Θεματική ενότητα 8: Δημιουργία προγραμμάτων

Παράδειγμα 1: **Να γραφεί πρόγραμμα που να δέχεται ως είσοδο κείμενο, να απαριθμεί τις εμφανίσεις των ψηφίων 0-9, τα λευκά διαστήματα και τους υπόλοιπους χαρακτήρες και στη συνέχεια να τυπώνει τα αποτελέσματα. Το πρόγραμμα ολοκληρώνεται όταν δοθεί ο χαρακτήρας τελεία '.'.**

Λύση:

Το παραπάνω πρόβλημα μπορεί να μορφοποιηθεί σε *δομημένα Ελληνικά* ως εξής:

Για κάθε χαρακτήρα του κειμένου που είναι διάφορος τής '.'
έλεγξε τον τύπο του χαρακτήρα
αν είναι ένας από τους ' ', '\t', '\n' 3
αύξησε τον απαριθμητή των διαστημάτων
αν είναι ψηφίο
αύξησε τον απαριθμητή που αντιστοιχεί στο ψηφίο
σε κάθε άλλη περίπτωση
αύξησε τον απαριθμητή των υπόλοιπων χαρακτήρων

Αναπαράσταση δεδομένων:

- **Όσον αφορά τις μεταβλητές απαιτείται:**

1) Ένας απαριθμητής για τα διαστήματα, τον οποίο ονομάζουμε n_white.

2) Ένας απαριθμητής για τους λοιπούς χαρακτήρες, ο n_other, και δέκα απαριθμητές για τα ψηφία.

Για την τελευταία περίπτωση μπορούμε να δηλώσουμε δέκα ανεξάρτητες μεταβλητές αλλά είναι προτιμότερο να επιλεχθεί ένας πίνακας δέκα θέσεων, όπως

```
int n_digit[10];
```

ο οποίος οδηγεί σε πιο συμπαγή και δομημένο κώδικα.

Αναπαράσταση δεδομένων:

- **Για την εκτύπωση των αριθμών των εμφανίσεων των δέκα ψηφίων, στην περίπτωση του πίνακα απαριθμητών, ο αντίστοιχος κώδικας θα έχει τη μορφή**

```
for (i=0;i<10;i++)
    printf("Το ψηφίο %d εμφανίσθηκε \t %d \t φορές\n",i,n_digit[i]);
```

Αναλογιστείτε τον κώδικα για την περίπτωση 10 ανεξάρτητων μεταβλητών!!!

Αναπαράσταση διεργασιών:

- **Για τη διεργασία *πάρε χαρακτήρα* χρησιμοποιείται η συνάρτηση *getchar()*, η οποία επιστρέφει τον χαρακτήρα που διαβάζει από την κύρια είσοδο. Αυτός ο χαρακτήρας πρέπει να αποθηκευθεί σε μία μεταβλητή τύπου χαρακτήρα για περαιτέρω επεξεργασία.**

Τα παραπάνω οδηγούν στη δήλωση

```
char ch;
```

και στην έκφραση

```
ch=getchar();
```

η τιμή της οποίας είναι η τιμή του αριστερού τελεστέου της έκφρασης.

Αναπαράσταση διεργασιών:

- **Για την ανίχνευση του τέλους του αρχείου χρησιμοποιούμε το συσχετιστικό τελεστή != οδηγούμαστε στην έκφραση**

(ch=getchar())!='.'

Η έκφραση γίνεται ψευδής όταν αναγνωσθεί '.'. Μπορεί επομένως να χρησιμοποιηθεί ως έκφραση μίας πρότασης *while*, που θα οδηγεί στην επανάληψη του συνόλου των ενεργειών που το πρόγραμμα πρέπει να εκτελεί για κάθε χαρακτήρα.

Διαμόρφωση της ροής ελέγχου:

Με βάση τα προηγούμενα, η περιγραφή διαμορφώνεται ως εξής:

```
while ((ch=getchar())!='.')
{
    έλεγξε τον τύπο του χαρακτήρα
        αν είναι ένας από τους ' ', '\t', '\n'
            αύξησε τον απαριθμητή των διαστημάτων
        αν είναι ψηφίο
            αύξησε τον απαριθμητή που αντιστοιχεί στο ψηφίο
        σε κάθε άλλη περίπτωση
            αύξησε τον απαριθμητή των υπόλοιπων χαρακτήρων
}
```

Το σώμα της *while* αποτελεί κλασική περίπτωση επιλογής από αμοιβαία αποκλειόμενες ενέργειες, γεγονός που οδηγεί στη χρήση της πρότασης *switch*. Η έκφραση ανάλογα με την τιμή της οποίας θα γίνει η επιλογή της κατάλληλης ενέργειας είναι η απλή έκφραση ch.

Α) ***εάν είναι ένας από τους ' ', '\t', '\n'***

αύξησε τον απαριθμητή n_white

ή εκφρασμένη στη **C**

```
case ' ':
case '\t':
case '\n':
   n_white++;
   break;
```

Κοινή έξοδος για τις τρεις *case*

Β) ***εάν ο χαρακτήρας είναι ψηφίο***

αύξησε τον απαριθμητή που αντιστοιχεί στο ψηφίο

μία πρώτη μορφή του κώδικα είναι η παρακάτω:

```
case '0':
   n_digit[0]++;  break;
. . . . . .
case '9':
   n_digit[9]++;   break;
```

- Ο κώδικας αυτός δεν εκμεταλλεύεται τη δήλωση των απαριθμητών των ψηφίων ως πίνακα χαρακτήρων. Για το λόγο αυτό, θα προσπαθήσουμε να ενοποιήσουμε τα *case* ώστε να έχουν μία παραμετρική πρόταση, που σε κάθε περίπτωση θα οδηγεί στην αύξηση του κατάλληλου απαριθμητή.
- Θεωρούμε την έκφραση

ch-‘0’

και εξετάζουμε την τιμή της για τιμές τής ch από το σύνολο *{‘0’, ‘1’, …, ‘9’}*. Είναι προφανές ότι, εάν το ch είναι *‘0’*, η έκφραση έχει τιμή 0 οπότε και η πρόταση

n_digit[ch-‘0’]++;

έχει ως αποτέλεσμα την αύξηση του απαριθμητή που αντιστοιχεί στο ψηφίο *‘0’*.

- Αντίστοιχα, η παραπάνω πρόταση για ch=‘8’ θα αυξήσει τον απαριθμητή που αντιστοιχεί στο *‘0’*. Κατά αυτόν τον τρόπο οδηγούμαστε στον ακόλουθο συμπαγή κώδικα:

```
case ‘0’:
case ‘1’:
…
…
case ‘9’:
    n_digit[ch-’0’]++;
    break;
```

Κοινή έξοδος για τις δέκα *case*

Γ) **Για τα υπόλοιπα στοιχεία έχουμε την κλασική περίπτωση χρήσης της εντολής default, οπότε προκύπτει:**

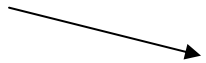
```
default:
    n_other++;
    break;
```

Το τελευταίο σημείο που πρέπει να προσεχθεί είναι η αρχικοποίηση των μεταβλητών.

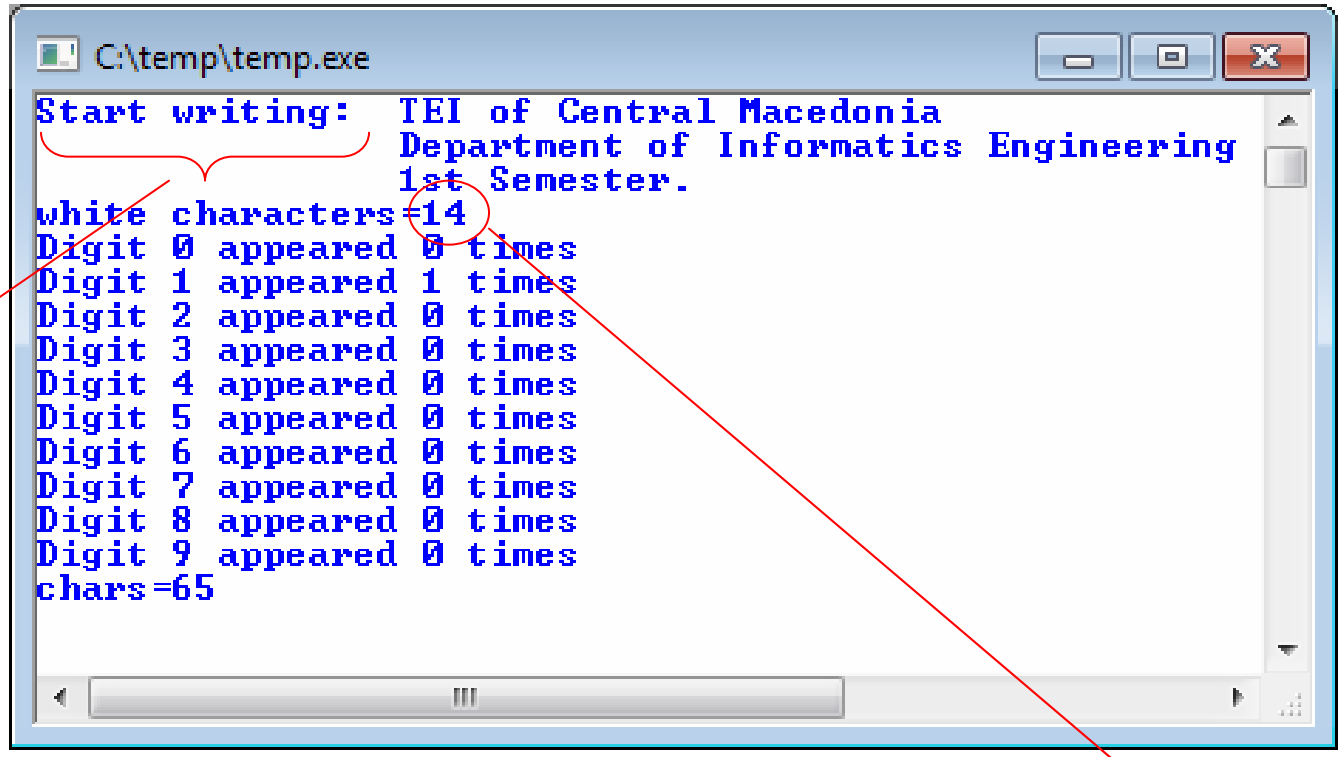
Ενδεικτικός κώδικας:

```
#include<stdio.h>
main() {
  char  ch;
  int i,n_white=0,n_digit[10],n_other=0;
  for(i=0;i<10;i++)   n_digit[i]=0;
  printf("Start writing:");
  while ((ch=getchar())!= '.' )   {
    switch(ch)   {
      case ' ':
      case '\t':
      case '\n':
        n_white++;
        break;
```

```
    case '0':    case '1':
    case '2':    case '3':
    case '4':    case '5':
    case '6':    case '7':
    case '8':    case '9':
      n_digit[ch-'0']++;
      break;
    default:
      n_other++;
      break;   }  //τέλος της switch
  } //τέλος της while
  printf("white characters=%d\n",n_white);
  for(i=0;i<10;i++) printf("Digit %d appeared %d times\n",i,n_digit[i]);
  printf("chars=%d \n",n_other);
}
```



Αποτελέσματα:



2 \t

5 \t, 2 \n, 7 κενά

Παράδειγμα 2: Να βρεθούν τα σφάλματα του ακόλουθου προγράμματος καθώς και τα σημεία στα οποία θα μπορούσαν να γίνουν βελτιώσεις.

```
#include <stdio.h>
main ()  {
 int a, b;
 do
 {
  printf (“Correspond the integers 1-5 to letters A-E\n");
  printf (“Give an integer within [1  5] ");  scanf ("%i",&b);
  switch (b)
  {
   case (1):
    printf ("A\n");
   break;
   case (2):
    printf ("B");
   break;
```

Έτσι όπως είναι δομημένο χρειάζεται
printf ("\nCorrespond …");

σε όλες τις *printf()* της *switch* χρειάζεται
\n για καλύτερο αισθητικό αποτέλεσμα

```
case (3):
      printf ("C");
    break;
    case (4):
      printf ("D");
    break;
    case (5):
      printf ("E");
    break;
    default:
      printf ("Ektos orion");
  }  // τέλος της switch
  printf (“Press 1 to continue");
  scanf ("%i",&a);
 } while (a!='1');  // τέλος της do-while
}  // τέλος της main()
```

Να μπει *break*, είναι καλύτερη προγραμματιστική τακτική

Σφάλμα πρώτο

Σφάλμα δεύτερο

Σφάλματα:

1) **Έπρεπε να μπει 1 κι όχι '1', γιατί η μεταβλητή είναι τύπου ακεραίου.**

2) **Η συνθήκη είναι while (a= = 1) κι όχι while (a! = 1), εφόσον η πληκτρολόγηση του 1 συνεπάγεται εκτέλεση εκ νέου του βρόχου.**

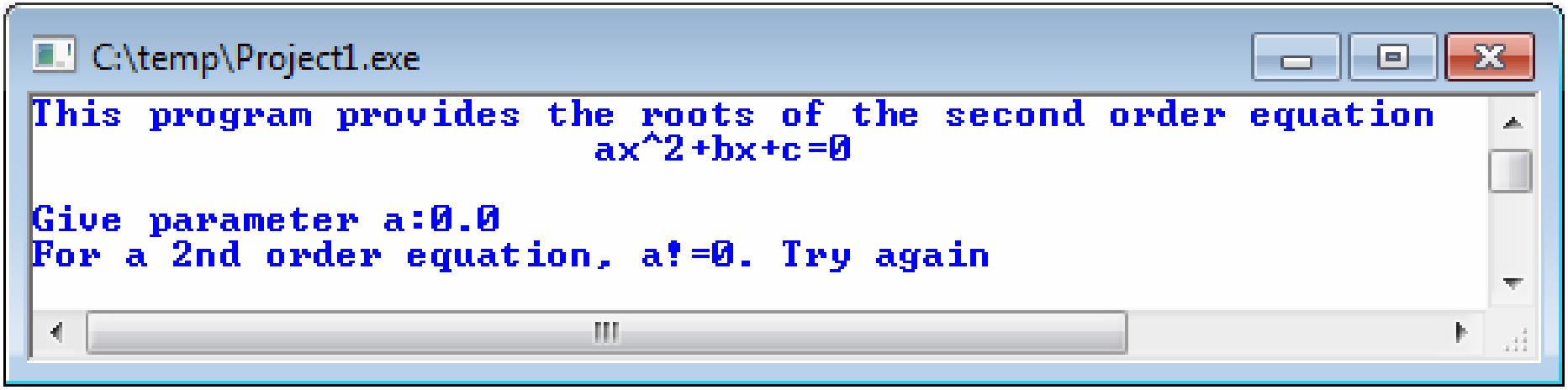
Παράδειγμα 3: **Να γραφεί πρόγραμμα που επιλύει δευτεροβάθμιες εξισώσεις. Να δέχεται ως είσοδο τους συντελεστές της εξίσωσης και να ελέγχει το είδος των ριζών (απλές πραγματικές, συζυγείς μιγαδικές ή διπλή πραγματική).**

```
#include<stdio.h>
#include<math.h>            // για sqrt(), fabs()
main()  {
  float a,b,c, D, r1,r2, r,im;
  printf( "This program provides the roots of the second order equation\n" );
  printf( "\t\t\tax^2+bx+c=0\n" );
  printf( "\nGive parameter a:");    scanf("%f",&a );
  while (fabs(a)<1e-10)  { // Έλεγχος για α=0, οπότε η εξίσωση γίνεται α/θμια
    printf( "For a 2nd order equation, a!=0. Try again\n" );
    printf( "\nGive parameter a:");  scanf("%f",&a );
  }  // τέλος της if
```

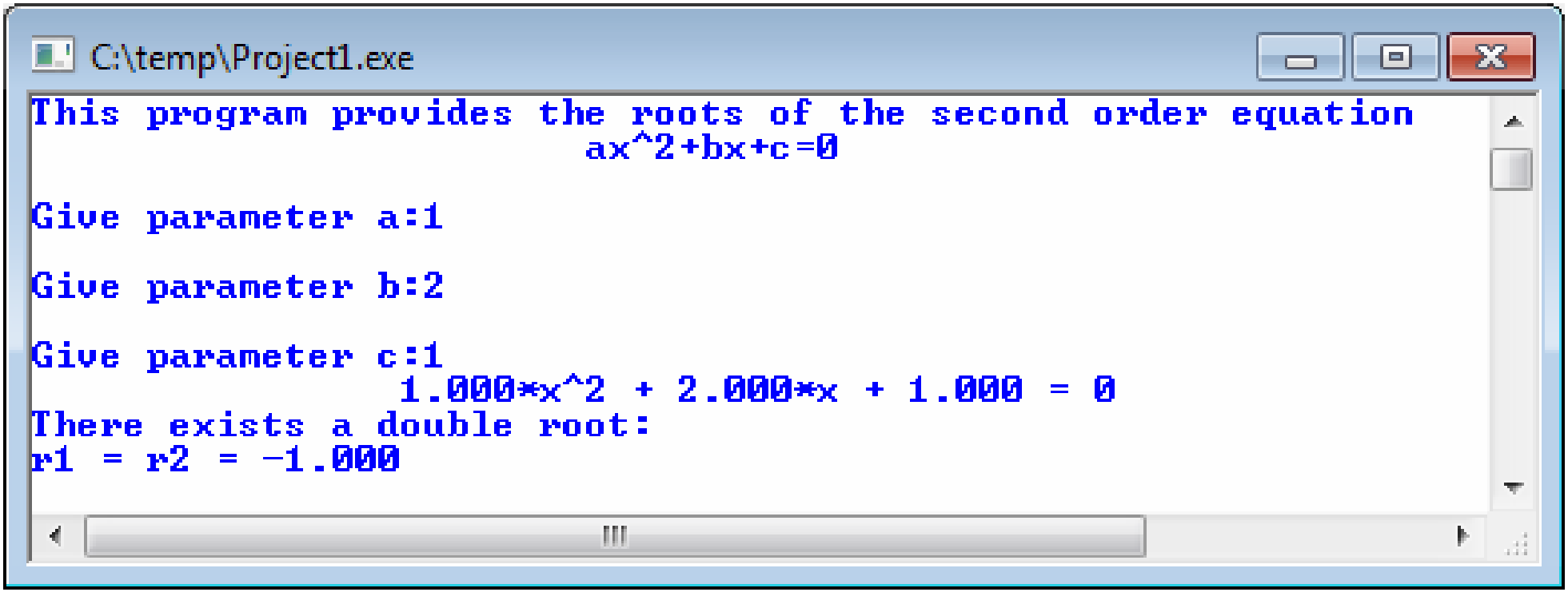
```
printf( "\nGive parameter b:"); scanf("%f",&b );
printf( "\nGive parameter c:"); scanf("%f",&c );
printf( "\t\t%.3f*x^2 + %.3f*x + %.3f = 0\n",a,b,c ) ;

D = b*b-4*a*c;    // Διακρίνουσα
if(D<0)  // Εάν Δ<0, οι ρίζες είναι συζυγείς μιγαδικές
{
  printf( "There exist two conjugate complex roots:\n" );
  r=-b/(2*a);
  im=sqrt(-D)/(2*a);
  printf( "r1 = %.3f + j%.3f\n",r,im );
  printf( "r2 = %.3f - j%.3f",r,im );
} // τέλος της if
else if (fabs(D)<1e-10)    // fabs -> float,   abs -> integer
{                          // Εάν Δ=0, υπάρχει διπλή ρίζα
  printf( "There exists a double root:\n" );
  printf( "r1 = r2 = %.3f\n", -b/(2*a) );
} // τέλος της else if
```

```
  else // Σε κάθε άλλη περίπτωση υπάρχουν δύο πραγματικές ρίζες
  {
   printf( "There exist two real roots:\n" );
   r1=(-b+sqrt(D))/(2*a);
   r2=(-b-sqrt(D))/(2*a);
   printf("r1=%.3f\nr2=%.3f\n",r1,r2 );
  } // τέλος της else
} // τέλος της main
```

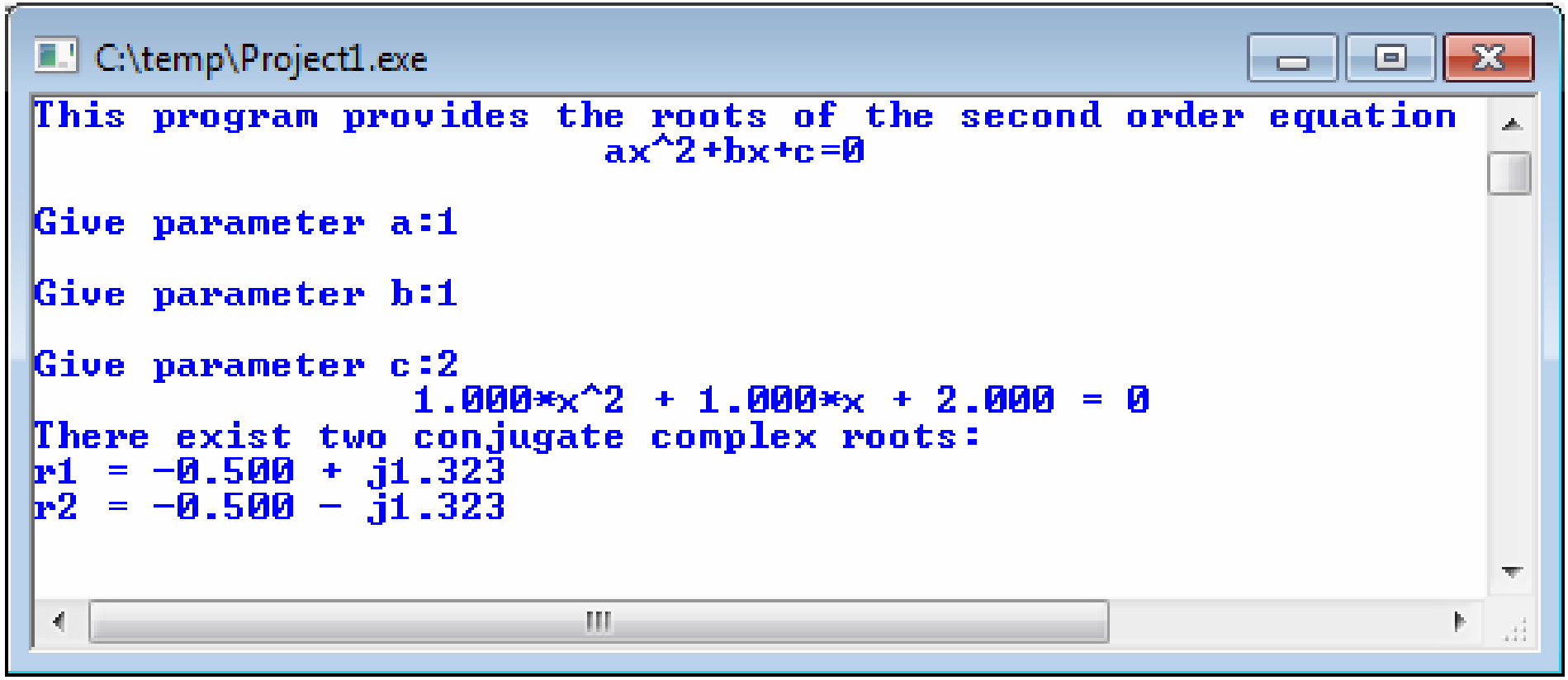
Αποτέλεσμα για α=0:



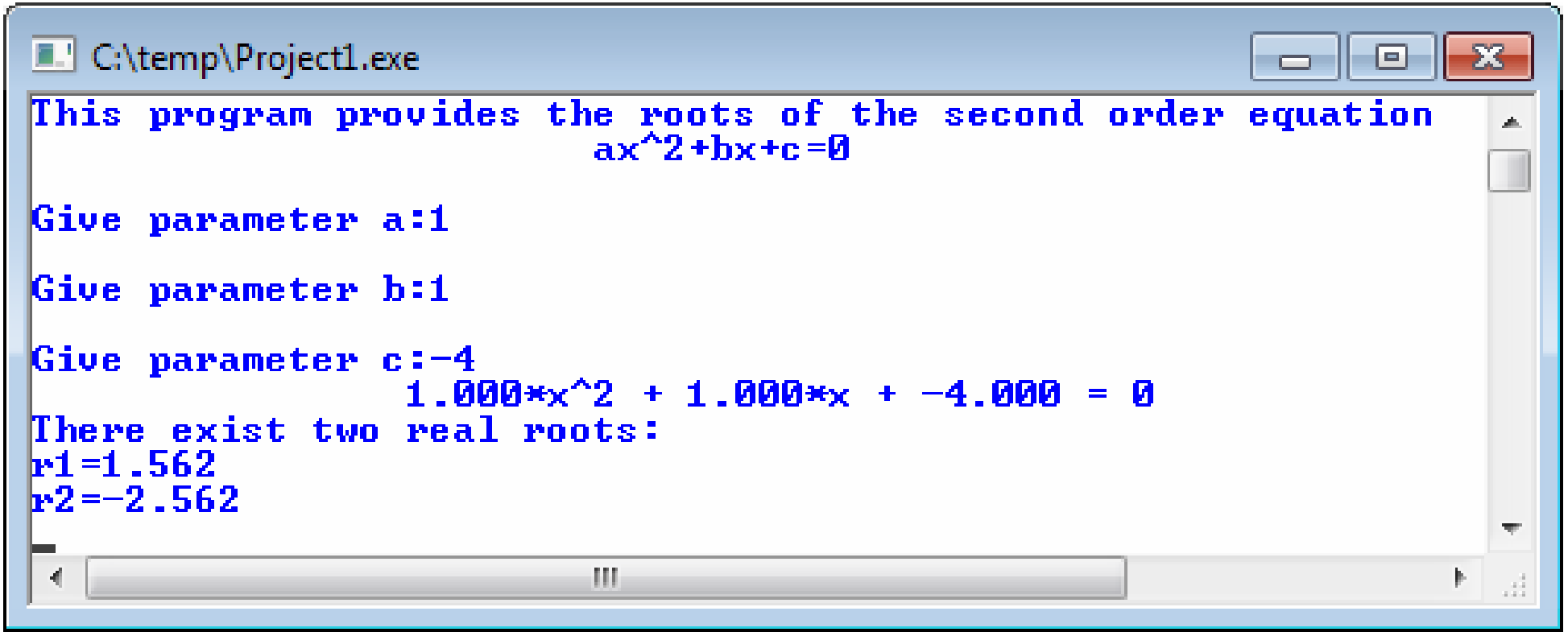
Αποτέλεσμα για διπλή πραγματική ρίζα:



Αποτέλεσμα για συζυγείς μιγαδικές ρίζες:



Αποτέλεσμα για απλές πραγματικές ρίζες:



Τροποποίηση του προηγούμενου προγράμματος. Επιτρέπει να εισαχθεί μηδενικός συντελεστής για τον μεγιστοβάθμιο όρο. Στην περίπτωση αυτή επιλύει πρωτοβάθμια εξίσωση:

ch_8_trinomial_ENHANCED.c

Παράδειγμα 4: Να γραφεί πρόγραμμα με το οποίο θα εισάγονται 6 πραγματικοί αριθμοί από το πληκτρολόγιο, θα αποθηκεύονται στον πίνακα array και θα τυπώνονται: α) οι θετικοί εξ αυτών, β) ο μεγαλύτερος, γ) ο αριθμός των στοιχείων του array, τα οποία έχουν τιμές στο διάστημα [1.05 50.8].

```
#include <stdio.h>
#include <math.h>
#define N 6
#define lower 1.05
#define upper 50.8
void main()  {
        float array[N],maxim;
        int i,count=0;
        for (i=0;i<N;i++)  {
                printf( "\nGive number %d: ",i+1 );
                scanf( "%f",&array[i] );
 }
```

```
maxim=array[0];
printf( "\n" );
for (i=0;i<N;i++)       // i=0 για να μπουν όλα σε ένα βρόχο,
                        //  μόνο για το μέγιστο θα ήταν i=1
{
  if (array[i]>0) printf( "array[%d]>0: %f\n",i,array[i] );
  if (array[i]>maxim) maxim=array[i];
  if ((array[i]>=lower) && (array[i]<=upper)) count++;
}
printf( "Maximum=%f\n",maxim );
printf( "Numbers within [lower,upper]: %d\n",count );
}
```

Αποτελέσματα: