Ε. Μ. Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχ. & Μηχ. Υπολογιστών.
Ε. Ζάχος, Ν. Παπασπύρου,
Δ. Φωτάκης, Π. Ποτίκας,
Δ. Σούλιου, Κ. Τζαμαλούκας

ΕΠΩΝΥΜΟ:	more
ΟΝΟΜΑ:	than
ΑΡ. ΜΗΤΡΩΟΥ:	
ΕΞΑΜΗΝΟ:	good
ΟΜΑΔΑ ΕΡΓ:	
ΑΜΦΙΘΕΑΤΡΟ:	enough
ΘΕΣΗ:	

1	
2	
3	
4	
5	
6	
ΣΥΝΟΛΟ	

A

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ Η/Υ

Κανονική εξέταση, Φεβρουάριος 2020

Κανονισμός εξέτασης: *1) Υποχρεούστε να δείξετε στον επιτηρητή όταν σας ζητηθεί τη φοιτητική σας ταυτότητα ή άλλο αποδεικτικό της ταυτότητάς σας με φωτογραφία. 2) Η εξέταση γίνεται με κλειστά βιβλία και σημειώσεις. 3) Δεν μπορείτε να χρησιμοποιείτε ηλεκτρονικές συσκευές. Αν έχετε μαζί σας κινητό τηλέφωνο, απενεργοποιήστε το και κρύψτε το.*

1. (8)

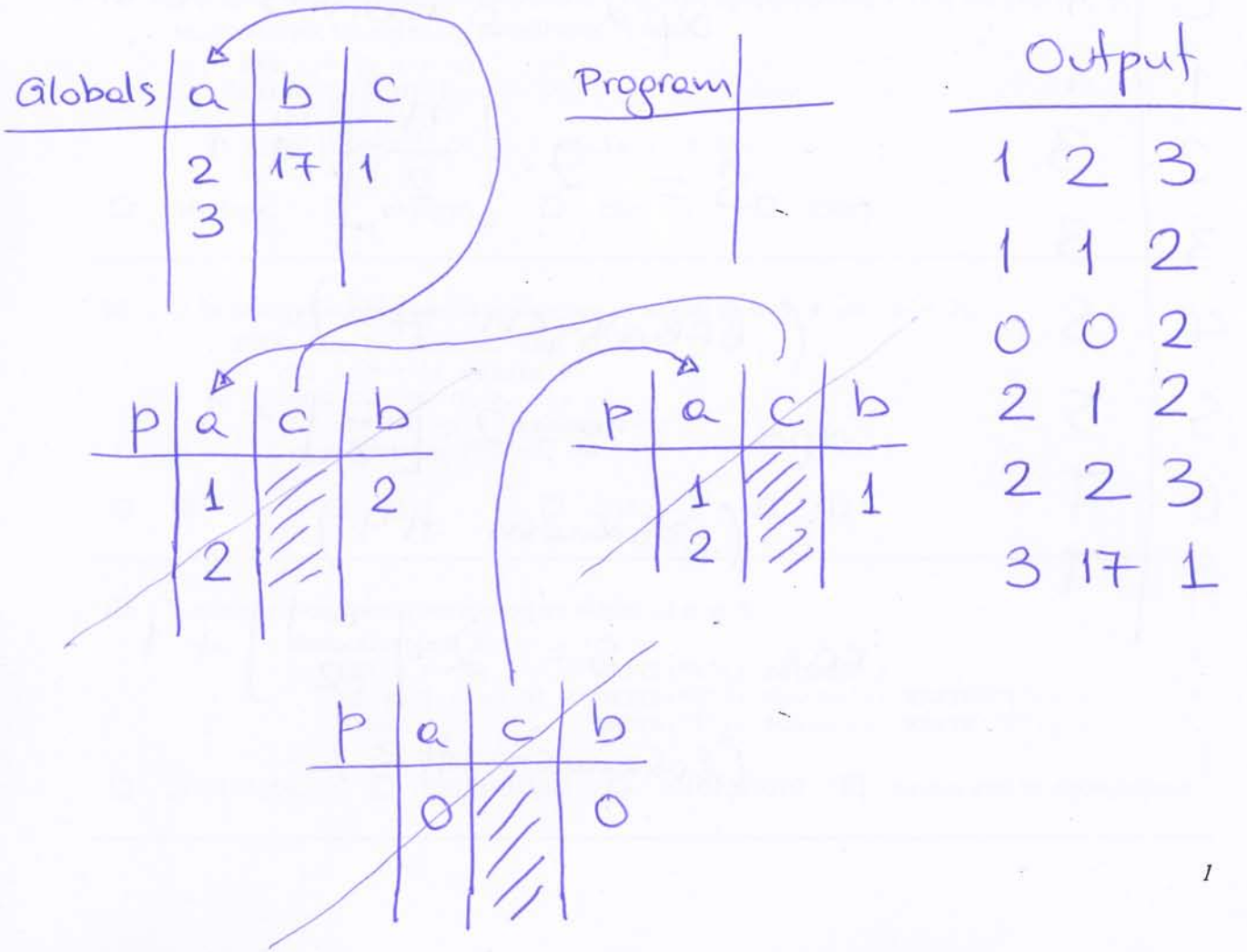
Να δείξετε σε **πίνακα** όλες τις **ενδιάμεσες τιμές** καθώς και τις **τιμές που τυπώνονται** από το παρακάτω πρόγραμμα C++ (εκτέλεση με το χέρι).

```
#include "pzhelp"

int a = 2, b = 17, c = 1;

PROC p(int a, int &c) {
  int b = a * c++;
  WRITELN(a, b, c);
  if (4*a > c) { p(b/2, a); WRITELN(a, b, c); }
}

PROGRAM { p(c, a); WRITELN(a, b, c); }
```



2. (8)

Έστω **n** μη αρνητικός ακέραιος αριθμός. Βρείτε τι κάνει το παρακάτω μέρος προγράμματος C++ και αποδείξτε την ορθότητά του, χρησιμοποιώντας βεβαιώσεις (και ροές), δηλαδή αξιωματική σημασιολογία.

```
        // 1: Βεβαίωση εισόδου:
s = 1;
        // 2:
for (int i = 1; i <= n; ++i)
        // 3: Αναλλοίωτη βρόχου:
    s = 2*i - s;
        // 4:
    ;   // 5: Βεβαίωση εξόδου:
```

Απόδειξη για όλες τις δυνατές ροές:

Βαριέμαι, αλλά το πρόγραμμα τρέχει κάπως έτσι:

i	s
0	1
1	1
2	3
3	3
4	5
5	5
6	7
7	7
⋮	

άρα ο τύπος είναι

$$s = 2 \cdot \left\lfloor \frac{n}{2} \right\rfloor + 1$$

(βεβαίωση #5)

άρα $s = 2 \cdot \left\lfloor \frac{i}{2} \right\rfloor + 1$

(βεβαίωση #4)

και $s = 2 \cdot \left\lfloor \frac{i-1}{2} \right\rfloor + 1$

(βεβαίωση #3)

Βαριέμαι !

3. (10)

Απαντήστε στις παρακάτω ερωτήσεις πολλαπλής επιλογής μαυρίζοντας σε κάθε μία το πολύ ένα από τα τέσσερα κουτάκια. Κάθε σωστή απάντηση παίρνει 1,5 μονάδα. Κάθε λάθος απάντηση χάνει 0,5 μονάδα (αρνητική βαθμολογία). Κενές ή άκυρες απαντήσεις δεν προσθέτουν ούτε αφαιρούν μονάδες.

(α) Έστω ότι έχετε τρεις διαφορετικούς αλγορίθμους Α, Β και Γ, που επιλύουν το ίδιο πρόβλημα. Η πολυπλοκότητα του Α είναι $O(n^3 (\log n)^6)$, του Β είναι $O(n^4)$, και του Γ είναι $O(2^n)$. Ποιον από τους τρεις θα προτιμούσατε; (Θεωρήστε ότι μας ενδιαφέρουν μεγάλες τιμές του n.)

☐ τον Γ ☐ τον Β ☐ τον Α ☐ οποιονδήποτε από τους Α ή Β, δεν έχουν διαφορά

(β) Ο αλγόριθμος γραμμικής αναζήτησης, για την εύρεση στοιχείου σε πίνακα με n στοιχεία:

☐ απαιτεί χρόνο $O(1)$ στη χειρότερη περίπτωση.
☐ απαιτεί χρόνο $O(\log n)$ στη χειρότερη περίπτωση.
☐ απαιτεί χρόνο $O(n)$ στη χειρότερη περίπτωση.
☐ απαιτεί χρόνο $\Theta(\log n)$ στην καλύτερη περίπτωση.

(γ) Ποιο από τα παρακάτω προγράμματα τυπώνει **17**;

```
int k = 4;
PROC proc1(int &n) {
  n = (n-1)*(n+1);
  WRITELN(n+k/2);
}
PROGRAM { proc1(k); }
```

```
int k = 4;
PROC proc2(int n) {
  n = (n-1)*(n+1);
  WRITELN(n+k/2);
}
PROGRAM { proc2(k); }
```

☐ το αριστερό ☐ το δεξιό ☐ και τα δύο ☐ κανένα από τα δύο

(δ) Στο τέλος της εκτέλεσης του ακόλουθου τμήματος προγράμματος, η τιμή της μεταβλητής t (ως συνάρτηση της τιμής της μεταβλητής n) είναι:

```
int t = 0, i = n;
while (i > 0) {
  int j = i--;
  do { t++; j /= 2; } while (j > 0);
}
```

☐ $\Theta(n \log n)$ ☐ $\Theta(\log n)$ ☐ $\Theta(n)$ ☐ $\Theta(n^2)$

(ε) Τι θα επιστρέψει η παρακάτω συνάρτηση αν κληθεί με **n = 9** και **x = 2**;

```
FUNC int fun(int n, int x) {
    if (n <= 1) return x;
    int t = fun(n/2, x);
    if (n % 2 == 0) return t*t;
    else return t*t*x;
}
```

☐ 18 ☐ 81 ☐ 256 ☐ 512

(ζ) Τι τυπώνει η παρακάτω συνάρτηση αν κληθεί με **n = 5**;

```
PROC fun(int n) {
    if (n == 0)     { WRITE("2"); return; }
    if (n % 2 == 0) { WRITE("0"); fun(n-1); WRITE("1"); }
    else            { WRITE("1"); fun(n-1); WRITE("0"); }
}
```

☐ 10101010102 ☐ 01010101012 ☐ 10101201010 ☐ κανένα από τα προηγούμενα

(η) Τι τυπώνει το παρακάτω πρόγραμμα;

```
PROGRAM {
    int *p = new int, *q = new int, *t = new int;
    *p=17; *q=2**p;
    *t=--*q/2;
    p=t; *p=*p+*q;
    *t=*p**q/2;
    WRITELN(*q+*p);
}
```

☐ 841 ☐ 748 ☐ 76 ☐ κανένα από τα προηγούμενα

ΠΡΟΧΕΙΡΟ

4. (10)

Αν γράψετε μόνο τη λέξη «ΚΕΝΟ» αντί λύσης σε αυτό το θέμα, θα πάρετε 2 μονάδες.

Δίνεται ένας πίνακας **A** αποτελούμενος από **N** ακέραιους αριθμούς ($1 \le \mathbf{N} \le 1.000.000$), και ένας θετικός φυσικός αριθμός **K**. Μας ενδιαφέρουν τα τμήματα του πίνακα **A** (δηλαδή διαδοχικοί όροι A[i], A[i+1] ... A[i+m]) στα οποία εμφανίζονται **τουλάχιστον** **K** περιττοί αριθμοί. Ποιο είναι το μήκος του μικρότερου τέτοιου τμήματος;

Να γράψετε μία *κομψή και αποδοτική* συνάρτηση που δέχεται ως παραμέτρους τα **N**, **A** και **K**, και υπολογίζει το **ελάχιστο** μήκος ενός τμήματος του πίνακα **A** που να περιέχει τουλάχιστον **K** περιττούς αριθμούς. Αν δεν υπάρχει τέτοιο τμήμα, η συνάρτηση πρέπει να επιστρέφει 0.

Παράδειγμα 1: (**N** = 10, **K** = 3)

A = [0, 3, 0, 2, 1, 2, 2, 7, 1, 0]
oddk(**N**, **K**, **A**) = 5

Το τμήμα που είναι παραπάνω υπογραμμισμένο για διευκόλυνσή σας, έχει μήκος 5 και περιέχει 3 περιττούς αριθμούς (1, 7 και 1).

Παράδειγμα 2: (**N** = 10, **K** = 5)

A = [0, 1, 1, 4, 0, 2, 0, 1, 0, 2]
oddk(**N**, **K**, **A**) = 0

Στον παραπάνω πίνακα δεν υπάρχει τμήμα που να περιέχει τουλάχιστον 5 περιττούς αριθμούς.

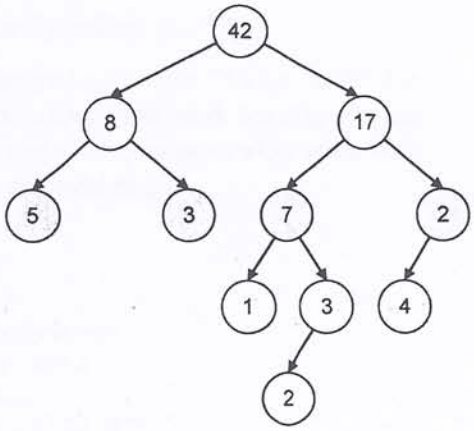
Ερώτηση bonus (2 επιπλέον μονάδες): Ποια είναι η πολυπλοκότητα της λύσης σας; Εξηγήστε.

```
// Γράφω την O(N²), "προφανή" / BONUS

int oddk(int N, int a[], int K) {
    int best = N+1;   // κάτι μεγάλο
    for (int i=0; i<N; ++i) {
        int count = 0;
        for (int j=i; j<N; ++j) {
            if (a[j]%2 == 1) ++count;
            if (count >= K) best = min(best,
                                       j-i+1);
        }
    }
    return best <= N ? best : 0;
}
```

5. (10)

Αν γράψετε μόνο τη λέξη «ΚΕΝΟ» αντί λύσης σε αυτό το θέμα, θα πάρετε 2 μονάδες.



Ορίστε τον τύπο **node** των κόμβων του δυαδικού δέντρου που περιέχουν ως πληροφορία ακέραιους αριθμούς.

Θεωρούμε ότι η ρίζα του δέντρου βρίσκεται στο επίπεδο 1, τα παιδιά της στο επίπεδο 2, τα παιδιά αυτών στο επίπεδο 3, κ.ο.κ.

Γράψτε μια *κομψή και αποδοτική* συνάρτηση που δέχεται ως παράμετρο ένα δέντρο **t** και έναν ακέραιο αριθμό **k** και επιστρέφει το άθροισμα των τιμών των κόμβων που βρίσκονται στο επίπεδο **k**. Η συνάρτηση θα πρέπει να έχει ως επικεφαλίδα:

```
. FUNC int sumlevel(node *t, int k);
```

Για το παράδειγμα του διπλανού σχήματος, αν **k** = 3, η συνάρτησή σας θα πρέπει να επιστρέφει το άθροισμα των κόμβων του τρίτου επιπέδου: 5 + 3 + 7 + 2 = 17.

```
struct node {
    int data;
    node *left, *right;
};

int sumlevel (node *t; int k ) {
    if (t == nullptr || k<=0 )
        return 0;
    if (k==1 ) return t→data;
    return sumlevel (t→left, k-1)
        + sumlevel (t→right, k-1);
}
```

6. (10)

Αν γράψετε μόνο τη λέξη «ΚΕΝΟ» αντί λύσης σε αυτό το θέμα, θα πάρετε 2 μονάδες.

Ζητείται ένα *κομψό και αποδοτικό* πρόγραμμα που διαβάζει από το αρχείο με όνομα "file.txt" ένα (μη κενό) κείμενο αποτελούμενο από πεζά γράμματα του λατινικού αλφαβήτου, κενά διαστήματα και αλλαγές γραμμής. Το πρόγραμμά σας πρέπει να εκτυπώνει στην οθόνη το ίδιο κείμενο αλλά κάθε λέξη με **λιγότερα από τέσσερα γράμματα** πρέπει να τυπώνεται **μέσα σε παρενθέσεις**.

Παράδειγμα:

(κείμενο):

```
the first electronic computers were monstrous contraptions
filling several rooms consuming as much electricity as a
good size factory and costing millions of dollars
but with the computing power of a modern hand held calculator
```

(οθόνη):

```
(the) first electronic computers were monstrous contraptions
filling several rooms consuming (as) much electricity (as) (a)
good size factory (and) costing millions (of) dollars
(but) with (the) computing power (of) (a) modern hand held calculator
```

```
#include "pzhelp"
#define MAXWORD 20
// Θεωρώ μεγιστο μηκος λέξης — δε χρειάζεται
// αλλά δε βαριέσαι

PROGRAM {
    INPUT("file.txt");
    int c;
    while ((c=getchar()) != EOF) {
        if (isletter(c)) {
            int n=0;
            char word[MAXWORD];
            do { word[n++]=c; c=getchar();
                } while(isletter(c));
            if (n<4) putchar('(');
            for (int i=0; i<n; ++i) putchar(word[i]);
            if (n<4) putchar(')');
        }
        putchar(c);
    }
```

```
// Εχαβα την isletter:
bool isletter(char c) {
  return c >= 'a' &&
         c <= 'z';
}
```