

example_strchr.c

Παράδειγμα συναρτήσεων διαχείρισης αλφαριθμητικών

Στο πρόγραμμα που ακολουθεί υλοποιείται η συνάρτηση `void replace(char *pword, char *poldString, char *pnewString)`, στην οποία οι δείκτες σε χαρακτήρα `pword`, `poldString` και `pnewString` μέσω της κλήσης κατ' αναφορά χειρίζονται τα αλφαριθμητικά `word`, `fnd`, `repl`, τα οποία δίνει ο χρήστης στη συνάρτηση `main()`. Σε κάθε εμφάνιση του αλφαριθμητικού `fnd` μέσα στο αλφαριθμητικό `word`, η συνάρτηση αντικαθιστά το `fnd` με το αλφαριθμητικό `repl`. Χάριν ευκολίας, το μήκος του `fnd` είναι πάντοτε ίδιο με αυτό του `repl`. Αρχικά, πρέπει να γίνει έλεγχος κατά πόσον το `fnd` υπάρχει μέσα στο `word`.

Πέραν της ανάγνωσης των αλφαριθμητικών, η συνάρτηση `main()` επιτελεί τα ακόλουθα:

- Καλεί τη συνάρτηση `replace(,)` με ορίσματα τις διευθύνσεις των αλφαριθμητικών `word`, `fnd`, `repl`.
- Μετά το πέρας της συνάρτησης `replace()` τυπώνεται στην οθόνη το νέο περιεχόμενο του αλφαριθμητικού `word`.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include <stdlib.h>
```

```
void replace(char *pword, char *poldString, char *pnewString);
```

```
int main()
```

```
{
```

```
    char word[21],fnd[11],repl[11];
```

```
    printf( "Give a word: " );          scanf( "%s",word);
```

```
    printf( "\nGive a string to find: " ); scanf( "%s",fnd );
```

```
    printf( "Give a string to replace: " ); scanf( "%s",repl );
```

```
    replace(word,fnd,repl);
```

```
    printf( "\nThe new word is: %s",word);
```

```
    return 0;
```

```
}
```

```
void replace(char *pword, char *poldString, char *pnewString) {  
    int i;  
    char *temp;  
    temp=strstr(pword,poldString);  
    if (temp!=NULL) {  
        do {  
            for (i=0;i<strlen(poldString);i++) temp[i]=pnewString[i];  
            temp=temp+strlen(poldString);  
            temp=strstr(temp,poldString);  
        } while (temp!=NULL);  
    }  
    else {  
        printf( "\n'%s' is not contained in '%s'.\n\n",poldString,pword );  
        printf( "Press any key to finish",poldString,pword );  
    }  
}
```

Give a string to find: ak
Give a string to replace: al

The new word is: alalos

Give a string to find: ek
Give a string to replace: al

'ek' is not contained in 'akakos'.

Δείκτες σε δομές

Όπως κάθε μεταβλητή έτσι και μία μεταβλητή τύπου δομής (π.χ. δομή **addressT**) που ορίζεται από τη δήλωση

```
struct addressT addr1;
```

έχει διεύθυνση. Η διεύθυνση αυτή μπορεί να ληφθεί εφαρμόζοντας τον τελεστή διεύθυνσης στη μεταβλητή **addr1**. Εάν δηλωθεί ένας δείκτης σε δομή **addressT**, αυτός θα δείχνει στη μεταβλητή **addr1** με τη δήλωση και την ανάθεση τιμής:

```
stuct addressT *paddr;
```

```
paddr=&addr1;
```

πλέον ο δείκτης **paddr** θα δείχνει στη δομή **addr1**, παρέχοντας έναν εναλλακτικό τρόπο πρόσβασης στα μέλη της.



Δείκτες σε δομές

Η προσπέλαση ενός μέλους της δομής μέσω ενός δείκτη γίνεται με χρήση του **τελεστή βέλους** (αποτελείται από το «μείον» και το «μεγαλύτερο», **->**). Η πρόταση

```
printf( "%s\n",paddr->name );
```

τυπώνει το μέλος **name** της δομής **addr1** ενώ η πρόταση

```
paddr->zipCode=62124;
```

αναθέτει το **62124** στο μέλος **zipCode** της δομής **addr1**.

Η έκφραση **paddr->zipCode** είναι ισοδύναμη με την έκφραση **(*paddr).zipCode**. Οι παρενθέσεις είναι απαραίτητες επειδή ο τελεστής τελείας (.) έχει μεγαλύτερη προτεραιότητα από τον τελεστή (*).

Δείκτες εντός δομών

```
struct anynameT {  
    int *point1;  
    char *point2;  
    float var3;  
};  
  
int main() {  
    struct anynameT deikt;  
    int x=10;  char y;  
    deikt.point1=&x; /* Ο δείκτης deikt.point1 δείχνει στη διεύθυνση της x */  
    deikt.point2=&y; /* Ο δείκτης deikt.point2 δείχνει στη διεύθυνση της y */  
    *(deikt.point1)=13; /* Το περιεχόμενο της διεύθυνσης που δείχνει  
                        ο δείκτης deikt.point1 γίνεται 13 */  
    .....  
    return 0;  
}
```

Παράδειγμα: Το ακόλουθο πρόγραμμα αφορά σε δείκτες που δείχνουν σε δομές και παρουσιάζει συγκριτικά τον τρόπο λειτουργίας της κλήσης κατ' αξία και της κλήσης κατ' αναφορά.

```
#include<conio.h>                                //vectors
#include<stdio.h>

struct vector //define the structure vector
{
    float x,y;
};
//-----
// function declaration
vector readvect(); // Reads a vector, call by value (cbv)
void prvect(char d, vector v); // Prints a vector, (cbv)
void scanvect( vector *p); // Reads a vector,call by reference (cbr)
float inprod(vector v, vector u); // Inner product, (cbv)
float inprodr(vector *p, vector *q); // Inner product, (cbr)
vector addvect(vector v, vector u); // Add vectors, (cbv)
vector addvectr(vector *p, vector *q); // Add vectors, (cbr)
```



```
main()
{
    vector a,b,c;
    a=readvect();
    prvect('a',a);
    scanvect(&b);
    prvect('b',b);
    printf("v a*b=%.2f\n",inprod(a,b));
    printf("r a*b=%.2f\n",inprodr(&a,&b));
    c = addvect(a,b);
    printf("cbv addition: ");
    prvect('c',c);
    addvectr(&a,&b,&c);
    printf("cbr addition: ");
    printf("Vector %c is (%.2f,%.2f)\n",'c',c.x,c.y);
    printf("\t\t\tPRESS ANY KEY TO FINISH\n" );
    getch();
} //end of main
```

```
void prvect(char d, vector v)
{
    printf( "Vector %c is ",d );
    printf( "(%.2f,%.2f)\n",v.x,v.y );
} //end of prvect

vector readvect()
{
    vector v;
    printf( "Give the x co-ordinate:" ); scanf("%f",&v.x);
    printf( "Give the y co-ordinate:" ); scanf("%f",&v.y);
    return(v);
} //end of readvect

void scanvect( vector *p)
{
    printf( "Give the x co-ordinate:" ); scanf("%f",&p->x);
    printf( "Give the y co-ordinate:" ); scanf("%f",&p->y);
} //end of scanvect
```

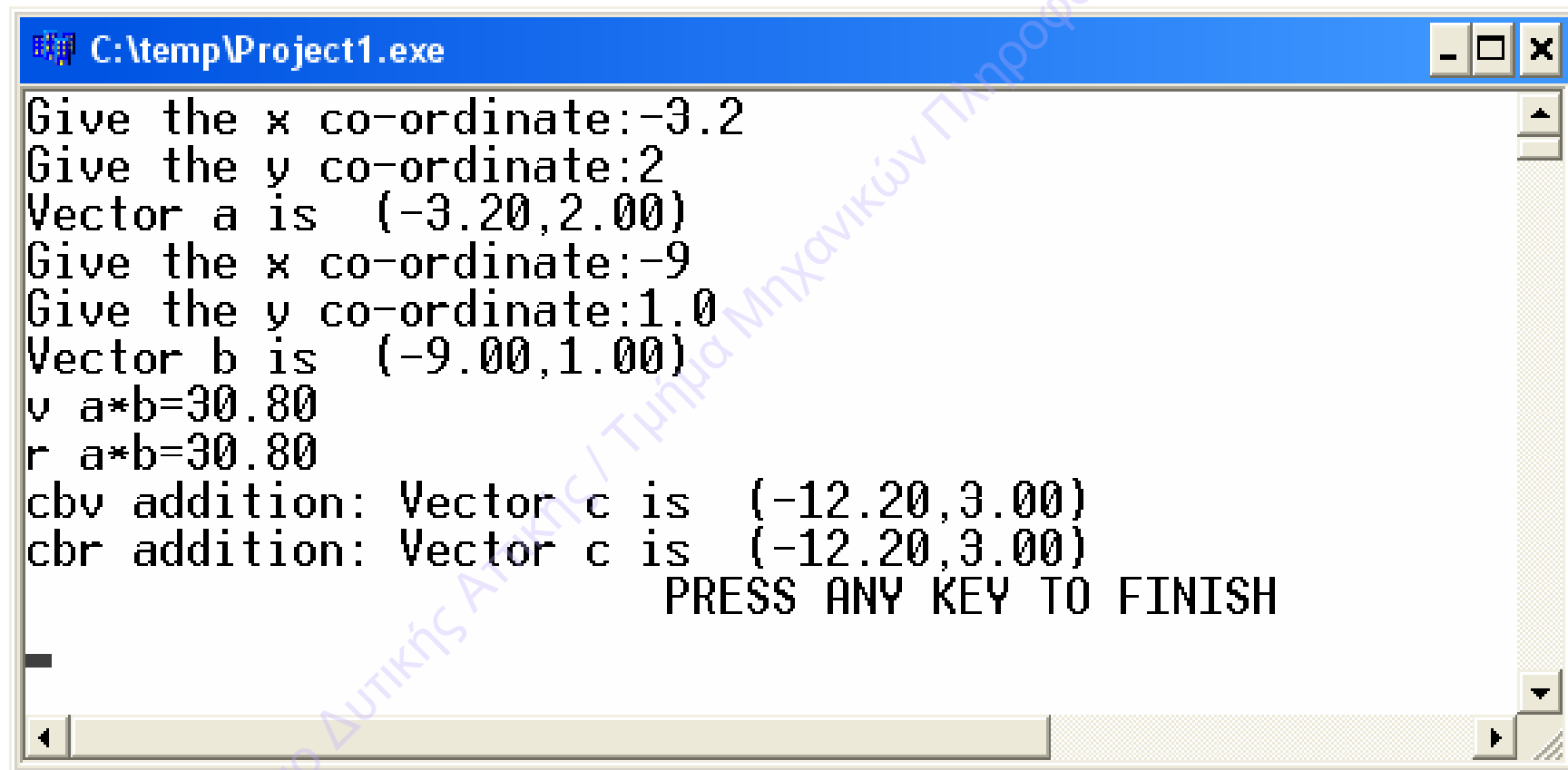
```
float inprod(vector v, vector u)
{
    return(v.x*u.x+v.y*u.y);
} //end of inprod

float inprodr(vector *p, vector *q)
{
    // Δύο διαφορετικοί τρόποι γραφής των μελών της δομής με χρήση δείκτη
    return((*p).x*(*q).x+(p->y)*(q->y));
} //end of inprodr

vector addvect(vector v, vector u)
{
    vector sum;
    sum.x=v.x+u.x; sum.y=v.y+u.y;
    return(sum);
} //end of addvect

void addvectr(vector *p, vector *q, vector *t)
{
    t->x=(p->x)+(q->x); t->y=(p->y)+(q->y);
} //end of addvectr
```

Αποτέλεσμα:



```
C:\temp\Project1.exe
Give the x co-ordinate:-3.2
Give the y co-ordinate:2
Vector a is (-3.20,2.00)
Give the x co-ordinate:-9
Give the y co-ordinate:1.0
Vector b is (-9.00,1.00)
v a*b=30.80
r a*b=30.80
cbv addition: Vector c is (-12.20,3.00)
cbr addition: Vector c is (-12.20,3.00)
PRESS ANY KEY TO FINISH
```

Παράδειγμα:

Να αναπτυχθεί πρόγραμμα που να λαμβάνει από το πληκτρολόγιο τα στοιχεία ενός εργαζόμενου και να δημιουργεί πίνακα εργαζόμενων, με τύπο δεδομένου κατάλληλη δομή. Η διαδικασία θα επαναλαμβάνεται για 10 διαφορετικούς εργαζόμενους, όσο είναι και το μέγεθος του πίνακα. Οι πληροφορίες που διαβάζονται για κάθε εργαζόμενο είναι:

Ονοματεπώνυμο: Όνομα και επώνυμο ξεχωριστά (*σε μεταβλητή τύπου δομής*)

Διεύθυνση: Όνομα οδού, αριθμός οδού, πόλη, ταχ. Κώδικας (*σε μεταβλητή τύπου δομής*)

Τηλέφωνο: Τηλέφωνο εργασίας, κινητό (*σε μεταβλητή τύπου δομής*)

Θέση: Τίτλος, κωδικός αριθμός εργαζόμενου, τομέας της επιχείρησης στον οποίο εργάζεται, αριθμός γραφείου, ονοματεπώνυμο προϊσταμένου, ημερομηνία πρόσληψης (ημέρα, μήνας, έτος), μισθός (*σε μεταβλητή τύπου δομής – θα απαιτηθεί ένθετη δομή για την ημερομηνία πρόσληψης*)

Στη συνέχεια να δίνεται κάποιος κωδικός αριθμός εργαζόμενου από το πληκτρολόγιο και να αναζητείται στον πίνακα. Αν υπάρχει, τότε να εμφανίζονται στην οθόνη οι πληροφορίες του αντίστοιχου εργαζόμενου, αλλιώς να εμφανίζεται ένα ανάλογο μήνυμα.

// Combined use of calling functions by value and by reference (including structures)

```
#include<conio.h>
```

```
#include<stdio.h>
```

```
#define N 2//10 //Number of employees
```

```
struct nmT {
```

```
    char name[40],surname[40];
```

```
};
```

```
struct addressT {
```

```
    char street_name[40], city[40];
```

```
    int street_number,zip_code;
```

```
};
```

```
struct teleT {
```

```
    char office_number[14],home_number[14];
```

```
};
```

```
struct hiredateT {
```

```
    int year,month,date;
```

```
};
```

```
struct job_descriptionT
```

```
{
```

```
    char title[40], sector[100], boss_name[40];
```

```
    int code_number,office_number,salary;
```

```
    hiredateT hire;
```

```
};
```

```
struct personnelT
{
    nmT nm;
    addressT addr;
    teleT tele;
    job_descriptionT job;
};
```

```
void get_employee(personnelT *pers_ptr);
void search_employee(int i, personnelT person[N]);
```

```
main()
{
    personnelT pers[N];
    int i;
    for (i=0;i<N;i++) get_employee(&pers[i]);
    printf("\nGive an employee's code_number: ");
    scanf("%d",&i);
    search_employee(i,pers);
}
```

Call by reference

Call by value

```
void get_employee(personnelT *pers_ptr)
{
    printf("\t\tAdd new employee:\n");

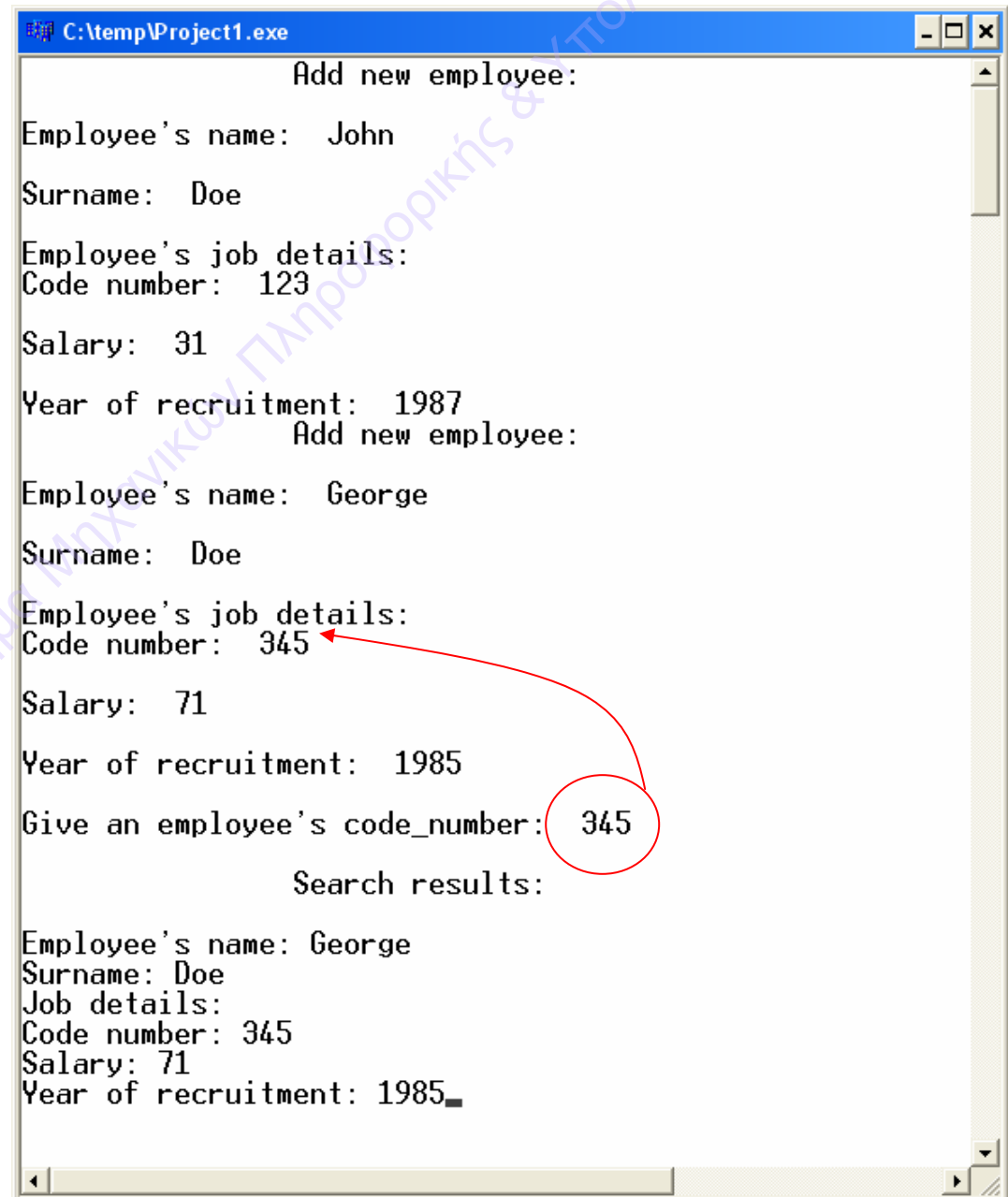
    printf("\nEmployee's name: ");
    scanf("%s",pers_ptr->nm.name);
    printf("\nSurname: ");
    scanf("%s",pers_ptr->nm.surname);

    printf("\t\nEmployee's job details:");
    printf("\nCode number: ");
    scanf("%d",&pers_ptr->job.code_number);
    printf("\nSalary: ");
    scanf("%d",&pers_ptr->job.salary);
    printf("\nYear of recruitment: ");
    scanf("%d",&pers_ptr->job.hire.year);
}
```



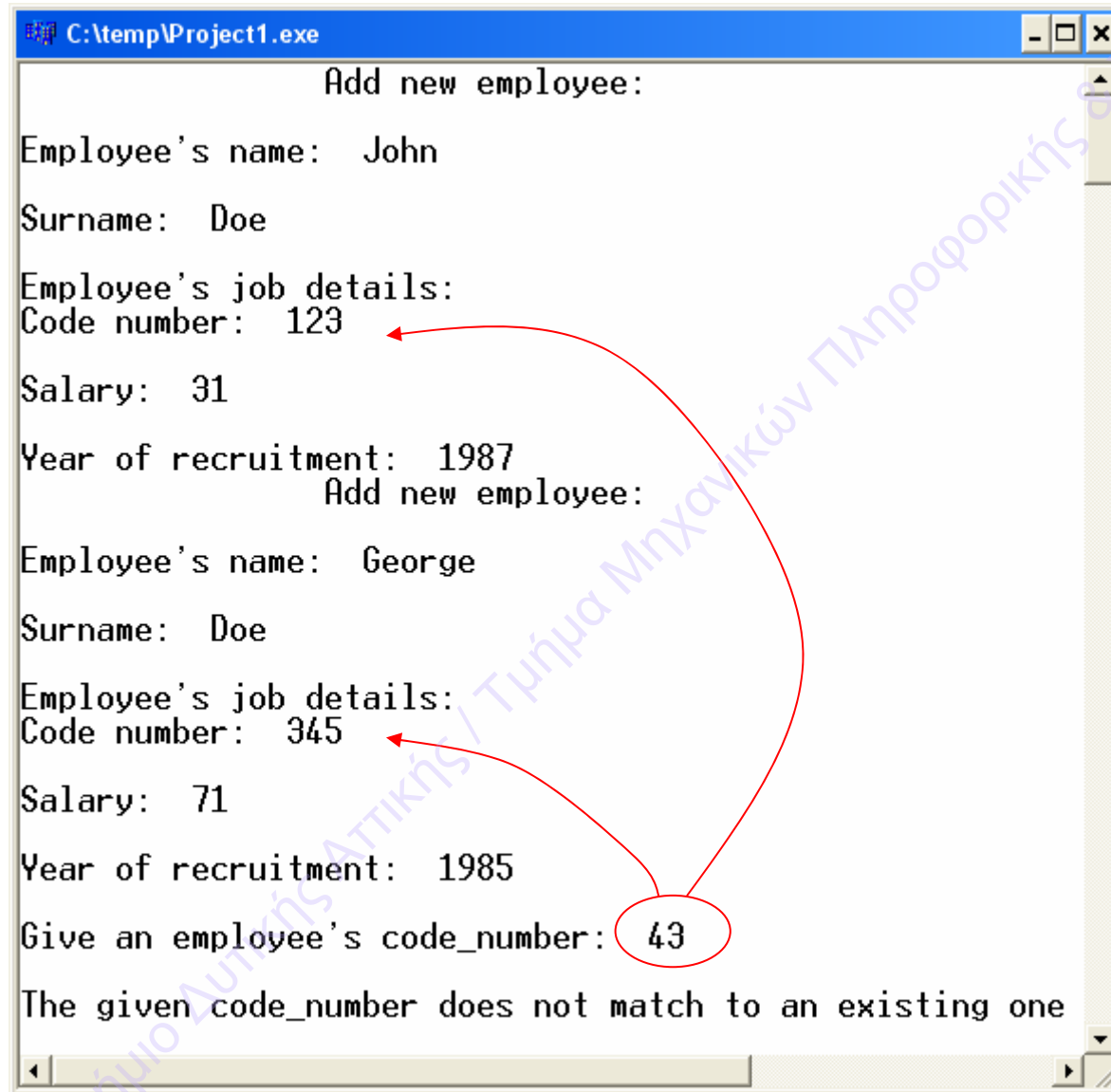
```
void search_employee(int code, personnelT person[N])
{
    int j=0;
    int index=-1;
    while ((j<N) && (index== -1))
    {
        if (code==person[j].job.code_number) index=j;
        j++;
    }
    if (index== -1)
        printf("\nThe given code_number does not match to an existing one\n");
    else
    {
        printf("\n\t\tSearch results:\n");
        printf("\nEmployee's name: %s",person[index].nm.name);
        printf("\nSurname: %s",person[index].nm.surname);
        printf("\n\t\tJob details:");
        printf("\nCode number: %d",person[index].job.code_number);
        printf("\nSalary: %d",person[index].job.salary);
        printf("\nYear of recruitment: %d",person[index].job.hire.year);
    }
}
```

Αποτελέσματα:



```
C:\temp\Project1.exe

Add new employee:
Employee's name: John
Surname: Doe
Employee's job details:
Code number: 123
Salary: 31
Year of recruitment: 1987
Add new employee:
Employee's name: George
Surname: Doe
Employee's job details:
Code number: 345
Salary: 71
Year of recruitment: 1985
Give an employee's code_number: 345
Search results:
Employee's name: George
Surname: Doe
Job details:
Code number: 345
Salary: 71
Year of recruitment: 1985
```



```
C:\temp\Project1.exe

Add new employee:

Employee's name: John
Surname: Doe
Employee's job details:
Code number: 123
Salary: 31
Year of recruitment: 1987
Add new employee:

Employee's name: George
Surname: Doe
Employee's job details:
Code number: 345
Salary: 71
Year of recruitment: 1985
Give an employee's code_number: 43
The given code_number does not match to an existing one
```

Ορίσματα γραμμής διαταγής

Όπως κάθε συνάρτηση, έτσι κι η *main* μπορεί να δεχθεί παραμέτρους, οι οποίες επιτρέπουν να δίνουμε στο καλούμενο πρόγραμμα ένα σύνολο από εισόδους, που καλούνται **ορίσματα γραμμής διαταγής**. Ο μηχανισμός περάσματος ορισμάτων βασίζεται στην ακόλουθη δήλωση της *main*:

```
main(int argc, char *argv[])  
{  
    ...  
}
```

- **argc** (*argument count*): ο αριθμός των ορισμάτων της γραμμής διαταγής, συμπεριλαμβανομένου και του ονόματος του προγράμματος.
- **argv** (*argument vector*): δείκτης σε πίνακα από δείκτες, που δείχνουν στα ορίσματα της γραμμής διαταγής, τα οποία αποθηκεύονται με τη μορφή αλφαριθμητικών. Ο πίνακας στον οποίο δείχνει ο *argv* έχει ένα επιπλέον στοιχείο, το *argv[argc]*, το οποίο έχει τιμή *NULL* (είναι δείκτης που δε δείχνει πουθενά).

Ορίσματα γραμμής διαταγής

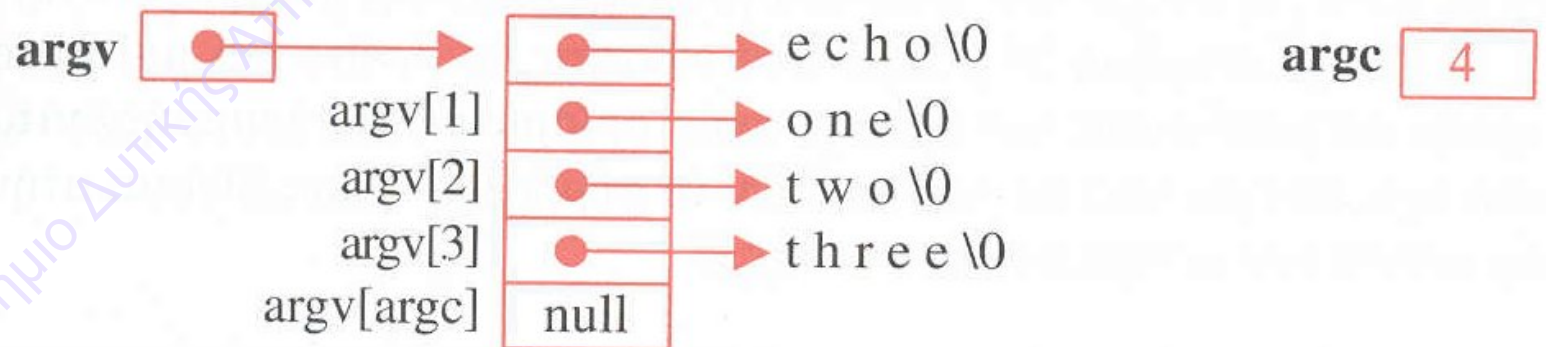
Παράδειγμα: Να καταστρωθεί πρόγραμμα που θα τυπώνει τα ορίσματα της γραμμής διαταγής

Λύση: Έστω *echo* το όνομα του προγράμματος. Όταν το καλούμε με την εντολή

echo one two three

θα πρέπει να εκτελείται και να τυπώνει στην οθόνη *one two three*.

Το σώμα της *main* έχει πρόσβαση στις μεταβλητές *argc* και *argv* όπως παριστάνονται από το ακόλουθο σχήμα:



Ορίσματα γραμμής διαταγής

με βάση το προηγούμενο σχήμα, η διαμόρφωση του σώματος της *main* είναι απλή:

```
main(int argc, char *argv[])  
{  
    int i;  
    for (i=1;i<argc;i++) printf( "%s%s",argv[i], " " );  
    printf( "\n" );  
}
```

Η `printf("%s%s",argv[i], " ");` τυπώνει μετά από κάθε όρισμα ένα κενό. Εάν θέλουμε να μην τυπώνεται το κενό μετά το τελευταίο όρισμα, η πρόταση πρέπει να διαμορφωθεί όπως παρακάτω:

```
for (i=1;i<argc;i++) printf("%s%s",argv[i],(i<argc-1)? " ":"");
```



Ορίσματα γραμμής διαταγής

Εάν χρησιμοποιήσουμε τη σημειολογία των δεικτών αντί αυτής του πίνακα, η πρόταση μπορεί να γραφεί ως εξής:

```
while (--argc) printf( "%s%s",*++argv,(i<argc-1)?" ":"" );
```

Εναλλακτικά, θα μπορούσαμε να γράψουμε:

```
main(int argc, char *argv[])  
{  
while (--argc) printf( (argc>1)?"%s":"%s",*++argv );  
}
```

Δυσνόητο; δοκιμάστε το!!

Θεματική ενότητα 11:

Αρχεία

Τα κανάλια *stdin*, *stdout*, *stderr*

- Κάθε φορά που ξεκινά η εκτέλεση ενός προγράμματος, ο υπολογιστής ανοίγει αυτόματα το **κανάλι καθιερωμένης εισόδου *stdin*** (standard input), το **κανάλι καθιερωμένης εξόδου *stdout*** (standard output) και το **κανάλι σφαλμάτων *stderr*** (standard errors). Γενικά αυτά τα κανάλια αναφέρονται στην κονσόλα αλλά το λειτουργικό σύστημα μπορεί να τα ανακατευθύνει σε κάποια άλλη συσκευή.
- Επειδή πρόκειται για δείκτες αρχείου το σύστημα I/O **με ενδιάμεση αποθήκευση (buffering)** μπορεί να χρησιμοποιεί αυτά τα κανάλια για να εκτελεί λειτουργίες I/O στην κονσόλα.
- Το *stdin* χρησιμοποιείται για ανάγνωση από την κονσόλα και τα *stdout*, *stderr* για εγγραφή στην κονσόλα. Μπορούν να χρησιμοποιηθούν τα *stdin*, *stdout*, *stderr* ως δείκτες αρχείου σε οποιαδήποτε συνάρτηση χρησιμοποιεί μία μεταβλητή τύπου **FILE ***.

Τα κανάλια *stdin*, *stdout*, *stderr*

- Μπορούν να ανακατευθυνθούν τα κανάλια και να γράφονται τα μηνύματα λάθους σε αρχείο αντί να εμφανίζονται στην οθόνη.
- Με την εντολή *program_name < filename* ορίζεται ως κύρια είσοδος αντί του πληκτρολογίου το αρχείο **filename**.
- Με την εντολή *program_name > filename* ορίζεται ως κύρια έξοδος αντί για την οθόνη το αρχείο **filename**.
- Οι παραπάνω εντολές δίνονται από τη γραμμή διαταγής (command line). Η *printf* γράφει στο **stdout** και η *scanf* στο **stdin**.

Τα *stdin*, *stdout*, *stderr* δεν είναι μεταβλητές αλλά σταθερές και δεν μπορούν να αλλαχθούν. Όπως ο υπολογιστής δημιουργεί αυτόματα αυτούς τους δείκτες αρχείου στην αρχή του προγράμματος, έτσι και τους αποσύρει αυτόματα στο τέλος του προγράμματος. Δε θα πρέπει να κλείσουν αυτά τα κανάλια με παρέμβαση του χρήστη.

Δύο ευρείες τάξεις αρχείων

- **Δυαδικά αρχεία (binary files):**

- Αποθηκεύουν **κάθε** τύπο δεδομένου: οριζόμενο από το χρήστη, char, int, float, double, string, data struct, κ.λ.π.
- Συνήθως ΔΕΝ είναι αναγνώσιμα από τους συντάκτες (editors)
- Συνήθως ΔΕΝ είναι φορητά (δεν ανοίγουν σε όλα τα μηχανήματα)

- **Αρχεία κειμένου (text files)**

- Αποθηκεύουν μία ακολουθία (ένα 'ρεύμα - stream') από bytes χαρακτήρων.
- Είναι αναγνώσιμα από τους συντάκτες (π.χ. αρχεία **.h**, **.c**)
- Είναι φορητά σε κάθε υπολογιστή (σχεδόν)

- Στα αρχεία κειμένου τα πάντα αποθηκεύονται ως ακολουθίες χαρακτήρων (τα *stdin*, *stdout* είναι ανοικτά ως κανάλια κειμένου).
- Στα δυαδικά αρχεία ο μεταγλωττιστής δεν κάνει μεταγλώττιση των bytes, απλά διαβάζει και γράφει bits, ακριβώς όπως αυτά εμφανίζονται.

Παράδειγμα:

Ο αριθμός 12345 εγγράφεται ως αλφαριθμητικό σε αρχείο κειμένου, απαιτώντας 6 bytes (1 για κάθε χαρακτήρα κι ένα για το χαρακτήρα τερματισμού '\0'). Αντίθετα, σε ένα δυαδικό αρχείο εγγράφεται ως ακέραιος, απαιτώντας 4 bytes.

Άνοιγμα – κλείσιμο αρχείου

- ΠΑΝΤΟΤΕ **να ανοίγετε** ένα αρχείο πριν τη χρήση,
- ΠΑΝΤΟΤΕ **να το κλείνετε** όταν περατώνεται η χρήση του.

```
FILE *pF;    /* δήλωση ενός pointer σε μεταβλητή FILE */
```

```
pF = fopen("myfile.txt", "r");    /* άνοιγμα αρχείου */
```

```
/* . . . Διάφορες λειτουργίες . . . */
```

```
fclose(pF);
```

Το όνομα αρχείου εισέρχεται ως string
(ο δείκτης δείχνει στον πρώτο χαρακτήρα του)

Προσδιοριστής string που ελέγχει το
είδος της πρόσβασης

Άνοιγμα – κλείσιμο αρχείου

```
FILE *pF;    /* δήλωση ενός pointer σε μεταβλητή FILE */
```

```
pF = fopen("myfile.txt", "r");    /* άνοιγμα αρχείου */
```

```
/* . . . Διάφορες λειτουργίες . . . */
```

```
fclose(pF);
```

Επιστρεφόμενη τιμή: 'pointer-to-FILE'
(ή NULL σε περίπτωση σφάλματος)

Άνοιγμα – κλείσιμο αρχείου

```
FILE *pF; /* δήλωση ενός pointer σε μεταβλητή FILE */  
pF = fopen("myfile.txt", "r"); /* άνοιγμα αρχείου */  
/* . . . Διάφορες λειτουργίες . . . */  
fclose(pF);
```

Κλείσιμο αρχείου: Επιστρέφει 0 όταν κλείνει σωστά
ή EOF όταν υπάρχει σφάλμα

Άνοιγμα – κλείσιμο αρχείου

```
FILE *pF;    /* δήλωση ενός pointer σε μεταβλητή FILE */  
pF = fopen("myfile.txt", "r");    /* άνοιγμα αρχείου */  
/* . . . Διάφορες λειτουργίες . . . */  
fclose(pF);
```


Εάν θέλουμε ολόκληρη τη διαδρομή μέσα στο μέσο αποθήκευσης:
Π.χ. `pF = fopen("c:\\teicm\\progrII\\myfile.txt", "r");`

Άνοιγμα – κλείσιμο αρχείου

- Η **fopen** δεσμεύει τους απαραίτητους πόρους από το λειτουργικό σύστημα, δημιουργεί το κανάλι επικοινωνίας κι επιστρέφει στο πρόγραμμα που την κάλεσε ένα δείκτη, που δείχνει σε δομή τύπου **FILE**.
- Ο δείκτης, που δείχνει σε δομή τύπου **FILE**, είναι γνωστός ως **διαχειριστής** ή **δείκτης αρχείου** (**file handler** ή **file descriptor**). Χρησιμοποιείται για να κρατήσει την ταυτότητα του καναλιού που επιστρέφεται από την **fopen**.
- Η **FILE** ορίζεται στο **<stdio.h>**.
- Ένα από τα πεδία της δομής **FILE** είναι ο **δείκτης θέσης αρχείου** (**file position indicator**), ο οποίος δείχνει στο byte από όπου ο επόμενος χαρακτήρας πρόκειται να διαβασθεί ή όπου ο επόμενος χαρακτήρας πρόκειται να εγγραφεί.

Άνοιγμα – κλείσιμο αρχείου

Παράμετροι προσδιορισμού του τρόπου πρόσβασης σε αρχεία κειμένου:

- **'r'**: άνοιγμα αρχείου για ανάγνωση. Ο δείκτης θέσης αρχείου βρίσκεται στην αρχή του κειμένου.
 - **'w'**: δημιουργία νέου αρχείου για εγγραφή. Εάν το αρχείο υπάρχει ήδη, το μέγεθός του θα μηδενισθεί και τα περιεχόμενα θα διαγραφούν. Ο δείκτης θέσης αρχείου τίθεται στην αρχή του αρχείου.
 - **'a'**: άνοιγμα υπάρχοντος αρχείου κειμένου, στο οποίο όμως μπορούμε να γράψουμε μόνο στο τέλος του αρχείου.
 - **'r+'**: άνοιγμα υπάρχοντος αρχείου κειμένου για ανάγνωση και εγγραφή. Ο δείκτης θέσης αρχείου τίθεται στην αρχή του αρχείου.
 - **'w+'**: δημιουργία νέου αρχείου για ανάγνωση και εγγραφή. Εάν το αρχείο υπάρχει ήδη, το μέγεθός του θα μηδενισθεί και τα περιεχόμενα θα διαγραφούν.
- 

Παράμετροι προσδιορισμού του τρόπου πρόσβασης σε αρχεία κειμένου (συνέχεια):

- **‘a+’**: άνοιγμα υπάρχοντος αρχείου ή δημιουργία νέου σε append μορφή. Μπορούμε να διαβάσουμε δεδομένα από οποιοδήποτε σημείο του αρχείου, αλλά μπορούμε να γράψουμε δεδομένα μόνο στη θέση του δείκτη end-of-file.

Οι προσδιοριστές για τα δυαδικά αρχεία μορφή είναι ίδιοι, με τη διαφορά ότι έχουν ένα **b** που τους ακολουθεί. Έτσι, για να ανοίξουμε ένα δυαδικό αρχείο και να διαβάσουμε, θα πρέπει να χρησιμοποιήσουμε τον προσδιοριστή **‘rb’**.