„*” (oprócz operacji mnożenia) oznacza zadania [nieco] trudniejsze. (Tym bardziej warto się więc z nimi zmierzyć!)

Przypomnienie/wyjaśnienie: dla naturalnych *a* i *b* napis „*a*%*b*” oznacza w C/C++ resztę z dzielenia *a* przez *b*, czyli to co matematycy i informatycy czasem (nawet całkiem często!) zapisują jako „*a* mod *b*”. Na wszelki wypadek przypomnę też, że z kolei „*a*/*b*” oznacza w C/C++ (dla *a* i *b* naturalnych!) część całkowitą liczby *a*/*b*.

Z1. Jak przebiega działanie poniższych algorytmów? (Ustal, jakie wartości przyjmują kolejno zmienne i co pojawia się na ekranie). Oczywiście warto (później) sprawdzić swoje przypuszczenia na komputerze. (Oczywiście wpisując to wraz z odpowiednimi deklaracjami zmiennych w funkcji *main*, oczywiście dopisując ew. instrukcje, które wyświetlą jeszcze wartości interesujących nas zmiennych).

```
a)
for (i=2019; i>0; i=i-9)
  cout << i;

b)
z=0;
for (i=2019; i>0; i=i-9)
  z=z+1;

c)
for (i=0; i<44; i=i+1)
  if (i%3==0) cout << "Hehe!";
/* Chyba wiadomo juz,
co robi ta instrukcja z "if"?
I co znaczy "=="? - w odroznieniu od "="! */

d)
S=0;
for (i=0; i<100; i=i+1) S=S+i;
// Lepiej nie mylic "s" z "S"
// - to dwie rozne rzeczy!
```

```
e)
a=0;
b=0;
for (i=0; i<1234; i=i+1)
  if (i%3==0) if (i%2==1) a++; else b++;

f)
for (i=1; i<10; i=i+1)
  for (j=0; j<10; j=j+1)
    cout << i << j;

g)
s=0;
for (i=1; i<10; i=i+1)
  for (j=0; j<i; j=j+1)
    s=s+i;

h)
for (i=10; i>0; i=i-1)
  for (j=1; j<10; j=j+1)
    if (j<i) cout << i;
```

Z1’. Czy umiesz **dopisać** coś do algorytmu e), tak żeby wartość *b* rosła przy każdym *i* niepodzielnym przez 3? Przy użyciu tylko elementów C++, które powinniście już znać, daje się to zrobić nawet na dwa sposoby!

Z2. A co wypisze ten programik? (Informatycy mówią raczej „fragment kodu” – sorry!)

```
for (i=1; i<1000; i=i+1)
     cout << "Nigdy nie odpisuję zadań domowych!" << endl;
     cout << "I love computer science!";
```

Zauważ, że ostatnia instrukcja wykona się tylko raz – czyli już po wyjściu z pętli (mimo pięknego przecież wcięcia w zapisie – komputer na estetykę nie jest w ogóle czuły!).

A ten?

```
for (i=1; i<1000; i=i+1)
{
   cout << "Nigdy nie odpisuję zadań domowych!" << endl;
   cout << "I love computer science!";
 }
```

(miejsce na notatki)

Niezłe, co?
Wobec tego zastosuj teraz polecenie Z1 do fragmentów:

```
a)
for (i=1; i<10; i=i+1)
{
  for (j=1; j<10; j=j+1)
    cout << i;
  cout << endl;
}
```

```
b)
for (i=1; i<10; i=i+1)
{
  for (j=1; j<i; j=j+1)
    cout << j;
  cout << endl;
}
```

```
c)
for (i=1; i<10; i=i+1)
{
  cout << i;
  for (j=1; j<6; j=j+1)
    cout << j;
  cout << i;
  cout << endl;
}
```

Z2,8. I odwrotnie – spróbuj (pętlami!)
wygenerować na ekranie coś takiego:

```
d)
5
45
345
2345
12345
```

```
e)
12345
23451
34512
45123
51234
```

```
ę)
54321
45432
34543
23454
12345
```

Z3. Napisz program znajdujący sumę liczb nieparzystych dodatnich mniejszych od *n* (gdzie *n* wczyta się od Użytkownika („cin")).

Z4. Napisz program wypisujący dla danych liczb całkowitych *a* i *b* (zakładamy, że $a \leq b$) w kolejności malejącej wszystkie wielokrotności siódemki z przedziału (*a*, *b*) i pisze, ile ich jest.

Z5. Napisz program, który pozwoli Ci się wreszcie dowiedzieć, dla których naturalnych liczb dwucyfrowych wyrażenie x^3-77x^2+1001 ma wartości niedodatnie.

Z6. Napisz program, który dla danej całkowitej dodatniej liczby *k* wyświetli na ekranie *k* kolejnych potęg naturalnych liczby *k*, oddzielając je przecinkami.
Np. dla *k*=6 powinno się wyświetlić: „1, 6, 36, 216, 1296, 7776" (bez przecinka na końcu!).

Z7. Napisz program obliczający sumę $1^2 - 3^3 + 5^2 - 7^3 + ... - 2019^2$.

Z8. Napisz program, dzięki któremu będziesz być może pierwszym człowiekiem, który dowie się, jaka jest najmniejsza liczba trzycyfrowa spełniająca warunek $x^2 > 123x+999999$.

Z9. Napisz programik, który ustali, czy dane *x* jest doskonałe.
(Doskonałe są np. 6 (=1+2+3) i 28 (=1+2+4+7+14), bo (DEFINICJA!)
Liczba naturalna *n* jest doskonała. ⇔ *n* jest sumą swoich dzielników mniejszych od siebie (tzw. swoich dzielników *właściwych*).).

Z9,8. Znajdź kolejne liczby doskonałe.

Z9,C. Napisz program, który wypisze *n* (*n* podaje użytkownik) najmniejszych liczb doskonałych. (Wystarczy, że będzie działał dla *n*<=4, bo już przy *n*=5 czas jego działania... – no właśnie: potrafisz ten czas oszacować? (Zad. nieobowiązkowe (możliwa gratyfikacja!), rozwiązania (pełne rozumowanie + rachunki) proszę przesyłać mejlem. Podpowiedź: z sensownym przybliżeniem (całkiem dobrym przy dużych x) można przyjąć, że czas ten jest proporcjonalny do liczby wszystkich obrotów wewn. pętli. (Wiesz dlaczego?))

ZA. (Święta za pasem) Co wypiszą programy: (" " to łańcuch (tekst) będący spacją – odstępem, "*" – to gwiazdka)

```
a)
for (i=44; i>0; i--)
{
  for (j=1; j<i; j++) cout << " ";
  for (j=i; j<44; j++) cout << "*";
}

b)
for (i=1; i<=4; i++)
   for (j=1; j<=4; j++)
   {
      for (k=j; k<=3; k++) cout << " ";
      for (k=1; k<=2*j; k++) cout << "*";
      cout << endl;
   }
cout << "   **" << endl;
```

```
c)
for (i=2; i<=6; i++)
   for (j=1; j<=i; j++)
   {
      for (k=j; k<=5; k++) cout << " ";
      for (k=1; k<=2*j; k++) cout << "*";
      cout << endl;
   }
cout << "     **" << endl;

d)
for (i=1; i<=16; i++)
{
   j=i%4;
   if (j==0) j=4;
   for (k=j; k<=3; k++) cout << " ";
   for (k=1; k<=2*j; k++) cout << "*";
   cout << endl;
}
cout << "   **" << endl;
```

ZB. („O wdowim groszu") W roku narodzin Chrystusa pewna wdowa złożyła 1gr na konto oprocentowane na 1% w skali roku. Ile złotych będzie miał na tym koncie w nadchodzącego Sylwestra prapraprapraprapraprapraprapraprapra-prapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprapra-prapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprapraprawnuk tej pani? Napisz program, który to obliczy, używając najwyżej 2 zmiennych. (Wynik może zostać wyświetlony w postaci wykładniczej i wiedząc to, powinnaś/-nieneś ją rozumieć!)
Tak rośnie funkcja wykładnicza...

ZC. W którym z powyższych zadań wygodnie/sensownie jest użyć pętli *while*?

ZD. Co powinien wypisać program:
```
a=123045600789;
while (a>0) { cout << a%10;  a = (a-(a%10))/10; }
```
?

D'. Jak zadziałałby ten program, gdyby początkową wartością *a* była liczba:
a) 54321, b) 10, c) jednocyfrowa, d) 0?

D''. Jak można w C++ (znacznie) uprościć wartość w ostatnim podstawieniu?

ZE. Zapisz pętlę *for (i=0; i<77; i++) for (j=1; j<10; j++) cout << i << j;*, używając [podwójnej] pętli *while*.

Uwaga: w poniższych zadaniach należy nie znać potęgowania w C++!

ZF. Napisz programi[cze]k, który po podaniu przez użytkownika naturalnych *a* i *b* (Załóżmy spokojnie, że $a<b$ i że nie będziemy musieli próbować obliczać 0^0, czego matematycy raczej nie definiują.) wyświetli

a) $x^a, x^{a+1}, x^{a+2}, ..., x^b$ (gdzie *x* jest zadaną liczbą rzeczywistą).

b) $\frac{1}{a!}, \frac{1}{(a+1)!}, \frac{1}{(a+2)!}, \ldots, \frac{1}{b!}$ (co oznacza „!" (w tym „0!"!), należy się w razie potrzeby dowiedzieć).

c) $a^{11}, (a+1)^{11}, (a+2)^{11}, ..., b^{11}$.

d) $a^a, (a+1)^{a+1}, (a+2)^{a+2}, ..., b^b$.

Z10. Napisz funkcję *liczbadzielnikow*, która jako wartość będzie dawać – uwaga – liczbę naturalnych dzielników jej argumentu (który jest liczbą całk. dodatnią), czyli np. *liczbadzielnikow*(4)=3, *liczbadzielnikow*(13)=2.

Z11. Sprawdź, czy dana liczba jest pierwsza (polecam zastosować funkcję z Z10), chociaż można inaczej). Znajdź:
11.a) *k*-tą liczbę pierwszą / *k* najmniejszych liczb pierwszych;
11.b) najmniejszą liczbę pierwszą większą od ...;
11.c) wszystkie liczby pierwsze z przedziału ...

Z12. Napisz program ustalający, czy
a) podany przez użytkownika rok jest przestępny.
UWAGA: Rok *k* jest przestępny. <=> $(4|k \wedge (100|k \Rightarrow 400|k))$
b) podana przez użytkownika data jest możliwa. Tj. użytkownik podaje 3 liczby naturalne – dzień, miesiąc i rok, a program pisze „możliwe" / „niemożliwe". (Np. niemożliwe są daty „31 4 2017", „30 2 2018", „29 2 2100", „32 cokolwiek cokolwiek", „5 13 2000"). Spróbuj to zrobić, nie rozwlekając niepotrzebnie sprawdzeń.

Z13. Napisz program znajdujący
a) pierwszą całkowitą potęgę dwójki przekraczającą milion;
b) największą liczbę naturalną, której silnia nie przekracza 1.000.000.000;
c) $\max\{n: 1/1+1/2+1/3+...+1/n < s\}$, jeśli *s* jest dodatnią liczbą podaną przez użytkownika.

Z14. Napisz program[ik], który:
a) dla wczytanych naturalnych *a* i *b* (zakładamy spokojnie, że $a<b$) wypisze w kolejnych linijkach niepodzielne przez 3 wielokrotności piątki z przedziału* $[a, b)$, numerując je kolejno;
(Czyli np. dla $a=3$ i $b=20$ powinno się wyświetlić: 1. 5
2. 10).

b) stwierdzi, ile liczb z przedziału* $[12, k]$ (*k* podaje użytkownik) ma nieparzystą przedostatnią cyfrę;

c) będzie reagował na wprowadzane przez użytkownika liczby naturalne, pisząc „nie", jeśli jest ona podzielna przez 3, a „OK" i kończąc wczytywanie, jeśli nie;
(Czyli np. może przebiec tak o:
12
nie
27
nie
6
nie
12
nie
13
OK).

d) dla podanego *x* rzeczywistego znajdzie sumę naturalnych potęg dwójki, które nie przekraczają *x*, tj. np. dla $x=8$ obliczy (i wypisze) wartość $2^0+2^1+2^2+2^3$;

e) wypisze wszystkie pary liczb całkowitych *x* i *y* z przedziału $[-100,100]$* spełniające równanie $x^3=xy+y^2$;

* Przy oznaczaniu przedziałów nawias kwadratowy oznacza to samo co trójkątny – domkniętość z danej strony.

f) zliczy pary dwucyfrowych liczb *a* i *b*, suma sześcianów których jest mniejsza od 12345;

g) zliczy, ile jest sześciocyfrowych (= z przedziału [100000, 999999]) sześcianów liczb naturalnych;

h) wypisze wszystkie liczby trójkątne z przedziału (*a*, *b*), oddzielając je przecinkami (ale nie stawiając oczywiście przecinka po ostatniej!); (Czyli np. dla *a* = 3, *b* = 15,2 napisze „6, 10, 15”).
(Liczby trójkątne to sumy kolejnych liczb naturalnych, czyli kolejno 1 (=1), 3 (=1+2), 6 (=1+2+3), 10, ... Ponieważ stanowią one analogię silni (dlaczegóż?!!), dowcipni informatycy nazywają je czasem słabniami i oznaczają: 1? (=1), 2? (=3) itd.).

i) dla podanych całkowitych *a* i *b* wypisuje wyrażenie *a**(*a*+1)+(*a*+1)*(*a*+2)+…+(*b*-1)**b*, liczby ujemne otaczając nawiasami, znak równości i jego wartość, tzn. np. dla *a*=–2, *b*=1 napisze „(-2)*(-1)+(-1)*0+0*1=2”;

j) znajdzie wszystkie trójkąty pitagorejskie o wszystkich bokach mniejszych od 1000;

k) dla podanych *m* i *n* wyświetla coś, co dla (*m*, *n*) = (3, 5) wygląda tak:
```
3 6 9 12 15
2 4 6 8 10
1 2 3 4 5
```
;
```
0
12
345
6789
01234
```
l) dla podanego *n* wyświetli *n* wierszy piramidki, która dla *n*=5 wygląda tak o: (w kolejnych wierszach dopisywane są cyklicznie kolejne cyfry);

ł) dla podanego naturalnego *n* drukuje w kolejnych liniach kolejno: kolejną liczbę naturalną $i \in [1, n]$ i najmniejszą liczbę naturalną, której silnia przekracza *i*. Aspirujący do uzyskania mojej pochwały niechaj spróbują zrobić to, nie zmuszając komputera do liczenia kolejno kolejnych silni;

m) taką ramkę:
```
111111
2    2
3    3
4    4
5    5
666666
```
(a ściślej jej odpowiednik dla parametru *n*, którego wartość (mniejszą niż 10) poda użytkownik – w przykładzie po lewej *n*=6 – to chyba jasne?)

n) … i taką piramidkę jak po prawej (również sparametryzowaną):
```
    1
   121
  12321
 1234321
```

Z15. Znajdź rozkład na czynniki pierwsze danej liczby.

Z16. Wypisz (elementami, może być bez spacji i przecinków) wszystkie 4-elementowe podzbiory (każdy tylko raz!) zbioru {0, 1, 2, ..., 9}. Tzn. program może wypisać kolejno: 0123, 0124, 0125, ..., 6789 i w ciągu tym nie pojawi się np. napis 4012 (bo oznacza on ten sam zbiór, co wypisany już jako drugi).

Z17*. Napisz program, który po wczytaniu liczb naturalnych *n* i *p* poda zapis *n* w systemie o bazie *p*. Czyli np. dla *n*=1234 i *p*=5 napisze 14414. W wersji alfa program może działać tylko dla *p*<10, ale w wersji beta powinien też radzić sobie z większymi *p* („cyfry dwucyfrowe” najłatwiej chyba pisząc w nawiasach, tzn. np. największą cyfrę w systemie dwunastkowym oznaczając przez „(11)”).

Z18. Pola niekoniecznie kwadratowej szachownicy opisujemy współrzędnymi (*x*,*y*), gdzie *y* jest numerem wiersza, a *x* – kolumny, przy czym lewy dolny róg szachownicy to (1,1), a współrzędne rosną w prawo i do góry. Na pewnym polu stoi goniec, który znienacka rusza na wycieczkę, poruszając się na ukos do góry w prawo, a po dojściu do każdej krawędzi odbijając się od niej zgodnie z prawem odbicia. Napisz program, który wczyta liczby kolumn i wierszy szachownicy oraz współrzędne początkowego położenia gońca, a w odpowiedzi poda jego trasę do momentu zamknięcia cyklu, który następnie zacznie się powtarzać. Np. dla danych 3, 6, 1, 3 powinno się wypisać: (2,4), (3,5), (2,6), (1,5), (2,4), (3,3), (2,2), (1,1), (2,2), (3,3), (2,4), (1,5), (2,6), (3,5), (2,4), (1,3), (2,2), (3,1), (2,2), (1,3).

Z19*. Napisz program podający wszystkie trójkąty pitagorejskie, których jeden z boków ma długość będącą podaną liczbą naturalną. Przykładowa reakcja: dla podanej długości 15 wypisuje 9, 12, 15; 15, 20, 25 oraz 15, 112, 113, a dla podanej długości 2 wypisuje „Nie ma!”.

Z1A. Zapisz (w C++!) definicję funkcji *NWD* („return” na końcu – co powinna zwracać?). Użyj jej w programie, w którym użytkownik poda dwie liczby naturalne, a program powie mu, czy są względnie pierwsze (czy nie).

Z1B. Dane są dwie liczby oraz ich NWD (ktoś miły obliczył). Jak bez rozkładania czegokolwiek na czynniki znaleźć ich NWW?

Z1C(p). A jak można użyć dwuargumentowej funkcji *NWD* (obliczającej – nie zgadniesz! – NWD dwóch liczb naturalnych) do znalezienia NWD trzech danych liczb? (Tzn. zakładamy np., że za pomocą komputera potrafimy (nieważne jak) ustalić NWD dwóch liczb, i pytanie, jak dzięki temu znaleźć NWD jakichś trzech liczb).

Z1D. Czy jeśli trzy liczby są względnie pierwsze, to każde dwie z nich muszą być względnie pierwsze? A odwrotnie?

Z1E. Dla jakich *z* zachodzi: x) (*z* shr 1) shl 1 = *z* ? y) (*z* shl 1) shr 1 = *z* ?

Przy Z1F-21 (ale nie tylko!): pomocą służy WolframAlpha.com! A w Z20 i 21 można też wspomóc się własnoręcznie napisanymi pętlami!! (W C++!!!) Tylko jakie mogłyby to być pętle w Z21??

Z1F. Podaj wyniczki:
a) 2^{1024} or 10^{123}
b) 2^{1024} and 2^{123}
c) 2^{1024} xor 2^{123}
d) 7^{777} and 1
e) *x* or *x*
f) *x* xor *x*
g) *x* and *x*
h) 2^{100} xor 512
i) 1023 xor 345
j) $(2^{2019}-1)$ xor 1234567890
k) *x* and 0
l) *x* or 0
ł) 0 xor *x*
m) 123765965123 and 8076238076 and 132 and 3218907 and 68
n) (255 and 7654321) or ((255·256) and 7654321) or ((255·256^2) and 7654321)
o) (123456789 or 1026) and (123456789 or 514) and (123456789 or 258)
p) ((80763240787 or (32478948978 xor 248761434 xor 123967324966)) and 3
r) 123456789 xor 132765132495 xor 123456789
Ś) (1234567 and 7654321) xor (1234566 and 7654321)

Z20. Ile wyniosą *x*, *y* i *z*, jeśli w miejsce ☺ wstawimy: and/or/xor?
x = 1☺2☺4☺…☺1024
y = 1☺2☺3☺…☺1023
z = 1☺3☺5☺…☺1023

Z21. Jakie może być *x*, skoro:
a) 2^{123} or *x* = 2^{123} ?
b) *x* xor 2*y* = 1 ?
c) *x* and *x* = 178 ?
d) *x* or 17 = 1 ?
e) *x* xor 2 > 2 ?
f) 111 and *x* = 111 ?
g) 12345 or *x* > 12345 ?
h) *x* and 54321 > 54321 ?
i) *x* xor 1024 = *x* ?
j) *x* xor 1023 = 1023 or *x* ?
k) (*x* or 1) xor *x* = 0 ?

Z22. Jakimi operacjami bitowymi na liczbie *x* można:
a) znaleźć jej resztę z dzielenia przez 4?
b) zmienić jej bit czwórek?
c) znaleźć jej siedemnastą od końca cyfrę dwójkową?

Z23. Ostatnio mi się przyśniło, że domek zbuduję aż miło, gdzieś na Ziemi Szczecińskiej, domek dla Pani Śliwińskiej.
By oświetlić salę balową, instalację mam już gotową: wokół sali, na tarasie, 100 włączników da się.
Uwagę zrobię tu szczerą – każdy w pozycji jeden być może lub zero.
Instalację całą dawno podłączyłem, wszystkie włączniki w pozycji 1 ustawiłem,
zda się jednak, że nadaremno – w sali bowiem wciąż jest ciemno!
Lecz jeśli którykolwiek włącznik teraz się przestawi, światło w sali się pojawi!
Każdym kolejnym przestawieniem stan światła w sali zmienię
i nie ma przy tym znaczenia, pozycję którego włącznika zmieniam.
(Dowolnie zatem pstryk, pstryk, pstryk i światła ni ma, jest i znów ni ma w mig!)
Instalację zatem mam magiczną – realizuje pewną funkcję logiczną
(w zależności od stanów włączników oczywiście), więc śmiało myślcie i piszcie,
jakaż to funkcja, Kochani – o to jesteście właśnie pytani!

Z24. W długi weekend Zenek ułożył tabelkę 100×100, wypełniając jej pola poza ostatnim wierszem i ostatnią kolumną zerami i jedynkami. Ostatnią kolumnę poza ostatnim wierszem wypełnił tak, żeby suma liczba w każdym wierszu była parzysta, a ostatni wiersz poza ostatnim polem tak, żeby w każdej kolumnie była parzysta liczba jedynek. Zadowolony z siebie Zenek zasnął, a wtedy czyhający na taką gratkę złośliwy gnom zanegował jeden bit w jego tabliczce…
Mimo to po przebudzeniu Zenek był w stanie odtworzyć oryginalny układ bitów. Jak to zrobił? (Zenek zdecydował nie używać czarów, tylko rozumu). Co to ma wspólnego z xorem?
A gdyby gnom zanegował dwa bity? (Co Zenek byłby w stanie stwierdzić?)
A gdyby wiedział tylko, że gnom zanegował maksymalnie dwa bity?

Z25*. Jaś i Małgosia znów pojmani przez straszną wiedźmę! Tym razem mogą ocalić życie, jeśli uda im się wygrać następującą grę: na polach szachownicy 8×8 wiedźma położy monety, każdą dowolnie – orłem lub reszką, następnie pokaże Jasiowi tę szachownicę i wskaże mu jedno pole. Jaś może obrócić dowolną z monet. Potem wiedźma zamknie Jasia w komórce i wypuści oboje dzieci, jeżeli Małgosia odgadnie (za jednym razem), które pole wiedźma wskazała Jasiowi. Jaką strategię powinny przyjąć dzieci? (Muszą ją ustalić przed zobaczeniem przez Jasia szachownicy!)

Istnieje taka, która gwarantuje im wolność, i to bez używania żadnych czarów, naprawdę!
Podpowiedź: jak myślisz – czego Jaś i Małgosia uczyli się niedawno na lekcjach informatyki?

Z26**. *W szponach hazardu albo: Kolejne nieoczekiwane zastosowanie alternatywy wykluczającej!*
W grze „nim”♥ dysponujemy pewną liczbą patyczków, które przed rozpoczęciem gry rozkładane są na kilka stosików. Gracze wykonują ruchy na przemian, a ruch polega na wyborze dowolnego niepustego stosiku i odłożeniu z niego dowolnej (dodatniej) liczby patyczków. Przegrywa gracz, który odkłada ostatni patyczek. Czy któryś z graczy może zapewnić sobie wygraną? Czy zawsze? Jak powinien grać? A co by było, gdyby wygrywał gracz opróżniający ostatni niepusty stosik?
Do znalezienia strategii pomocne jest zastanowienie się nad następującymi własnościami bitowej operacji *xor*:
- czy jeśli x_1 xor x_2 xor ... xor $x_n = 0$, to jakakolwiek zmiana jednego z argumentów zmieni wynik?
- czy jeśli x_1 xor x_2 xor ... xor $x_n \neq 0$, to zawsze da się tak zmienić któryś z argumentów, że wynik się zmieni? (W grze zmiana ta nie może być w dodatku całkiem dowolna!)
Nie muszę chyba dodawać, że zapis sytuacji w grze wygodnie jest kodować zapisami dwójkowymi odpowiednich liczb.

Z27. Zdefiniuj funkcje:
ilelat(*n*), zwracającą liczbę pełnych lat pozostałych do zakończenia wieku zawierającego rok *n* w roku *n*,
min(*a*,*b*,*c*,*d*), zwracającą min(*a*,*b*,*c*,*d*).

Z28. Napisz program, który wypisze coś takiego (ładniej pewnie będzie w kolejnych wierszach): F0=0, F1=1, F2=1, ... (np. do F_n, gdzie *n* raczy podać użytkownik (ang. „user”)). Program niech używa zdefiniowanej przez nas ostatnio funkcji (chociaż wiemy (? :>) już, że może się to okazać niezbyt roztropne). Przy jakich wartościach zaczyna wymagać to od odbiorcy sporej dozy cierpliwości?

Z29. Przypominam: żeby obliczyć F_3, nasza funkcja zostanie wywołana 5 razy. A ilu wywołań wymaga obliczenie F_5? Narysuj odpowiednie drzewko. A F_6? A F_7?
Oznaczmy te liczby przez w_n, tzn. niech w_n oznacza liczbę wywołań naszej funkcji przy obliczaniu F_n. Potrafisz podać zależność rekurencyjną dla w_n?

Z2A. Ciągiem Tribonacciego nazywa się ciąg zdefiniowany podobnie jak Fibonacciego, ale taki, gdzie każdy kolejny wyraz jest sumą trzech poprzednich. Zdefiniuj w C++ funkcję *T*(*n*), która będzie zwracać *n*-ty wyraz ciągu Tribonacciego, przyjmując dowolne warunki początkowe (tzn. dowolne wartości, ale definiując ich dokładnie tyle, ilu potrzeba!).
Ile wywołań tej funkcji nastąpi przy obliczaniu *T*(4)? A *T*(5)? A *T*(6)? Ile czasu i miejsca zajmie Ci narysowanie odpowiednich drzewek wywołań w zeszycie?

Z2B. Zdefiniuj (w C++) funkcję *Fi*(*n*), która jak *F*(*n*) będzie zwracać *n*-ty wyraz ciągu Fibonacciego, ale obliczy go pętlą. (Taki sposób nazywamy iteracyjnym). Użyj tej funkcji tak samo jak *F*(*n*) w zad. 29. Jak dużo wyrazów ciągu Fibonacciego można teraz śmiało obliczać?

Z2C. Zdefiniuj ciąg spełniający zależność rekurencyjną: $a_n = \begin{cases} {a_{n-2}}^2 - 1, & n \le 5 \\ 3a_{n-3} - a_{n-1}, & n > 5 \end{cases}$. Warunki początkowe zadaj dowolnie, ale sensownie (nie za dużo i nie za mało). Narysuj drzewko wywołań tej funkcji przy obliczaniu a_8.

Z2D. Zdefiniuj numerowany od 1 ciąg spełniający następującą zależność rekurencyjną:
$x_n = \begin{cases} x_{n-3} - x_{\frac{n}{5}}, & 5|n \\ {x_{n-2}}^2 + x_1 & \text{w przeciwnym razie} \end{cases}$. (Warunkami początkowymi zajmij się jak w Z33).
Narysuj drzewko wywołań tej funkcji przy obliczaniu x_9. W Pascalu i innych językach programowania (ale nie C++) dostępna jest funkcja *sqr* obliczająca kwadrat argumentu. Na co będzie miało wpływ jej użycie?

Z2E. Zdefiniuj ciąg określony wzorem: $a_n = \begin{cases} 2a_{n-1} & \text{dla } n \text{ nieparzystych podzielnych przez } 3 \\ -n/3 & \text{dla } n \text{ parzystych} \\ 7 & \text{w pozostałych przypadkach} \end{cases}$.
Podanie jakich argumentów spowoduje najwięcej wywołań tej funkcji?

♥ prawdopodobnie pochodzenia chińskiego, w Europie znana już co najmniej w XV w.; nazwa pochodzi prawdopodobnie ze staroangielskiego albo niemczyzny (por. też biorącą się z tego samego nazwę znanych cukierków do ssania z witaminami)

Ż0. Żółw Hieronim uwielbia wykonywać procedurę *acoto* o argumencie *n* zdefiniowaną rekurencyjnie:

jeśli $n>0$, pójdź do przodu o 44 kroki, obróć się w prawo o 90° i wykonaj *acoto*(n–1).

Po jakim torze będzie spacerować Hirek przy wykonywaniu polecenia *acoto*(*n*) dla różnych *n*?

Ż1. Oto powyższa procedura zapisana w języku,
w którym można się z Hirkiem dogadać:

```
def acoto(n):
  if n>0:
    forward(44)
    right(90)
    acoto(n-1)
```

A tutaj inna:

```
def spacer(n):
  if n>0:
    forward(n%3+10)
    spacer(n-1)
```

Naszkicuj tor ruchu Hieronima przy wykonywaniu polecenia *spacer*(11*k*), gdzie *k* jest Twoim nrem w dzienniku..

Ż1'. Zachęcam do zabawy z żółwiem online! (Można też weryfikować swoje hipotezy).

Gdyby zachciało Ci się wpuścić Hirka w pętlę, trzeba wpisać coś typu *for i in range(a, b):*

i pod tym nagłówkiem wpisać ciało pętli, koniecznie z wcięciem, np. tak:

```
for i in range(a, b):
  forward(44)
  right(45)
```

- wówczas dwie ostatnie instrukcje wykonają się dla kolejnych $i \in [a, b)$.

Ż2. A co rysują te procedury?

```
def rekurencja(n,a):
  if n>0:
    forward(a)
    right(90)
    rekurencja(n-1,a*2)
```

```
def rekursja(n,a):
  if n>0:
    forward(a)
    right(90)
    rekursja(n-1,a*.9)
```

Jak zawsze chciałbym, żebyście najpierw poMYŚLELI, a potem można popróbować, co będzie robić żółw dla małych *n* i odpowiednich *a*. (Jakie są odpowiednie, również można (warto!) przeMYŚLEĆ, a można też popróbować. Szybko stanie się chyba jasne, że aby rysunek wyszedł sensowny, nie mogą być za duże / za małe. :))

TU MIAŁEM DOPISAĆ JESZCZE TROCHĘ ĆWICZEŃ DO TRENINGU Z ŻÓŁWIEM…

I może jeszcze dopiszę, zwłaszcza jeśli ktoś się upomni!

POTEM BYŁY DZIESIĄTKI ZADAŃ DOT. SORTOWANIA… 😊

A TERAZ NOWY ROZDZIAŁ – *Klasa II (Cp)*!

2.1. Po spektakularnym sukcesie w ogrodzie Hesperyd Herakles dostał zamówienia na więcej złotych owoców. C++ nie ma przed takim herosem tajemnic, więc dzięki informacjom pozyskanym od Nereusa stworzył on spis wszystkich jabłek w postaci dwóch tablic: *m*[1] oznacza masę pierwszego jabłka, a *w*[1] – jego wartość (tzn. za ile może je sprzedać – większe nie znaczy zawsze droższe!), *m*[2] i *w*[2] – odpowiednio masę i wartość drugiego itd. Heros spieszy się, żeby nie ubiegł go Atlas, więc nie nauczył się żadnego algorytmu sortowania, ale postanowił zabierać kolejno jabłka według wartości (czyli jako pierwsze wziąć to, za które dostanie najwięcej, itd.), przy czym nie jest w ciemię bity i dla owoców o tej samej wartości kolejno wybiera oczywiście... Wszystkie jabłka zamierza zapakować do szmacianego worka na owoce z metali szlachetnych, ale że ma on już kilkaset lat, porwie się, gdy zapakuje się do niego ciężar większy niż 64 (przypadek?!).

Przemyśl funkcję, która dla danych tablic *m* i *w* poda przychód herosa w razie powodzenia misji.
Uwaga: wyobraź sobie, że jesteś Heraklesem, i wykonaj to zadanie bez sortowania!
A czy w ogóle opłaca się tu sortować (w sensie złożoności)?

Czy podejście Heraklesa (tzw. *zachłanne, άπληστος αλγόριθμος*) gwarantuje, że przychód będzie największy?
A gdyby robić tak, jak wymyślił Atlas – pakować jabłka wg rosnącego ciężaru (czyli najpierw najlżejsze, potem…, spośród kilku o tej samej masie wybierając oczywiście kolejno…)?
A może heros Jeremeus ma jeszcze inny pomysł? A groźne boginie Nina i Olga?

2.2. Napisz funkcję: *CHAROWANIE*

a) *ilecyfr* – która poda liczbę cyfr arabskich w podanym jej jako argumencie napisie (np. *ilecyfr*("Agent007") = 3, *ilecyfr*("3 maja") = 1);

b) *jest* – która stwierdzi, czy dany znak występuje w danym napisie (np. *jest*('0',"Agent007") = 1);

c) *podnapis* – która stwierdzi, czy drugi napis jest podnapisem pierwszego, czyli np. *podnapis*("informatyka","infa") = 0, *podnapis*("informatyka","tyka") = *podnapis*("informatyka","ma") = *podnapis*("informatyka","informa") = 1;
napisz ją, nie używając gotowej funkcji bibliotecznej, ale możesz też jej poszukać i użyć w rozwiązaniu alternatywnym;

d) *pasuje* – która stwierdzi, czy dany napis pasuje w kratki krzyżówki podane jako drugi argument, gdzie puste kratki oznaczymy kropkami, czyli np. *pasuje*("FOKA","..FA") = *pasuje*("INFA",".....") = *pasuje*("INFA","...FA") = 0, *pasuje*("INFA","..FA") = *pasuje*("FIFA","..FA") = *pasuje*("FOKA","…A") = *pasuje*("FOKA",".OK.") = 1;

e) *palindrom* – która stwierdzi, czy podany jej jako argument łańcuch jest palindromem. Można założyć, że składa się z samych małych lub samych wielkich liter, ale ambitni mogliby zająć się nawet palindromicznymi zdaniami („Kobyla ma maly bok.”, „Wol utyl i ma mily tulow.”), czyli łańcuchami, w których należy zignorować wszystkie znaki nieliterowe (natomiast można nie używać polskiej diakrytyki), na początek możesz przyjąć, że wszystkie użyte litery są duże i ew. że poza nimi występują tylko spacje (ale może czasem wielokrotne!). Zadbaj o złożoność!

ę) *palindrom* działającą dla argumentu typu *int*, która sprawdzi palindromiczność zapisu dziesiętnego liczby naturalnej;
napisz ją, używając tylko typów liczbowych, chociaż możesz też (dodatkowo!) spróbować zrobić to za pomocą pewnej funkcji bibliotecznej;

f) *anagramy* – która stwierdzi, czy podane jej jako argument dwa łańcuchy są anagramami.

2.3. Napisz a) funkcję, b) procedurę, która:
I) odwraca podany jako argument napis (czyli np. z „ANETA” powinna wyjść „ATENA”); zrób to, zmieniając tylko daną zmienną, bez żadnych pomocniczych!
II) zmienia w danym napisie wszystkie małe litery na wielkie. (Tzn. z „ABra qa-d’AbRa” zrobi „ABRA QA-D’ABRA”).

2.4. Wieloletnie prowadzone w Internecie badania dowiodły niezbicie, że 102 jest kodem ASCII znaku ‘f’. Co się tu kolejno wyświetli?

```
char z='g';
cout << int (z-1);
z++;
cout << char (z);
cout << char (z-'f');
cout << char (z+'P'-'p');
```

2.5. Co wypisze w C++ polecenie `cout << int ('4'+'5'-'6')`?

2.6. Podaj popularne formaty plików graficznych. Które z nich zapisują grafikę rastrowo?

2.7. Czym różni się format bmp od jpg, a co mają wspólnego?

2.8. Co jest trudniejsze (i dlaczego!): konwersja grafiki pikselowej na wektorową czy odwrotna?

2.9. Do jakiego typu obrazków grafika wektorowa nadaje się szczególnie wyśmienicie, a do jakich nadaje się raczej nieszczególnie? Dlaczego?

2.A. Dany jest obrazek flagi Japonii (czerwono koło na białym tle). Czy zajmie on więcej, czy (mniej więcej?) tyle samo, jeśli: a) tło zmienimy na zielone?
b) pod tym kółkiem dopiszemy na czarno kilka japońskich znaczków, np.: けいさんきかがく?
Podaj odpowiedzi a) i b) dla zapisu: 0. rastrowego, 1. wektorowego.

2.B. Ile bitów trzeba przeznaczyć na piksel w modelu RGB, jeśli poziomów czerwieni ma być 64, a zieleni i błękitu po 32?

2.C. Ile w przybliżeniu zajmie bitmapa 500×1000 px zapisana w modelu o 256 kolorach? A czarno-biała?

2.D. Ile pikseli można zapisać na 1 MB, jeśli zapisujemy bitmapę o 16 kolorach? A czarno-białą?

2.E. Zapisujemy kolory w modelu RGB, przeznaczając po 5 b na składową.
a) Ile kolorów mamy do dyspozycji? b) Ile jest odcieni czerwieni?
c) Opisz cechy koloru, który zostanie zapisany jako: c_1 = (2, 2, 2), c_2 = (0, 12, 0), c_3 = (2, 31, 0).
d) Ile mniej więcej pamięci zajmuje bitmapa 1000×2000 px?

2.F. Ile jest odcieni czerwieni, jeśli stosujemy model RGB: a) 24-bitowy?
b) w którym maksymalna biel to (15, 63, 15)?

3.0. Jakie są wartości RGB kolorów z tabelki?

liczba bitów na składową / kolor	po 8	po 4	3-2-2
czerwony			
zielony			
czarny			
biały			

3.1. Ustaw piksele w kolejności od najjaśniejszego do najciemniejszego: (0, 0, 0), (255, 255, 255), (3, 5, 5), (255, 5, 5). Czy da się powiedzieć, jakiego są koloru?

3.2. o to jakieś działanie binarne. Zapisz wyrażenie:

x) o o o *a b* o *c* o *d* o *e f g* a) jako drzewo, b) infiksowo, c) w ONP.

y) *a b c* o *d* o *e* o *f g* o o o a) jako drzewo, b) nawiasowo, c) w NP.

3.2'. Co by było, gdyby jedno z kółeczek w którymś z powyższych wyrażeń oznaczało działanie trójargumentowe?

3.2". A gdyby jedno oznaczało działanie trójargumentowe, a inne jedno-? (Czyli byłoby np. o o_1 o *a b* o_3 *c* o *d* o *e f g*

albo *a b c* o *d* o_3 *e* o *f g* o o_1 o). Pobaw się różnymi układami! Które nie są poprawnymi wyrażeniami?

3.3. Nic nie upraszczając, zapisz wyrażenie –(–*a*–(–(*b*–(–*c*)))) a) jako drzewo, b) przedrostkowo, c) przyrostkowo.

3.4. Załóżmy, że C++ rozmieści kolejno definiowane zmienne w pamięci tak, jak obserwowaliśmy to na lekcji (* przypomnienie niżej). Przypomnij sobie lub dowiedz się (najlepiej z dokumentacji C++!), jak działa procedura *sort*, i odpowiedz, jaki będzie efekt wykonania instrukcji:

α) `sort(t,&t[2]);` β) `t[3]=3;` γ) `sort(t,&a);` δ) `sort(&a,t);`
– jeśli w programie mamy wcześniej: `int a=1, b=5, c=2, d=8, t[]={6,4,7};`

* Tzn., że instrukcja `cout<<&a<<&b<<&c<<&d<<t;` spowoduje wypisanie np. `0x61fe1c0x61fe180x61fe140x61fe100x61fe04`.

3.5. Jakie wartości oznaczają (jeśli w ogóle mają sens): *t*, &*t*, &**t*, *&*t*, *(*t*+3), *t*[-1], **a*, *t*-1, *(*t*-2), *p*, &*p*, *(*p*+6), ... pod wykonaniu instrukcji:

```
int t[4]={11,12,13,14}, a=123; int*p; p=&t[0]; p=p-1; *(p+4)=-44; t[5]=77; ?
```

Eksperymentuj z C++, pamiętając, że niektóre wartości są przypadkowe, no i że niektóre operacje mogą spowodować błąd systemu (dlaczego?) – przed ich ew. wykonaniem zapisz zatem lepiej z uruchomionych programów wszystko, czego wolał[a]byś nie stracić!

3.6. Jak przekonać się, ile pamięci zajmuje zmienna danego typu (*double* albo np. wskaźnik)? A jeśli nie znalibyśmy funkcji *sizeof*? I nie polegajmy na założeniu, że kolejno deklarowane zmienne C++ rozmieszcza w pamięci w jakiś konkretny sposób!

3.7. Co się odbywa w tym programie?

```
int a=22, b=333; int * c = &a;
*((&b)+1)=111; *c=9; c--; *c=88; cout << ((&b)==c);
```

**3.8. A w tym? `int * a; int * b; **((&b)+1)=111;`
Jak nie używając zmiennej *b*, zmienić po takich deklaracjach jej wartość? (Załóżmy tu, że C++ umieszcza kolejno deklarowane zmienne bezpośrednio przed poprzednimi).

* Ta gwiazdka oznacza przypis. Gwiazdka, do której się on odnosi – zadanie o podwyższonym stopniu trudności. Większość gwiazdek na tej stronie to zaś… Byle do Gwiazdki!

3.9. Czemu po deklaracjach „int *a*; int *b*[10];" podstawienie „*b*=*a*" nie jest możliwe? A gdyby podstawić *b* do *a* (tylko jak to zrobić?!), jakimi wartościami są (co znaczą, co mogą dawać): **b*, **a*, &*a*, &*b*, *a*[2], &*b*[2], (**a*)+2, (**a*+2), (**b*)+2, (**b*+2), ...?

3.A. Co wyświetli program? (Niektórych wartości może nie dać się ustalić, a o innych można tylko powiedzieć, że na pewno wynoszą tyle, co ...) UWAGA jak w zad. 3.5.

```
int a, *p1, *p2;
p2 = p1;
```

```
cout << p1 << " " << *p1 << " " << &p1 << " ";
cout << p2 << " " << *p2 << " " << &p2 << " " << &a << endl;
a=5; p1=&a; *p2=a;
cout << p1 << " " << *p1 << " " << &p1 << " ";
cout << p2 << " " << *p2 << " " << &p2 << " " << &a << endl;
//W tej części programu umiejętny programista wpisał pod adresem 123
//zmienną typu int o wartości 100.
p1=(int *)123; *p2=*p1; (*p1)++;
cout << p1 << " " << *p1 << " " << &p1 << " ";
cout << p2 << " " << *p2 << " " << &p2 << " " << &a << endl;
p1=p2;
cout << p1 << " " << *p1 << " " << &p1 << " ";
cout << p2 << " " << *p2 << " " << &p2 << " " << &a << endl;
```

3.B. Niech *L* oznacza przybliżenie *I* lewymi, a *P* – prawymi prostokątami.
Uporządkuj rosnąco wartości *I*, *L* i *P*, jeśli: x) $I = \int_{-1}^{1} \sqrt{1-x^2}dx$, y) $I = \int_{0}^{\pi} \sin x \, dx$.

3.C(p). Dla jakich funkcji (o wartościach dodatnich, jak zwykle) suma pól lewych prostokątów jest mniejsza od pola pod wykresem?