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

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

1	
2	
3	
4	
5	
6	
ΣΥΝΟΛΟ	

A

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ Η/Υ

Επαναληπτική εξέταση, Σεπτέμβριος 2019

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)

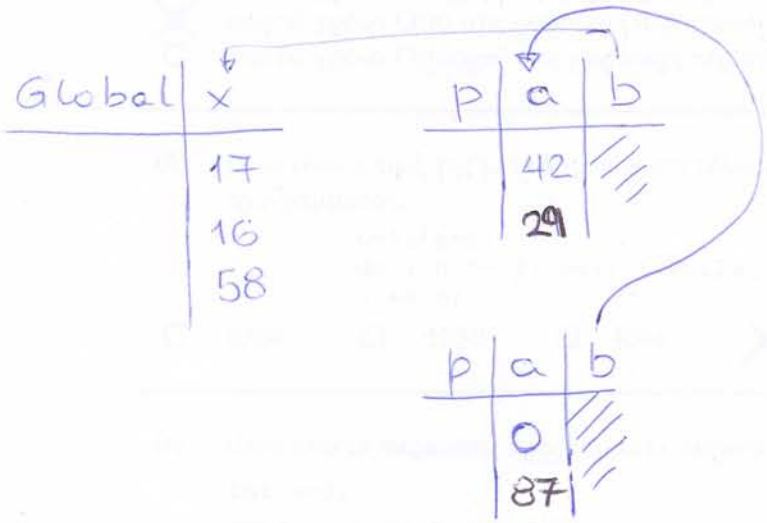
Να δείξετε σε **πίνακα** όλες τις **ενδιάμεσες τιμές** καθώς και τις **τιμές που τυπώνονται** από το παρακάτω πρόγραμμα C++ (εκτέλεση με το χέρι).

```
#include "pzhelp"

int x;

void p(int a, int &b) {
  WRITELN(x, a, b);
  if (a < b) a = x+b;
  else       { --x; b += a; a = x/2; WRITELN(x, a, b); p(b-x, a); }
  WRITELN(x, a, b);
}

int main() { x = 17; p(42, x); WRITELN(x); }
```



Output

17 42 17
58 29 58
58 0 29
58 87 29
58 29 58
58

Αγνοήστε αυτό το θέμα!
Τα συντακτικά διαγράμματα δεν είναι στην ύλη που εξετάζουμε στο μάθημα Προγραμματισμός Ηλεκτρονικών Υπολογιστών για το ακαδημαϊκό έτος 2024-25.

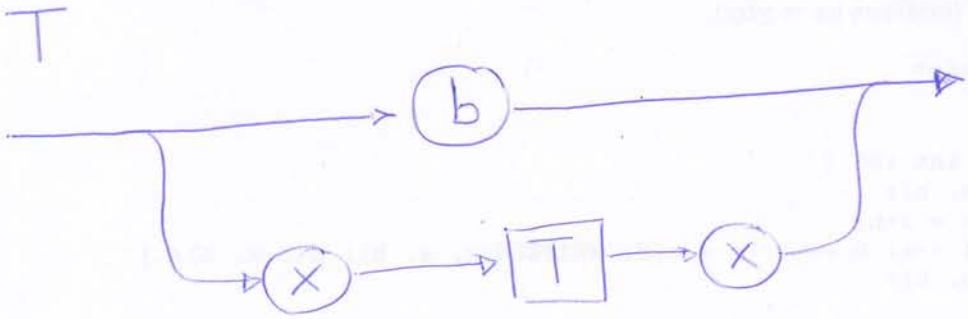
2. (8)

Κατασκευάστε **συντακτικό διάγραμμα** που περιγράφει όλων τις συμβολοσειρές της μορφής:

abc **axbxc** **axxbxxc** **axxxbxxxc** **axxxxbxxxxc** ...

δηλαδή τις συμβολοσειρές που:

- έχουν περιττό πλήθος χαρακτήρων,
- ο πρώτος, ο μεσαίος και ο τελευταίος χαρακτήρας είναι αντίστοιχα **a**, **b**, **c**, και
- οι υπόλοιποι χαρακτήρες είναι **x**.



(η εξήγηση δε χρειάζεται)

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 $O(n^{4/3})$, του Β είναι $O(n \log^2 n)$, και του Γ είναι $O((4/3)^n)$. Ποιον από τους τρεις θα προτιμούσατε; (Θεωρήστε ότι μας ενδιαφέρουν μεγάλες τιμές του n.)

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

$B < A < \Gamma$

$O(n \log^2 n) < O(n^{4/3}) < O(\left(\frac{4}{3}\right)^n)$

(β) Τοποθετήστε τους παρακάτω αλγορίθμους σε αύξουσα σειρά (πρώτος αυτός με τη μικρότερη πολυπλοκότητα), ως προς την χρονική τους πολυπλοκότητα στη χειρότερη περίπτωση.

Α. ταξινόμηση με διαμέριση (quicksort) σε πίνακα n στοιχείων $O(n^2)$

Β. δυαδική αναζήτηση σε ταξινομημένο πίνακα n στοιχείων $O(\log n)$

Γ. εύρεση του 30ου μικρότερου στοιχείου σε μη ταξινομημένο πίνακα n στοιχείων. $O(n)$

- ☐ Γ, Α, Β
- ☒ Β, Γ, Α
- ☐ Β, Α, Γ
- ☐ Γ, Β, Α

$B < \Gamma < A$

$O(\log n) < O(n) < O(n^2)$

(γ) Ο καλύτερος αλγόριθμος αναζήτησης στοιχείου σε διπλά συνδεδεμένη λίστα με n στοιχεία ταξινομημένα σε αύξουσα σειρά:

- ☐ απαιτεί χρόνο O(1) στη χειρότερη περίπτωση.
- ☐ απαιτεί χρόνο O(log n) στη χειρότερη περίπτωση.
- ☒ απαιτεί χρόνο O(n) στη χειρότερη περίπτωση.
- ☐ απαιτεί χρόνο O(nlog n) στη χειρότερη περίπτωση.

Είναι $O(n)$ γιατί αναγκαστικά ψάχνω γραμμικά (δε γίνεται binary search σε λίστα)

(δ) Ποια είναι η τιμή της μεταβλητής **n** στο τέλος της εκτέλεσης του ακόλουθου τμήματος προγράμματος;

```
n=1; p=1;
do { n *= 4; p++; } while (n <= 1024);
n += p;
```

Invariant: $n = 4^{p-1}$

1, 4, 16, 64, 256, 1024, 4096

και τότε p=7

άρα 4096 + 7 = 4103

- ☐ 1024
- ☐ 1030
- ☐ 4096
- ☒ 4103

(ε) Ποιο από τα παρακάτω προγράμματα τυπώνει **144**;

```
int k=3;
PROC proc1(int &n) {
  n=k*(k+1); WRITELN(n*k);
}
PROGRAM test1() { proc1(k); }
```

```
int k=3;
PROC proc2(int n) {
  n=k*(k+1); WRITELN(n*k);
}
PROGRAM test2() { proc2(k); }
```

- ☒ test1
- ☐ test2
- ☐ και τα δύο
- ☐ κανένα από τα δύο

Το test1() κάνει n = 3·4 = 12 αλλά το n είναι το k άρα τυπώνει 12·12 = 144

Το test2() κάνει πάλι n = 3·4 = 12 αλλά το k είναι ακόμα 3, άρα τυπώνει 12·3 = 36.

Επιστρέφει το αθροισμα των 1ης στηλης εκτα:

x	n
800	100
400	99
200	98
100	97
50	96
25	95
12	94
6	93
3	92
1	91
0	90
0	89
⋮	⋮
0	0

(ζ) Τι επιστρέφει η παρακάτω συνάρτηση αν κληθεί με **x = 800** και **n = 100**;

```
FUNC int fun(int x, int n) {
   if (n == 0) return x;
   else return x + fun(x/2, n-1);
}
```

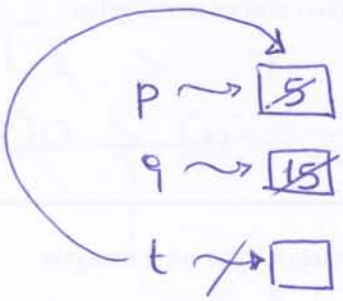
☒ 1597 ☐ 1600 ☐ 80000 ☐ κανένα από τα προηγούμενα

(η) Τι τυπώνει το παρακάτω πρόγραμμα;

```
PROGRAM {
  int *p = new int, *q = new int, *t = new int;
  *p=5; *q=15; t=p;
  *t*=*p; *q+=*t+*p/2;
  WRITELN(*q);
}
```

☐ 40 ☐ 42 ☒ 52 ☐ κανένα από τα προηγούμενα

ΠΡΟΧΕΙΡΟ



4. (10)

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

Κατά μήκος ενός δρόμου, υπάρχουν **N** σπίτια ($1 \le \mathbf{N} \le 1.000.000$). Ξεκινώντας από την αρχή του δρόμου, για το i-οστό σπίτι γνωρίζουμε τη θέση του **x**[i], δηλαδή την απόστασή του από την αρχή του δρόμου. (Προφανώς ισχύει **x**[i] < **x**[j] για κάθε **i** < **j**.) Λέμε ότι το i-οστό σπίτι και το j-οστό σπίτι ($i \ne j$) είναι γειτονικά αν η απόσταση μεταξύ τους δεν υπερβαίνει κάποια δοθείσα τιμή **D**. Τέλος, ορίζουμε ως C(i) το πλήθος των σπιτιών που είναι γειτονικά με το i.

Να γράψετε μία *κομψή και αποδοτική* συνάρτηση που να δέχεται ως παραμέτρους τα **N**, **x** και **D** και να υπολογίζει τη μέγιστη τιμή του C(i).

Παράδειγμα: **N** = 9, D = 30, **x** = { 10, 17, 20, 40, 42, 60, 70, 75, 99 }
max_neighbors(**N**, **x**, **D**) = 6

Το σπίτι με τους περισσότερους γείτονες είναι αυτό στη θέση 40. Οι γείτονές του είναι τα σπίτια στις θέσεις 10, 17, 20, 42, 60 και 70.

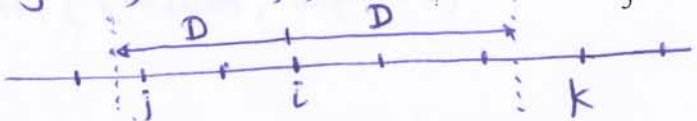
Ερώτηση bonus (2 επιπλέον μονάδες): Ποια είναι η πολυπλοκότητα της λύσης σας; Εξηγήστε.

#1) Με πολυπλοκότητα $O(N^2)$ – η προφανής λύση.

```
int max_neighbors ( int N , int x[] , int D) {
    int best = 0 ;
    for (int i = 0 ; i<N ; ++i ) {
        int count = 0 ;
        for (int j= 0 ; j< N ; ++j )
            if ( i != j && abs(x[i]-x[j]) <= D)
                ++count;
        best = max ( count, best) ;
    }
    return best;
}
```

#2) Με πολυπλοκότητα $O(N)$ – η καλή λύση.

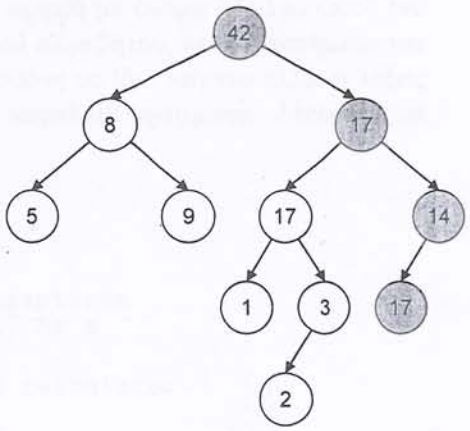
```
int max_neighbors (int N , int x[] , int D ) {
    int best = 0 ;
    for (int i=0 , j=0 , k=0 ; i< N ; ++i) {
        while (k<N && x[k]-x[i] <= D ) ++k;
        while ( j<i && x[i]-x[j] > D) ++j;
        best = max ( k-j-1 , best) ;
    }
    return best
}
```



5. (10)

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

Ορίστε τον τύπο **tree** του δυαδικού δέντρου που περιέχει στους κόμβους του ακέραιους αριθμούς, καθώς και τον τύπο **node** του κόμβου του.



Γράψτε μια *κομψή και αποδοτική* συνάρτηση που να δέχεται ως παράμετρο ένα τέτοιο δένδρο και να επιστρέφει το πλήθος των κόμβων του μακρύτερου μονοπατιού από τη ρίζα προς κάποιο φύλλο, το οποίο να μην περνά από δύο διαδοχικούς ίδιους αριθμούς. Αν δεν υπάρχει τέτοιο μονοπάτι, η συνάρτησή σας θα πρέπει να επιστρέφει μηδέν.

Η συνάρτηση θα πρέπει να έχει ως επικεφαλίδα:

```
FUNC int longestPath(tree t);
```

Για το παράδειγμα του διπλανού σχήματος, η συνάρτησή σας θα πρέπει να επιστρέφει 4 (το μακρύτερο μονοπάτι είναι σημειωμένο με γκρίζους κόμβους).

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

ΑΝ ΔΕ ΣΚΕΦΤΗΚΑΜΕ να βάλουμε επιπλέον παραμέτρους

#1)

```
int longestPath ( tree t ) {
    if (t == nullptr ) return 0;
    int n_left = (t->left != nullptr && t->data != t->left->data)
        ? longestPath (t->left) : 0;
    if (t->left == nullptr || n_left > 0) ++n_left;
    int n_right = (t->right != nullptr && t->data != t->right->data)
        ? longestPath (t->right) : 0;
    if (t->right == nullptr || n_right > 0 ) ++n_right;
    return max(n_left, n_right);
}
```

ΕΝΑΛΛΑΚΤΙΚΑ (και πολύ κομψότερα)

#2)

```
int longestPath ( tree t, int depth = 0, int parent = 0) {
    if (t == nullptr) return depth;
    if (depth > 0 && parent == t->data) return 0;
    return max (longestPath( t->left, depth+1, t->data),
                longestPath ( t->right, depth+1, t->data));
}
```

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 οι λέξεις που έχουν άρτιο πλήθος γραμμάτων θα πρέπει να τυπώνονται **με κεφαλαία γράμματα**. Μπορείτε να θεωρήσετε ότι οι λέξεις έχουν το πολύ 30 γράμματα.

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"

PROGRAM {
   INPUT("file.txt");
   int c;
   while ((c=getchar()) != EOF) {
      if (isletter(c)) {
         int n=0;
         char s[MAX];
         do { s[n++] = c; c= getchar(); } while(isletter(c));
         for (int i=0; i<n; ++i)
           putchar( n%2==0 ? uppercase(s[i]) : s[i] );
      }
      putchar(c);
   }
}
```

Πρέπει να προηγηθούν:

```
#define MAX 30
```

και αν χρειαζόμαστε τις isletter/uppercase και όχι τις αντίστοιχες της βιβλιοθήκης (isalpha/toupper):

```
FUNC bool isletter(char c) {
  return c>='a' && c<='z';
}
FUNC char uppercase(char c) {
  return c-'a'+'A';
}
```