за работа с паметта, МНР притежават и една инструкция за промяна на последователноста на изпълнение на програмата:

4) `JUMP x`: Изпълнението на програмата "прескача" и продължава от инструкцията с пореден номер x. Ако програмата има по-малко от $x+1$ инструкции, изпълнението ѝ се прекратява

5) `JUMP x y z`: Ако съдържанията на клетките x и y съвпадат, изпълнението на програмата "прескача" и продължава от инструкцията с пореден номер z. В противен случай, програмата продължава със следващата инструкция. Ако програмата има по-малко от $z+1$ инструкции, изпълнението ѝ се прекратява

Например, нека изпълнето на следната програма започва при стойности на клиетките на паметта 10,0,0,...:

```
0: JUMP 0 1 5
1: INC 1
2: INC 2
3: INC 2
4: JUMP 0
```

След приключване на програмата, първите три клетки на машината ще имат стойности 10, 10, 20.

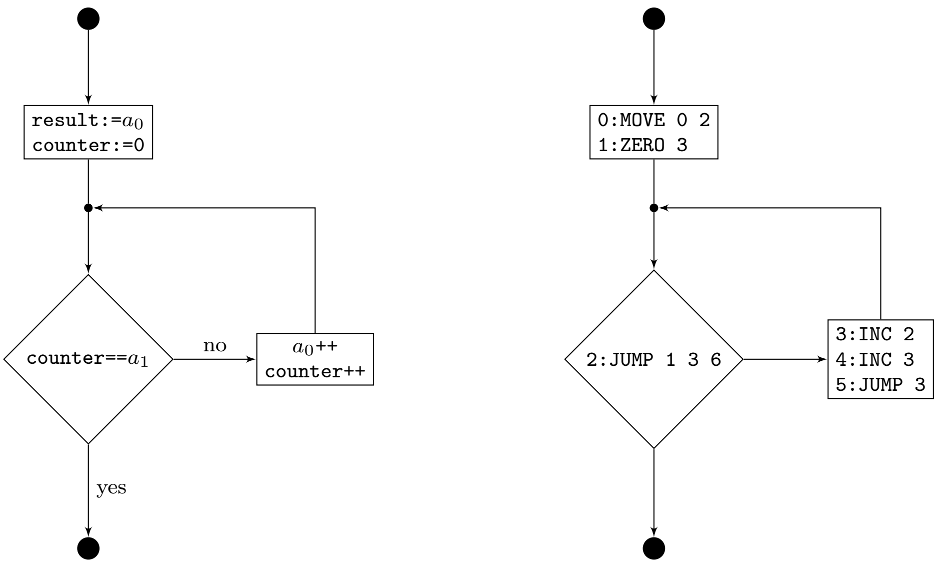
1.12. Нека паметта на МНР е инициалирана с редицата $m, n, 0, 0, \ldots$. Да се напише програма на МНР, след изпълнението на която клетката с адрес 2 съдържа числото $m+n$.

1.13. Нека паметта на МНР е инициалирана с редицата $m, n, 0, 0, \ldots$. Да се напише програма на МНР, след изпълнението на която клетката с адрес 2 съдържа числото $m \times n$.

1.14. Нека паметта на МНР е инициалирана с редицата $m, n, 0, 0, \ldots$. Да се напише програма на МНР, след изпълнението на която клетката с адрес 2 съдържа числото 1 тогава и само тогава, когато $m > n$ и числото 0 във всички останали случаи.

Упътване: На Фигура 1 (a) е показана блок схема на програма, изпозлваща само операторите `=`, `==`, `++` и `if`, която намира в променливата `result` сумата на променливите a_0 и a_1. a_0 и a_1 считаме за дадени. Променливата `count` се иницилиаира с 0, а `result` - с a_0. В цикъл се добавя по една единица към `count` и `result` дотогава, докато `count` достигне стойността на a_1. По този начин, към `result` се добавят a_1 на брой единици, т.е. стойността ѝ се увеличава с a_1 спрямо налчалната ѝ стойност a_0.

На Фигура 1 (b) е показана същата програма, като операторите от първата са заменени със сътответните им инструцкии на МНР. Резултатът от



(a)Програма за сумиране на числата a_0 и a_1 с използване само на операторите =, ==, ++ и if.

(b)Програма за сумиране на клетките $m[0]$ и $m[1]$ с инструкции на МНР.

Фигура 1: Блок схеми на програма за сумиране на числа

програмата се получава в клетката $m[2]$, а за брояч се ползва клетката $m[3]$. На блок схемата са дадени поредните номера на инструкциите в окончателната програмата на МНР:

```
0:MOVE 0 2
1:ZERO 3
2:JUMP 1 3 6
3:INC 2
4:INC 3
5:JUMP 3
```

2 Типове и функции

2.1 Прости примери за функции

2.1. Задача 4.12.[1] Да се напише булева функция, която проверява дали дата, зададена в следния формат: dd.mm.yyyy е коректна дата от грегорианския календар.

2.2. Задача 4.25.[1] Да се дефинира процедура, която получава целочислен параметър n и база на бройна система $k \leq 16$. Процедурата да отпечатва на екрана представянето на числото n в системата с база k.

2.3. Задача 2.57.[1] Да се напише булева функция, която проверява дали сумата от цифрите на дадено като параметър положително цяло число е кратна на 3.

2.4. Задача 2.81.[1] Едно положително цяло число е съвършено, ако е равно на сумата от своите делители (без самото число). Например, 6 е съвършено, защото $6 = 1+2+3$; числото 1 не е съвършено. Да се напише процедура, която намира и отпечатва на екрана всички съвършени числа, ненадминаващи дадено положително цяло число в параметър n.

2.2 Елементарна растерна графика

Следните задачи да се решат с показаните на лекции графични примитиви, базирани на платформата за компютърни игри SDL2. За целта е необходимо да инсталирате SDL2 на компютъра си и да настроите средата си за програмиране така, че да свърже SLD2 с вашия проект. Информация за това можете да намерите на сайта на платформата. Задачите можете да решите с помощта на всяка друга билиотека, поддържаща примитивите за рисуване на точки и отсечки.

Примерната програма от лекции използва файла `mygraphics.h`, който можете да намерите в хранилището на курса:

```
#include "mygraphics.h"
```

`Mygraphics` "обвива" библиотеката SLD2 и дефинира следните лесни за използване макроси:

- `setColor (r,g,b)`: Дефинира цвят на рисуване с компоненти $r, g, b \in [0, 255]$. Например, белият цвят се задава с $(255, 255, 255)$, червеният с $(255, 0, 0)$ и т.н.
- `drawPixel(x,y)`: Поставя една точка на екранни коррдинати (x, y).

- `drawLine (x1,y1,x2,y2)`: Рисува отсечка, свързваща точките с екранни координати (x_1, y_1) и (x_2, y_2).
- `updateGraphics()`: Извиква се веднъж в края на програмата, за да се изобрази нарисуваното с горните примитиви.

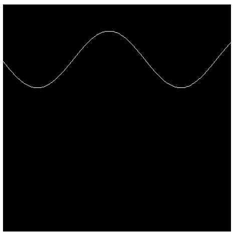
2.5. По дадени екранни координати (x, y) на горния ляв ъгъл на квадрат, дължина на страната a на квадрата и число n:

- Да се нарисува квадратна матрица от $n\times$n квадрата със страна a/n, изпълваща дадения квдрат.
- Квадратите от горното условие да се заменят с тригълниците, образувани от пресичането на диагоналите им.

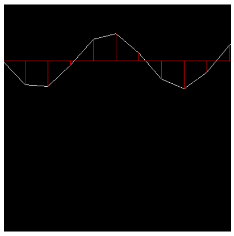
2.6. Да се нарисуват програмно следните фигури:

- Равностранен триъгълник
- Равностранен шестоъгълник
- Равностранен многоъгълник по дадени координати на пресечната точка на симетралите му (център), брой страни n и разстояние от центъра до върховете r. При какви стойности на n фигурата наподобява окръжност?
- Логаритмична крива
- Елипса с център дадени (x, y) и радиуси дадени r_1 и r_2

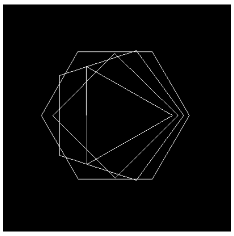
Упътване към задачата за чертане на графика на функцията $y = sin(x)$: Рисуват се отсечки между последователни точки от графиката на функцията, като всяка следваща точка се получава като увеличаваме стойността на аргумента x с числото `stepX`. Тъй като $sin(x) \in [-1, 1]$, ако директно визуализираме точките на получените по този начин координати $(x, sin(x))$, те ще са “сгъстени” около правата $y = 0$ и резултатът няма да е добър. Поради това, координатите на получените точки от кривата се умножават по `scaleX` и `scaleY` съответно, $(x.scaleX, sin(x).scaleY)$, за да се “разпъне” графиката по двете оси. (Вж. Фигура 2)



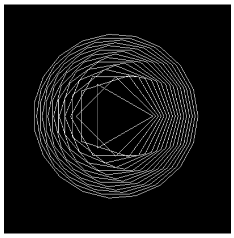
(a)Графика на $y = sin(x)$, нарисувана с 300 отсечки



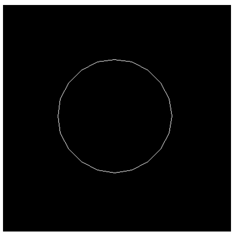
(b)Графика на $y = sin(x)$ с 10 отсечки. Нарисувани са също отсечки между точките от графиката на функцията и абсцисата



(c)Четири многоъгълника, нарисувани един върху друг с нарастващ радиус



(d)Шестнадесет многоъгълника, нарисувани един върху друг с нарастващ радиус



(e)Многоъгълник с 20 върха, приближаващ окръжност

Фигура 2: Примерни резултати от решенията на някои задачи

```
const double //scaleX: коефициент на скалиране по X
             scaleX = 40.0,
             //y0: ордината на началната точка
             y0 = 100,
             //scaleY: коефициент на скалиране по Y
             scaleY = 50.0,
             //stepX: стъпка за нарастване на аргумента
             stepX = 0.05;
             //nsegments: брой сегменти от кривата
const int nsegments = 300;

for (int i = 0; i < nsegments; i++)
{
     double x     = scaleX*i*stepX,
            xnext = scaleX*(i+1)*stepX,
            y     = y0+scaleY*sin(stepX*i),
            ynext = y0+scaleY*sin(stepX*(i+1));
     drawLine (x,y,xnext,ynext);
}
```

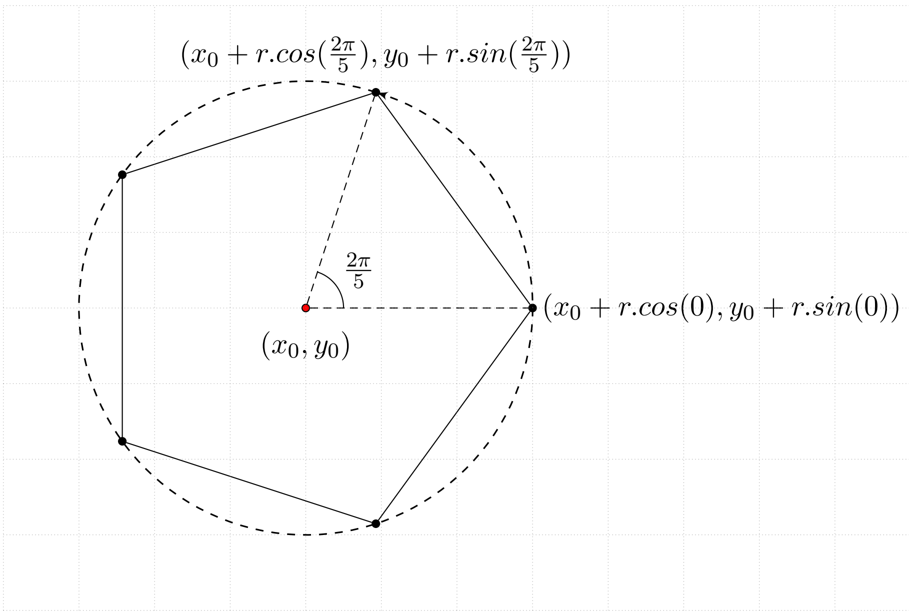
Упътване към задачата за чертане на многоъгълник: Върховете на многоъгълника получаваме, като започнем с точката с координати $(x + radius, y)$ и получаваме всяка следваща точка като "завъртим" предишната около (x, y) с $2\pi/n$ радиана, където n е броят на върховете на многоъгълника. Получените по този начин точки се съединияват с отсечки. (Вж. Фигура 2 и Фигура 3.)

```
/*
Функция polygon (n,x,y,radius): Рисуване на многоъгълник
параметър n: брой върхове на многоъгълника
параметри x,y: координати на центъра на многоъгълника
параметър radius: разстояние от центъра до върховете
*/
void polygon (int n, double x, double y, double radius)
{
  for (int side = 0; side < n; side++)
  {
     drawLine (radius*cos(side*2.0*M_PI/n)+x,
               radius*sin(side*2.0*M_PI/n)+y,
               radius*cos((side+1)*2.0*M_PI/n)+x,
               radius*sin((side+1)*2.0*M_PI/n)+y);
  }
}
```

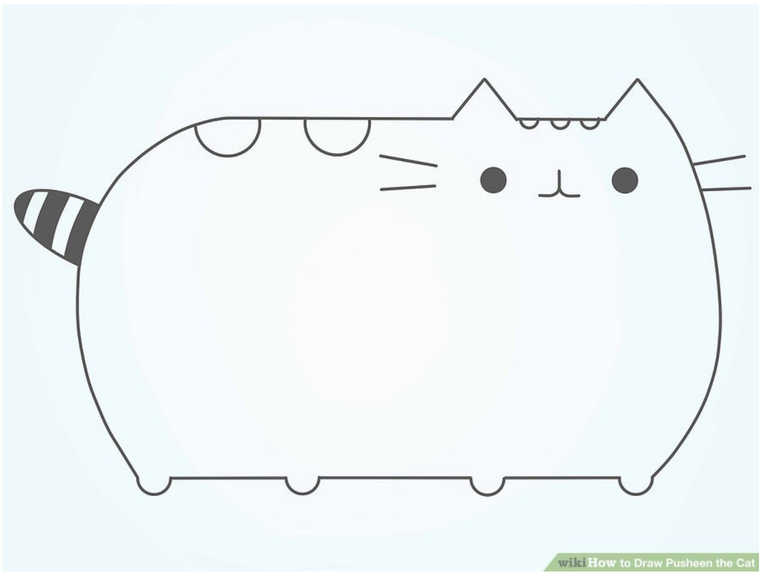
2.7. Да се нарисува програмно котката Pusheen на Фигура 4 (вж. [3]).

2.8. (*) *Следната задача илюстрира метода на трапеците (Trapezoidal rule) за приближено изчисление на определени интеграли:*

Да се нарисуват програмно координатни оси на евклидова коорднинатна система с даден център в екранните координати (x,y). Да приемем,



Фигура 3: Рисуване на петоъгълник чрез намиране на 5 равноотдалечени точки по окръжността с радиус r и център (x_0, y_0)



Фигура 4: Pusheen the cat. Фигурата е от [3]

че в програмата е дефинирана функция `double f (double x)`, за която знаем, че е дефинирана за всяка стойност на x.

- Да се изобрази графиката на фукцията спрямо нарисувана координатна система
- Да се приближи чрез трапеци с дадена дължина на основата δ фигурата, заключена между видимата графика на фигурата и абсцисата
- Да се визуализират така получените трапеци
- Да се изчисли сумата от лицата на така получените трапеци
- Да се експериментра с различни дефиниции на функцията f

3 Цикли, масиви и низове

3.1 Цикли II

Където не е посочено изрично, под "редица от числа $a_0, a_1, ..., a_{n-1}$" по-долу се разбира последователност от n числа, въведени от стандартния вход. Задачите да се решат *без* използването на масиви.

3.1. Задача 3.1. [1] Да се напише програма, която въвежда редица от n цели числа ($1 \leq n \leq 50$) и намира и извежда минималното от тях.

3.2. Задача 3.2. [1] Да се напише програма, която въвежда редицата от n ($1 \leq n \leq 50$) цели числа $a_0, a_1, ..., a_{n-1}$ и намира и извежда сумата на тези елементи на редицата, които се явяват удвоени нечетни числа.

3.3. Задача 3.3. [1] Да се напише програма, която намира и извежда сумата от положителните и произведението на отрицателните елементи на редицата от реални числа $a_0, a_1, ..., a_{n-1}$ ($1 \leq n \leq 20$).

3.4. Задача 3.7. [1] Да се напише програма, която изяснява има ли в редицата от цели числа $a_0, a_1, ..., a_{n-1}$ ($1 \leq n \leq 100$) поне два последователни елемента с равни стойности.

3.5. Задача 3.8. [1] Да се напише програма, която проверява дали редицата от реални числа $a_0, a_1, ..., a_{n-1}$ ($1 \leq n \leq 100$) е монотонно растяща.

3.6. Задача 3.15. [1] Да се напише програма, която въвежда реланите вектори $a_0, a_1, ..., a_{n-1}$ и $b_0, b_1, ..., b_{n-1}$ ($1 \leq n \leq 100$), намира скаларното им произведение и го извежда на екрана.

3.7. Задача 3.10. [1] Да се напише програма, която за дадена числова редица $a_0, a_1, ..., a_{n-1}$ ($1 \leq n \leq 100$) намира дължината на най-дългата ú ненамаляваща подредица $a_i, a_{i+1}, ..., a_{i+k}$ ($a_i \leq a_{i+1} \leq ... \leq a_{i+k}$).

3.2 Цикли и низове

Където не е посочено изрично, под “редица от символи $s_0, s_1, ..., s_{n-1}$ $(1 \leq n \leq 100)$ $a_0, a_1, ..., a_{n-1}$” по-долу се разбира символен низ с дължина n, въведен от клавиатурата в масив от тип `char[255]`.

3.8. Задача 3.11. [1] Дадена е редицата от символи $s_0, s_1, ..., s_{n-1}$ $(1 \leq n \leq 100)$. Да се напише програма, която извежда отначало всички символи, които са цифри, след това всички символи, които са малки латински букви и накрая всички останали символи от редицата, запазвайки реда им в редицата.

3.9. Задача 3.13. [1] Задача 3.13. Да се напише програма, която определя дали редицата от символи $s_0, s_1, ..., s_{n-1}$ $(1 \leq n \leq 100)$ е симетрична, т.е. четена отляво надясно и отдясно наляво е една и съща.

3.10. Да се напише функция, която по два низа намира дължината на най-дългия им общ префикс. *Префикс на низ наричаме подниз със същото начало като дадения. Пример: празният низ и низовете “a”, “ab”, и “abc” са всички възможни префикси на низа “abc”. Дължината на най-дългия общ префикс на низовете “abcde” и “abcuwz” е 3.*

3.11. Да се напише функция, която в даден низ замества всички малки латински букви със съответните им големи латински букви.

3.12. Да се напише функция `reverse(s)`, която превръща даден низ в огледалния му образ. *Например, низът “abc” ще се пребразува до “cba”.*

3.13. Да се напише функция, която по даден низ s, всички букви в който са латински, извършва следната манипулация над него: Ако s съдържа повече малки, отколкото големи букви, замества всички големи букви в s с малки. В останалите случаи, всички малки букви се заместват с големи.

3.14. Задача 3.26. "Хистограма на символите"[1] Символен низ е съставен единствено от малки латински букви. Да се напише програма, която намира и извежда на екрана броя на срещанията на всяка от буквите на низа.

3.15. Да се напише булева функция, която по дадени низове s_1 и s_2 проверява дали s_2 е подниз на s_1 (*Например, низът “uv” е подниз на низовете “abuvc”, “uvz”, “zuv” и “uv”, но не е подниз на низа “uwv”.*). Функцията да *не* използва вложени цикли.

3.16. Задача 3.28. "Търсене на функция"[1] Дадени са два символни низа с еднаква дължина s_1 и s_2, съставени от малки латински букви. Да се напише програма, която проверява дали съществува функция $f : char \rightarrow$

$char$, изобразяваща s_1 в s_2, така че $f(s_1[i]) = f(s_2[i])$ и $i = 1..$дължината на s_1 и s_2. *Упътване: За да е възможна такава функция, не трябва в s_1 да има сивмол, на който съответстват два или повече различни символи в s_2. Например, низът "aba" може да бъде изобразен в низа "zwz", но не и в низа "zwu".*

3.3 Матрици и вложени цикли

3.17. Задача 3.18. [1] Дадени са числовите редици $a_0, a_1, ..., a_{n-1}$ и $b_0, b_1, ..., b_{n-1}$ $(1 \leq n \leq 50)$. Да се напише програма, която въвежда от клавиатурата двете редици и намира броя на равенствата от вида $a_i = b_j$ $(i = 0, ..., n-1, j = 0, ..., n-1)$.

3.18. Задача 3.21. [1] Две числови редици си приличат, ако съвпадат множествата от числата, които ги съставят. Да се напише програма, която въвежда числовите редици $a_0, a_1, ..., a_{n-1}$ и $b_0, b_1, ..., b_{n-1}$ $(1 \leq n \leq 50)$ и установява дали си приличат.

3.19. Задача 3.29. [1] Дадена е квадратна целочислена матрица A от n-ти ред $(1 \leq n \leq 50)$. Да се напише програма, която намира сумата от нечетните числа под главния диагонал на A (без него).

3.20. Задача 3.45. [1] Матрицата A има седлова точка в $a_{i,j}$, ако $a_{i,j}$ е минимален елемент в i-тия ред и максимален елемент в j-тия стълб на A. Да се напише програма, която извежда всички седлови точки на дадена матрица A с размерност $n \times m(1 \leq n \leq 20, 1 \leq m \leq 30)$.

3.21. Задача 3.113. (периодичност на масив). [1] Да се напише програма, която проверява дали в едномерен масив от цели числа съществува период. Например, ако масивът е с елементи 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, периодът е 3. Ако период съществува, да се изведе.

3.22. Да се напише програма, която въвежда от клавиатурата матриците от цели числа $A_{N \times M}$ и $B_{M \times N}$ и извежда на екрана резултатът от умножението на двете матрици.

3.4 Елементарно сортиране на масиви

3.23. Да се реализира функция, сортираща масив по метода на мехурчето. При този метод масивът $a[0], .., a[n-1]$ се обхожда от началото към края, като на всяка стъпка се сравняват двойката съседни елементи $a[i]$ и $a[i+1]$. Ако $a[i+1] < a[i]$, местата им се разменят. Този процес се извършва n пъти.

3.24. Да се напише функция `unsortedness`, която оценява доколко един масив е несортиран като преброява колко от елементите му “не са на местата си”. Т.е. функцията намира броя на тези елементи a_i, които не са i-ти по големина в масива. Например, за масива $\{0, 2, 1\}$ това число е 2.

3.25. Да се напише функция `swappable`, която за масива $a[0], .., a[n-1]$ проверява дали има такова число $i(0 < i < n-1)$, че масива $a_i, .., a_n, a_0, .., a_{i-1}$ е стортиран в нарастващ ред. Т.е. може ли масивът да се раздели на две части (незадължително с равна дължина) така, че ако частите се разменят, да се получи нареден масив. Пример за такъв масив е $\{3, 4, 5, 1, 2\}$.

Функцията `std::clock()` от `<ctime>` връща в абстрактни единици времето, което е изминало от началото на изпълнение на програмата. Обикновено тази единица за време, наречена "`tick`", е фиксиран интервал "реално" време, който зависи от хардуера на системата и конфирграцията ѝ. Константата `CLOCKS_PER_SEC` дава броя `tick`-ове, които се съдържат в една секунда реално време.

Чрез следния примерен код може да се измери в милисекунди времето за изпълнение на програмния блок, обозначен с "...".

```
clock_t start = std::clock();
//...
clock_t end = std::clock();

long milliseconds = (double)(end-start)/
                    (CLOCKS_PER_SEC/1000.0);
```

Функцията `rand()` от `<cstdlib>` генерира редица от псевдо-случайни числа. Всяко последователно изпълнение на функцията генерира следващото число от редицата. За да се осигури, че при всяко изпълнение на програмата ще се генерира различна редица от псевдо-случайни числа, е необходимо да се изпълни функцията `srand()` с параметър, който е различен за всяко изпълнение на програмата. Една лесна възможност е да се позлва резултата на функцията `time(0)`, която дава текущото време на системния часовник в стандарт `epoch time`. Достатъчно е `srand()` да се изпълни веднъж за цялото изпълнение на програмата.

Чрез следния примерен код може да се генерира редица от 10 (практически) случайни числа, които са различни при всяко изпълнение на програмата.

```
srand (time(0));
for (int i = 0; i < 10; i++)
{
  std::cout << rand () << std::endl;
}
```

Стойностите на `rand()` са в интервала $[0..INT\_MAX]$. Ако е нужно да генерирате стойности в друг интервал, например $[0..N]$, това може да стане по формулата $\frac{rand()}{INT\_MAX} \times N$ (трябва да избегнете целочисленото делене)!

3.26. Да се измери емпирично времето за изпълнение на алгоритъма за сортиране по метода на мехурчето. Да се начертае графика на зависимостта на времето за изпълнение от големината на масива. Всеки тест да е с наново генериран масив от случайни числа.

3.27. Да се въведе матрица от числа $A_{N \times M}$.

- Да се сортира всеки от редовете на матрицата
- Да се сортира всяка от колоните на матрицата

Така получените матрици да се отпечатат на стандартния изход.

`Bozosort` е случайностен алгоритъм за сортиране на масиви. При този алгоритъм, на всяка стъпка се разменят две случайни числа от масива, след което се проверява дали масивът се е сортирал. Процесът продължава до сортиране на масива.

3.28. Да се реализира алгоритъма `Bozosort`. Да се измери емпирично времето му за изпълнение. *Внимание: тествайте с достатъчно малки масиви, тъй като този алгоритъм е изключително бавен.*

3.5 Низове II

- Задача 3.55. [1] Дадена е квадратна таблица $A_{n \times n}$ $(1 \leq n \leq 30)$ от низове, съдържащи думи с максимална дължина 6. Да се напише програма, която проверява дали изречението, получено след конкатенацията на думите от главния диагонал (започващо от горния ляв ъгъл) съвпада с изречението, получено след конкатенацията на думите от вторичния главен диагонал на А (започващо от долния ляв ъгъл).

- Задача 3.56. [1] Дадена е квадратна таблица А от n-ти ред $(1 \leq n \leq 20)$ от низове, съдържащи думи с максимална дължина 9. Да се напише програма, която намира и извежда на екрана изречението, получено след обхождане на А по спирала в посока на движението на часовниковата стрелка, започвайки от горния ляв ъгъл. Например ако матрицата А има вида:
  $$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$
  изречението след обхождането по спирала е: “abcfihgde”.

- Задача 3.57 (Inner Join). [1] Нека са дадени два масива от низове – students и grades с най-много 20 низа във всеки. Низовете в масива students имат вида “XXXXXX YYYY...”, където “XXXXXX” е шестцифрен факултетен номер, а “YYYY...” е име с произволна дължина. Низовете в grades имат вида “XXXXXX YYYY”, където “XXXXXX” е шестцифрен факултетен номер, а “YYYY” е оценка под формата на число с плаваща запетая. И двата масива са сортирани във възходящ ред по факултетен номер. Възможно е в някой от двата масива да има данни за факултетни номера, за които няма данни в другия. И в двата списъка даден факултетен номер се среща най-много един път. Да се напише програма, която извежда на екрана имената и оценките на тези студенти, за които има информация и в двата списъка, като оценките са увеличени с 1 единица, но са максимум 6.00.

3.6 Инвариант на цикъл и верификация на програми

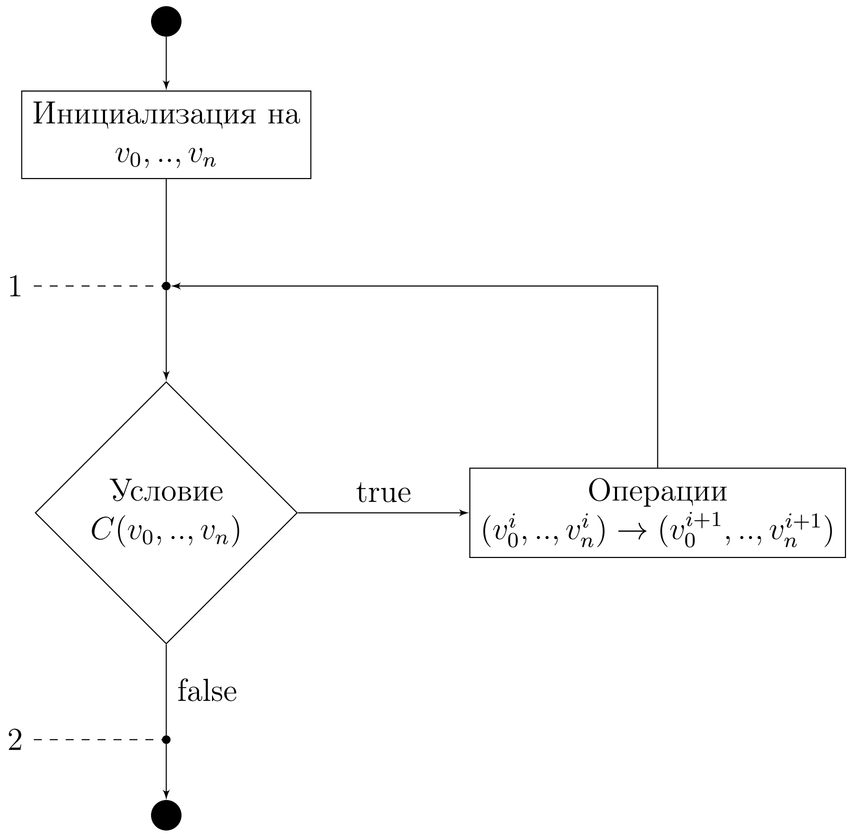
Изложението в тази секция е силно опростена версия на метода на Флойд (Floyd) за верификация на блок схеми, описан в монументалната му статия [4]. Редица идеи и задачи са взаимствани от книгата [5] на А. Соскова и С. Николова “Теория на програмите в задачи”. За по-подробно изложение и допълнителни примери и задачи препоръчваме тази книга. Изложението по-долу е нарочно опростено, като на места е жертвана прецизността му.

Инвариант на цикъл

Нека е дадена програма или част от програма, която се се състои от единствен `while` цикъл. Нека имаме някакъв набор от работни променливи $v_1, .., v_n$, които се инициализират непосредствено преди `while` цикъла. Условието на цикъла зависи изцяло от работните променливи, а тялото на цикъла използва или променя само тях. Приемаме, че програмата не произвежда странични ефекти и не зависи от такива.

Тоест, за програмата знаем, че: (1) не извършва входни и изходни операции, (2) поведението ѝ зависи изцяло от началните стойности на работните променливи $v_1, .., v_n$ и (3) не модифицира никакви други променливи, освен работните. Такава програма можем да изобразим чрез блок схемата на Фигура 5.

На фигурата $C(v_1, .., v_n)$ е някакъв логически израз, зависещ само от $v_1, .., v_n$. Да допуснем, че сме “замразили” изпълнението на програмата в точката, обозначена с “1”, точно преди да започне i-тото поред ($i \geq 0$) изпълнение на тялото на цикъла, т.е. непосредствено преди да бъдат изпълнени операторите в него за i-ти път. В този момент n-торката работни про-



Фигура 5: Блок схема на част от програма с `while` цикъл

менливи $(v_1, .., v_n)$ имат някакви конкретни стойности. Да ги обозначим с $(v_0^i, .., v_n^i)$. След изпълнение на всички оператори в тялото на цикъла работните променливи получават някакви нови стойности. Това са точно стойностите $(v_0^{i+1}, .., v_n^{i+1})$, които работните променливи ще имат в началото на $i + 1$-вата итерация. Това е изобразено на фигурата чрез прехода $(v_0^i, .., v_n^i) \to (v_0^{i+1}, .., v_n^{i+1})$.

Нека приемем, че при някакво конкретно изпълнение на програмата, при конкретни дадени начални стойности на работните променливи $(v_0^0, .., v_n^0)$, тялото на цикъла ще бъде изпълнено точно $k \geq 0$ пъти, след което условието $C(v_1, .., v_n)$ на цикъла ще се наруши и той ще приключи. Изпълнението на програмата достига точката, обозначена с "2". По този начин се получава редицата от n-торки $(v_0^0, .., v_n^0), .., (v_0^k, .., v_n^k)$, като за всички нейни членове $0 \leq i < k$, освен за последния, знаем че е вярно $C(v_0^i, .., v_n^i)$, а за последния, k-ти член, е вярно обратното — $\neg C(v_0^k, .., v_n^k)$.

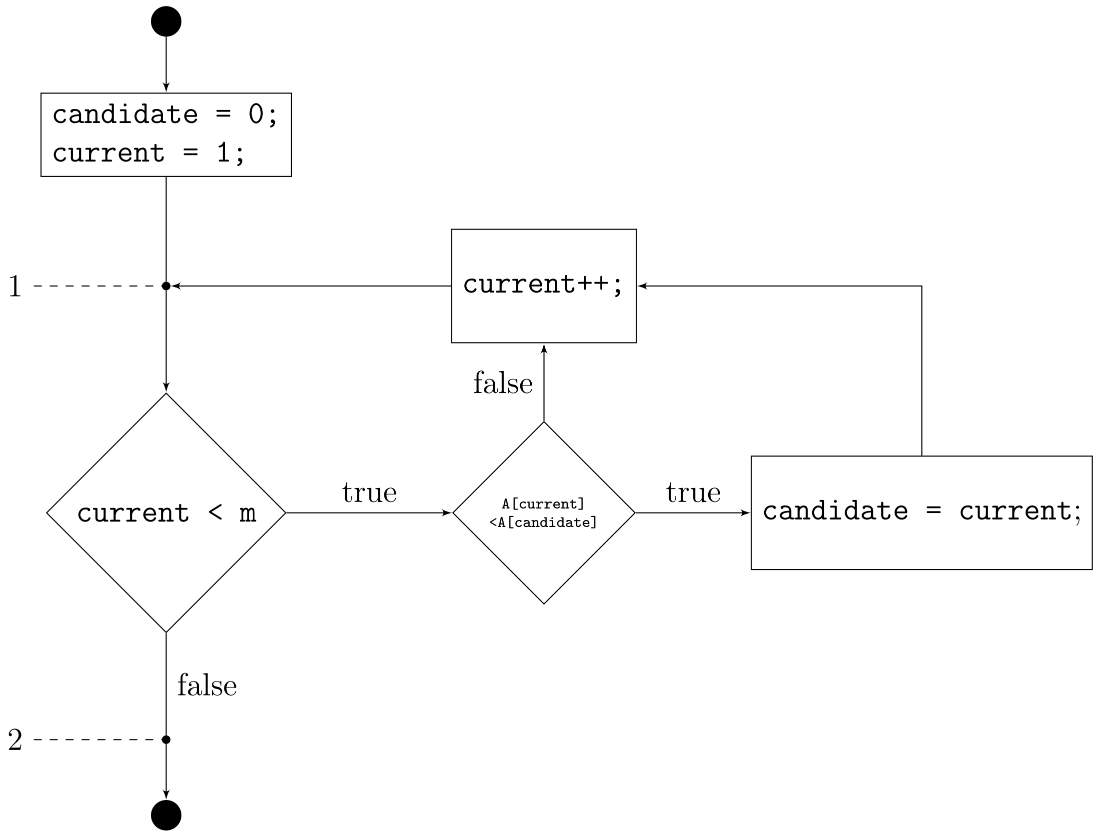
Освен свойството $C(v_0^i, .., v_n^i)$ (или неговото отрицание), което знаем за всички последователни стойности на работните променливи, можем да въведем още едно свойство $I(v_0^i, .., v_n^i)$, наречено "инвариант на цикъла". За свойството I искаме да е вярно за всички възможни стойности на работните променливи, дори и за последния член на редицата им. От там идва името "инвариант", т.е. факт, което е винаги верен.

Това свойство може да е произволно, например тъждествената истина `true` изпълнява условието за инвариант, тъй като е вярна за всички членове на редицата на работните променливи. Инвариантът обаче е най-полезен, когато представя "смисъла" на работните променливи, и ако от верността на $\neg C(v_0^k, .., v_n^k) \,\&\, I(v_0^k, .., v_n^k)$ можем да изведем нещо полезно за "крайния резултат" от изпълнението на цикъла, съдържащ се в стойностите $(v_0^k, .., v_n^k)$.

Верификация на програми

Понятието "инвариант" ще илюстрираме чрез едно негово приложение: "Верификация на програма". Верификацията на програма е доказателство, че при необходимите начални условия дадена програма изчислява стойност, удовлетворяваща някакво желано свойство. Като пример да разгледаме следната програма, за която ще се уверим, че намира най-малък елемент на едномерен масив A с $m > 0$ елемента $A[0], .., A[m-1]$.

```
size_t candidate = 0, current = 1;
while (current < m)
{
  if (A[candidate] < a[current])
```



Фигура 6: Блок схема на програмата за намиране на най-малък елемент в масив

```
  {
    candidate = current;
  }
  current++;
}
```

Можем да приложим метода на инварианта, за да докажем строго, че когато цикълът приключи, то елементът $A[candidate]$ е гарантирано по-малък или равен на всички останали елементи на масива. Като първа стъпка, за улеснение изобразяваме програмата като блок схемата на Фигура 6.

Работните променливи на програмата, освен масива A, са целочислените променливи `candidate` и `current`. Какъв е смисълът на тези променливи? Веднага се вижда, че `current` е просто брояч — служи за обхождане на елементите на масива последователно от $A[0]$ до $A[m-1]$, като на всяка итерация от цикъла се разглежда стойността на $A[current]$.

Интуитивно се вижда, че `candidate` е намереният най-малък елемент на масива до текущия момент от обхождането. Тоест, можем да се надяваме, че ако сме разгледали първите i елемента на масива, то правилно сме определили, че най-малкият от тях е $A[current]$.

Тези размишления може да запишем чрез инварианта:

$I ::= A[candidate] = min(A[0], .., A[current - 1])$.

Този инвариант очевидно е верен в началото на изпълнението на програмата, когато $current = 1$, а $candidate = 0$. Как да се уверим, че инвариантът е валиден за всички стойности, през които преминават работните променливи?

Нека допуснем, че инвариантът е верен за някаква стъпка i от изпълнението на цикъла. Тоест, допускаме $I(candidate^i, current^i)$, или $A[candidate^i] = min(A[0], .., A[current^i - 1])$.

На итерация $i+1$ имаме и $current^{i+1} = current^i + 1$, и следователно трябва да покажем, че $A[candidate^{i+1}] = min(A[0], .., A[current^{i+1} - 1])$.

След навлизане в тялото на цикъла, имаме две възможности:

1) $A[current^i] \geq A[candidate^i]$. От допускането следва, че добавянето на $A[current^i]$ към $min(A[0], .., A[current^i - 1])$ не променя стойността на минимума, или $min(A[0], .., A[current^i - 1]) = min(A[0], .., A[current^i - 1], A[current^i])$. От тук директно се вижда, че инвариантът е верен за итерация $i+1$.

2) $A[current^i] < A[candidate^i]$. Тъй като по допускане $A[candidate^i] = min(A[0], .., A[current^i - 1])$, от тук следва, че $A[current^i] < min(A[0], .., A[current^i - 1])$, следователно

   $A[current^i] = min(A[0], .., A[current^i - 1], A[current^i])$.

Но след присвояването имаме $candidate^{i+1} = current^i$, от където $A[candidate^{i+1}] = min(A[0], .., A[current^i])$. Но $current^i = current^{i+1} - 1$, от където получаваме верността на I за итерация $i+1$: $A[candidate^{i+1}] = min(A[0], .., A[current^{i+1} - 1])$.

Доказателството протича по индукция. Уверяваме се, че инвариантът е верен за цялата редица от стойности на `candidate` и `current`.

Какво се случва в края на цикъла, когато имаме $\neg(current < m)$? Тъй като числата са цели, а `current` се увеличава само с единица, можем да заключим, че $current = m$. Замествайки това равенство в инварианта, получаваме твърдението

$A[candidate] = min(A[0], .., A[m - 1])$.

Тоест, намерили сме минималния елемент на масива.

3.29. Да се докаже строго, че за следната функция е вярно $pow(x, y) = x^y$:

а)
```
unsigned int pow (unsigned int x, usnigned int y)
{
  unsigned int p = 1, i = 0;
  while (i < y)
  {
    p *= x;
    i++;
  }
  return p;
}
```

б)
```
unsigned int pow (unsigned int x, usnigned int y)
{
  unsigned int p = 1;
  while (y > 0)
  {
    p *= x;
    y--;
  }
  return p;
}
```

в)
```
unsigned int pow (unsigned int x, usnigned int y)
{
  unsigned int z = x, t = y, p = 1;
  while (t > 0)
  {
    if (t%2 == 0)
    {
      z *= z;
```

```
        t /= 2;
      }else{
        t = t - 1;
        p *= z;
      }
    }
    return p;
  }
```

3.30. Да се докаже строго, че за следната функция е вярно $sqrt(n) = [\sqrt{n}]$.

```
unsigned int sqrt (unsigned int n)
{
  unsigned int x = 0, y = 1, s = 1;
  while (s <= n)
  {
    x++;
    y += 2;
    s += y;
  }
  return x;
}
```

3.31. Да се напише програма, която проверява дали дадени два масива A и B с еднакъв брой елементи m са еднакви, т.е. съдържат същите елементи в същия ред. Да се докаже строго, че програмата работи правилно.

3.32. Да се докаже, че следната функция връща истина тогава и само тогава, когато масивът `A` с `n` елемента съдържа елемента `x`.

```
bool member (int A[], size_t n, int x)
{
  size_t i = 0;
  while (i < n && A[i] != x)
  {
    i++;
  }
  return i < n;
}
```

3.33. Да се напише програма, която проверява дали даден масив A с m елемента е сортиран, т.е. дали елементите му са наредени в нарастващ ред. Да се докаже строго, че програмата работи правилно.

Съществуват редица съвременни методи за верификация на програми. Пълната версия на метода, използван по-горе, се нарича "логика на Флойд-Хоар" (Floyd–Hoare logic) и е само един представител на този клас методи.

Също така, в компютърните науки има направление, наречено "Синтез на програми" (Program refinement). При синтеза на програми се решава обратната задача: по спецификация на входа и изхода да се генерира програма, която удовлетворява спецификацията.

Препоръчваме на любознателния читател да се запознае с логиката на Floyd–Hoare и методите за синтез на програми.

4 Указатели и програмен стек

4.1. Да се дефинира функция, която получава като параметри два масива с еднакъв брой елементи. Функцията да разменя съответните елементи на масивите ($a[i] \leftrightarrow b[i]$).

4.2. Да се напише булева функция `bool duplicates (long *ponters[])`, която получава като параметър масив `pointers` от указатели към целочислени променливи. Функцията да проверява дали поне две от съответните променливи имат еднакви стойности.

4.3. Да се дефинира функция `swap([подходящ тип] a,[подходящ тип] b)`, която разменя стойностите на целочислените променливи a и b.

4.4. Да се дефинира функцията `bool commonel (int *arrays[],int npointers, int arrlengths[])`. Масивът `arrays` съдържа `npointers` на брой указатели към масиви от цели числа. `i`-тият масив има големина `arrlengths[i]`. Функцията да връща истина, ако има поне едно число x, което е елемент на всички масиви.

4.5. Да се дефинира функцията `bool subarrays (int *arrays[],int npointers, int arrlengths[])`. Масивът `arrays` съдържа `npointers` на брой указатели към масиви от цели числа. `i`-тият масив има големина `arrlengths[i]`. Функцията да връща истина, ако поне един от масивите е подмасив на друг масив. Масивът a наричаме подмасив на b, ако заетата от a памет е част от заетата от b памет. Да се напишат подходящи тестове за функцията.

4.6. Да се дефинира рекурсивна функция `double sum(size_t n)`, която въвежда n числа от стандартния вход връща сумата им. *Да не се използват оператори за цикъл!*

4.7. Да се дефинира рекурсивна функция `reverse(n)`, която въвежда n числа от стандартния вход и ги извежда в обратен ред. *Да не се използват масиви. Да се използва програмния стек чрез рекурсия.*

4.8. Да се дефинира функция `void getmax (long *pmax, size_t n)`, която въвежда n числа от стандартния вход и записва максималното от тях в променливата, сочена от указателя `pmax`.

Пример: Следната програма ще изведе най-малкото от 5 въведени от стандартния вход числа.

```
int main ()
{
  long max = -1;
  getmax (&max,5);
```

```
  std::cout << max;

  return 0;
}
```

Функцията да се реализира по два начина: чрез цикъл и чрез използване на рекурсия без оператори за цикъл.

Литература

[1] Магдалина Тодорова, Петър Армянов, Дафина Петкова, Калин Георгиев, “Сборник от задачи по програмиране на C++. Първа част. Увод в програмирането”

[2] А. Дичев, И. Сосков, “Теория на програмите”, Издателство на СУ, София, 1998

[3] Wikihow, How to Draw Pusheen the Cat, https://www.wikihow.com/Draw-Pusheen-the-Cat

[4] R. W. Floyd. “Assigning meanings to programs.” Proceedings of the American Mathematical Society, Symposia on Applied Mathematics. Vol. 19, pp. 19–31. 1967.

[5] Александра Соскова, Стела Николова, “Теория на програмите в задачи”, Софтех, 2003