Идеи за решаване на задачите

Задача 1. Черно и бяло

Лентичките са подредени една след друга в кръг. Понеже са три, всяка от тях съседства с другите две от кръга. Това означава, че ако в кръга има бял участък, той е най-много един! И наистина: ако в кръга има бяла и две черни, то дължината на бялата е отговорът на задачата; ако има две бели и черна, то двете бели задължително са съседни и сумата от дължините им е отговорът; ако всичките са бели – тяхната сума е отговорът. Това показва как лесно да решим задачата – четем данни за поредната лентичка и ако тя е бяла добавяме дължината към сумата от дължините на всички бели.

Да припомним малко теория:

Величини (данни), които изразяват суми на други величини (данни) се наричат суматори. При програмирането на променливата, която ще се ползва за суматор, трябва първоначално да й се зададе стойност 0 (нула).

По време на четене на данните правим необходимите действия за обработката им (проверка дали лентичката е бяла и евентуалното добавяне на дължината й към сумата от дължини на белите) още след прочитането. След обработката на данните за поредната лентичка, те повече не трябват за останалата част от програмата. Това позволява да не заделяме променливи за данните на всички лентички, а само за една – след обработката на данните за текущата лентичка, „забравяме" за тях и променливите, които ги съхраняват използваме за други цели, а именно: за съхраняване на данни за следващата лентичка. Освен, че така се отърваваме от необходимостта да заделяме променливи (и да им измисляме уникални имена) за всички данни, това позволява действията за всички данни да бъдат описани чрез действие само върху една данна или един комплект данни (в нашия случай за всяка лентичка имаме комплект от 2 данни – дължина и цвят) и тези действия да се повторят толкова пъти, колкото са данните (комплектите данни). С други думи: възможно е действието върху данните да бъде поставено като тяло на цикъл. А употребата на цикъл (каквато и разновидност да се използва) е много приятно за програмиста ☺ - пише само едно! действие (е, понякога група действия) и посочва колко пъти да се изпълни, по веднъж за всяка от данните (комплектите данни). В нашия конкретен случай тялото на цикъла трябва да се изпълни три пъти.

По-долу е показан кодът за решаването на задачата, написан от автора. Използван е един от вариантите за цикъл в езика C++ – езиковата конструкция while. Повторението на тялото на цикъла се прави докато има необработени данни за лентички (`i > 0`) като се следи след обработката на данните за поредната лентичка да се намалява количеството на оставащите необработени данни (`i--`).

Възможни са и други начини за решаване на задачата, освен дискутирания тук и показан в авторовия код. За сега няма да се вглеждаме кой от всичките начини за решение е най-добър.

В заключение ще обърнем внимание върху обстоятелството, че решението на задачата се базира върху сумиране на белите дължини без значение в какъв ред са лентичките само и само защото лентичките са 3. Ако в кръга са повече лентички, тогава гореописаната логика на решението няма да е вярна и ще трябва да се търсят други начини за решение. Но това вече е друга задачка... (може би за друг кръг в Урановата дивизия ☺)

Кодът на програмата, написан на езика C++

```
#include <iostream>
using namespace std;

int main() {
  int length, color,      // дължина и цвят на поредната лентичка в кръга
      length_white = 0,   // дължина на обединения бял участък в кръга
                          // преди да имаме реални данни за лентичките
                          // предполагаме, че няма бели - т.е. дължината му е 0
      i = 3;              // за колко лентички остава да прочетем данни
                          // (първоначално за 3)

  while (i > 0) {
    cin >> length >> color;

    if (color == 0) {
      length_white += length;
    }

    i--;
  }

  cout << length_white << endl;

  return 0;
}
```

Задача 2. Рисувай шах

Ако никога не сте правили рисувални задачки- лошо за Вас. Почваме с най-лесната: да отпечатаме едно редче със звездички. За целта трябва много пъти да отпечатаме една звезидчка:

```
long long size,x;
cin>>size;
x=0;
while( x<size ){
  cout<<"*";
  x=x+1;
}
cout<<"\n";
```

Ако изпълним тоя код (е, може би без създаването на променливите и въвеждането на стойности за тях от клавиатурата) многократно, би се отпечатала последователност от много редове, всеки от тях с по `size` звездички.

Нещо такова:

```
long long sizex,sizy,x,y;
cin>>sizex;
cin>>sizey;
y=0;
while ( y<sizey ){
  x=0;
  while ( x<sizex ){
    cout<<"*";
    x=x+1;
  }
  cout<<"\n";
  y=y+1;
}
```

Просто трябваше да го вложим във външния цикъл – той брои до `sizey`. Но ‘що съм написал два различни брояча? Не можеше ли със само един брояч и за двата цикъла? Вие пробвахте ли? А разбрахте ли си що не бачка? Може би е добра идея да се престорите на комп и да проследите к’во се случва с променливите Ви ред, по ред.

Та, ОК – броячите са `x`, `y` – различни са... С това решихме ли си задачата, а? Трябваше да печатаме квадратче от звездички, а? Или май хич не беше така...Май трябваше всъщност да печатаме не винаги звездички, ами различни символи според зависи къде сме (на кой ред, на кой стълб). Трябва да измислим закономерност определяща `#` или `0` да печатаме. Тогава вместо:

```
cout<<"*";
```

ще можем да напишем:

```
if (намираме се намясто за #){
  cout<<"#";
  }
else{
  cout<<"0";
}
```

Каква ли да е тая зависимост?!?...
Е, можете и да си погледнете в авторския код, ама Ви предлагам да си я измислите сами – нарисувайте си правоъгълниче, вижте какви стойности имат `x` и `y` на различните места и какво общо има по местата с `#` и местата с `0`.

Кодът на програмата, написан на езика C++

```
#include <iostream>
using namespace std;

int main() {
  int sizex,              //размер на дъската по X (брой квадрати в ред)
      sizey,              //размер на дъската по Ъ (брой редове)
      y = 0,              //№ на реда, който се изписва в момента
      x;                  //№ на квадратчето от реда, което се изписва

  cin >> sizex >> sizey;

  while (y < syzey){      //изписва поредния ред от дъската
    x = 0;

    while (x < sizex) {  //изписва поредното поле от реда
      if ((x + y ) % 2 == 0) {
        cout << "0";
```

```
        }
      else {
        cout << "#";
      }
      x++;
    }

    cout << endl;
    y++;
  }

  return 0;
}
```

Задача 3. Почти като при шаха

Нека си представим, че върху всяко квадратче от дъската са поставени платените за него житени зрънца. Ми нека минем с каруцата покрай квадратчетата и да натоварим платеното: В началото остава да товарим от *N* квадратчета, товарим зрънце от първото и остават за товарене още от *N*–1 квадратчета, отиваме и товарим 2 от второто, остават още *N*–2 квадратчета, после от третото товарим 4 и остават още *N*–3 квадратчета... и така докато не натоварим всичко. Значи повтаряме следните действия:

- `Добавяме текущия брой зрънца от квадратче към общата сума`
- `Изчисляваме колко е броят зрънца в следващото квадратче (2 пъти повече от текущото)`
- `Намаляваме с 1 броя на оставащите квадратчета със зрънца`

След като сметнем общия брой зрънца, трябва да преценим колко каруци ще ни трябват: делим на броя зрънца за една каруца (ха сега да видя дали можете да превръщате от тонове в грамове ☺). Полученото е цяло количество каруци пълни „до горе”, понеже в C++ когато делите две цели числа едно на друго се получава винаги цял резултат. И тъй като може да има и остатък от деленето (т.е. ще има и малко зрънца за една частично пълна каруца) трябва в този случай да увеличим с 1 броя на каруците.

До тук нещата не са трудни. Има само една маааалка подробност: заради удвояването на зрънцата във всяко следващо квадратче, броят им много бързо расте. При по-големи стойности за N (над 31) броят на зрънцата за последното квадратче няма да се помести в допустимия интервал за типа `int` – ще трябва да изберете тип на променливата за брой зрънца в квадратчето `long long`, но дори и с него ще се получи грешно пресмятане при N= 64. Тогава остава последният възможен ход – използвайте беззнаково дълго цяло число (`unsigned long long`). При беззнаковите цели числа максималното допустимо число за стойност е два пъти по-голямо от това при цяло число със знак. За сумата на всички зрънца пак трябва да използваме `unsigned long long`, тъй като тя става още по-огромна от броя зрънца в последното квадратче.

(Само за сведение на любопитните: за дъска с 64 квадратчета софтуерната фирма трябва да Ви заплати 18 446 744 073 709 551 615 зрънца! Дори не знам как се произнася това число...)

Кодът на програмата, написан на езика C++

```
#include <iostream>
using namespace std;

int main() {
  unsigned long long
    total_beans = 0,    //Общ брой зрънца за цялата дъска
    current_beans = 1, //Брой зрънца за поредното поле от дъската
  long long
    N,                  //Брой полета върху игралната дъска
    wagons;             //Брой каруци

  cin >> N;

  while (N > 0) {
    total_beans += current_beans;
    current_beans *= 2;
    N--;
  }

  wagons = total_beans / 1000000;  // 1 каруца = 1 тон = 1000 кг = 1000000 г
  if (total_beans % 1000000 > 0) { // може да има и една частично пълна каруца
    wagons++ ;
  }

  cout << wagons << endl;
  return 0;
}
```

Задача 4. Яяя, кюфтенца!

Ясно е, че съм обречен на глад или преяждане (само не знам на кое точно). Проблем обаче (за програмиста, за Вас) е, че може дълго да се боря... хм – тъпча. Явно ще ни е нужно да повтаряме някакви действия, в каквито случаи ползваме цикъл. В условието си е посочено до кога да циклим – докат'

`изядените кюфтета са по-малко от 10000`
И
`броя изядени кюфтета завършва не на 0`

Но как да разберем, че число не завършва на `0`? Ам'чии математиката ни офертира една фитка – ***последната цифра на всяко число е остатъкът му при деление на 10.***

Яко! – тази цифра така или иначе ни трябваше, щото на всяка стъпка на цикъла ни трябва да я прибавяме към общо изядените кюфтета!

И... така – докат' излезем от цикъла.

Тъжно за мен... ☹

А на вас ви остава да отпечатате текущата стойност на общо изядените кюфтета и да си вземете поука за прекаляването с кюфтета.

Кодът на програмата, написан на езика C++

```
#include <iostream>
using namespace std;

int main() {
  int K;
  cin >> K;

  while ((K < 10000) && (K%10 != 0)) {
    K += K%10;
  }

  cout << K << endl;
  return 0;
}
```