ΤΕΧΝΟΛΟΓΙΑ ΚΑΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ – ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ενώσεις (Unions) -1

Μία ένωση (union) μπορεί να περιέχει αντικείμενα διαφορετικών τύπων και μεγεθών

π.χ.
```
union u_tag   {
                int ival;
                float fval;
                char *sval;
                } u;
```

Ενώσεις (Unions) -2

```
if (utype == INT)
    printf("%d\n", u.ival);
else if (utype==FLOAT)
  printf("%f\n", u.fval);
else if (utype==STRING)
  printf("%s\n", u.sval);
```

Στις ενώσεις επιτρέπεται απόδοση τιμής ή αντιγραφή σαν ενότητα, λήψη διεύθυνσης, προσπέλαση μέλους.

Η αρχική τιμή πρέπει να έχει τύπο ίδιο με αυτό του πρώτου της μέλους.

Ορίσματα Γραμμής Διαταγών (1)

```
#include <stdio.h>
main(int argc, char * argv[])
{
        int i;
        for (i=1; i<argc; i++)
        printf("%s%s", argv[i], (i<argc-1)? " " : " \n");
        return 0;
 }
```

Ορίσματα Γραμμής Διαταγών (2)

```
#include <stdio.h>
main(int argc, char * argv[])
{
        while (--argc >0)
        printf("%s%s", *++argv, (argc>1)?"  ": "\n ");
        return 0;
 }
```

Δείκτες σε Συναρτήσεις

Στην C η συνάρτηση δεν είναι μεταβλητή. Μπορούμε να ορίσουμε δείκτες σε συναρτήσεις, που μπορούν να αποδίδονται σαν τιμές, να τοποθετούνται σε πίνακες, να μεταβιβάζονται σε συναρτήσεις και να επιστρέφονται από αυτές.

Παράδειγμα (1)

```
#include <stdio.h>
#include <string.h>
#define MAXLINES 5000
char * lineptr[MAXLINES];
int readlines(char *lineptr[], int nlines);
void writelines(char *lineptr[], int nlines);
void qsort(void *lineptr[], int left, int right, int (*comp)(void *, void *));
int numcmp(char *, char *);

main(int argc, char *argv[])
{  int nlines; int numeric=0;
   If (argc>1 && strcmp(argv[1],"-n")==0) numeric=1;
   If ((nlines=readlines(lineptr,MAXLINES))>=0) {
qsort((void **) lineptr,0,nlines-1, (int (*)(void *, void *))(numeric?numcmp:strcmp));
writelines(lineptr,nlines);
return 0;    }                                          }
else  {printf("Η είσοδος είναι μεγάλη"); return 1; }
} }
```

Παράδειγμα (2)

```
void swap(char *v[], int, int);

void qsort(char *v[], int left, int right, int (*comp)(void *,void *))
{
    int i, last;
    if (left>=right) return;
    swap(v, left, (left+right)/2);
    last=left;
    for (i=left+1; i<=right; i++)
        if ((*comp)(v[i],v[left])<0)
            swap(v,++last,i);
    swap(v, left, last);
    qsort(v, left, last-1, comp);
    qsort(v, last+1, right, comp);
}
```

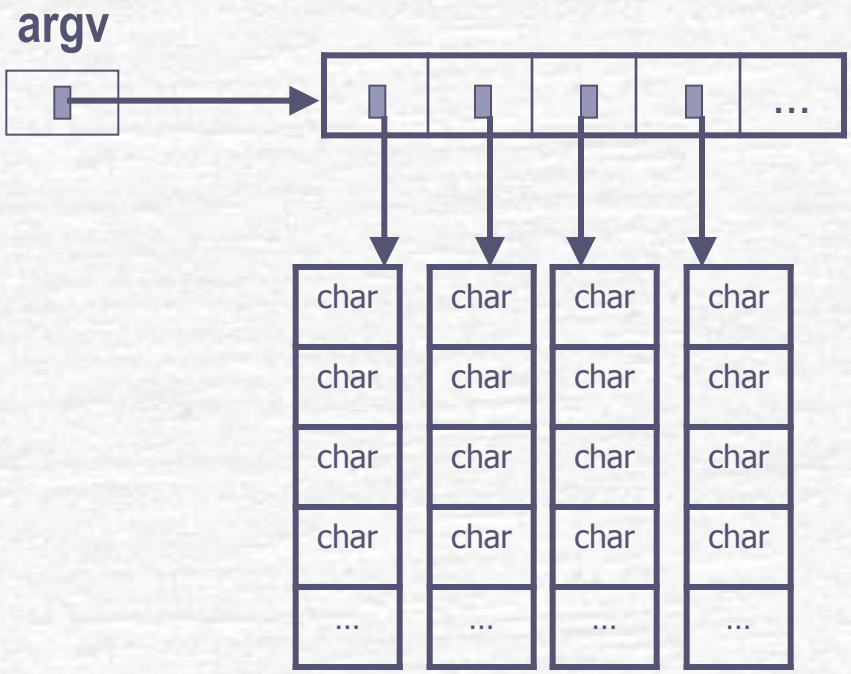
Παράδειγμα (3)

```
int numcmp(char *s1, char *s2)
{
    double v1, v2;
    v1=atof(s1);
    v2=atof(s2);
    if (v1<v2) return -1;
    else if (v1>v2) return 1;
    else return 0;
}
 void swap(void *v[], int i, int j)
 {   void *temp;
     temp=v[i];
     v[i]=v[j];
     v[j]=temp;
 }
```

Παραδείγματα Δηλώσεων

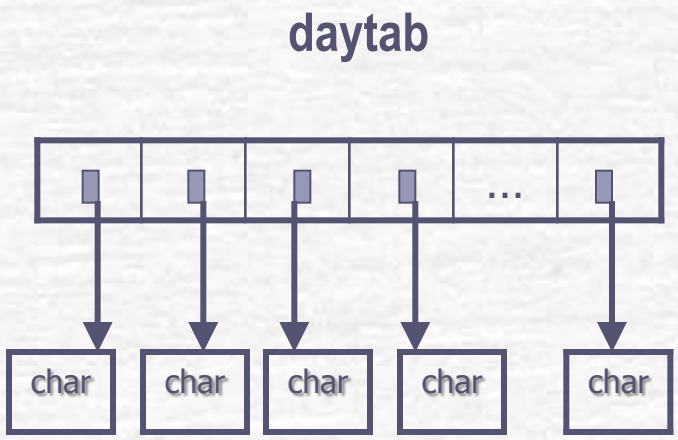
char **argv	δείκτης σε δείκτη σε char
int (*daytab)[13]	δείκτης σε πίνακα[13] int
int *daytab[13]	πίνακας[13] με δείκτες σε int
void *comp()	συνάρτηση που επιστρέφει δείκτη σε void
void (*comp)()	δείκτης σε συνάρτηση που επιστρέφει void
char (*(*x())[])()	συνάρτηση που επιστρέφει δείκτη σε πίνακα δεικτών σε συνάρτηση που επιστρέφει char
char (*(*x[3])())[5]	πίνακας [3] με δείκτες σε συνάρτηση που επιστρέφει δείκτη σε πίνακα [] char

char **argv

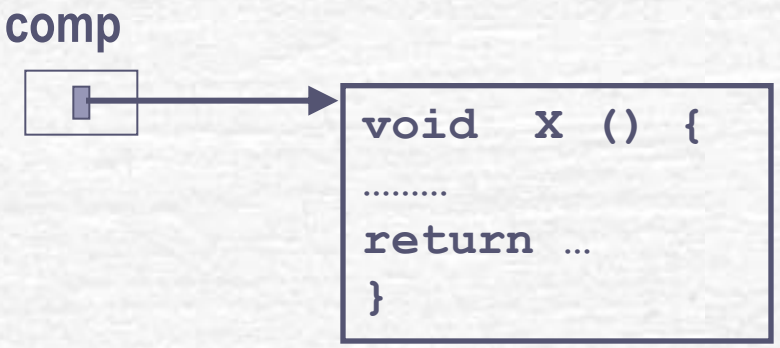

argv
...
char
char
char
char
...
char
char
char
char
...
char
char
char
char
...
char
char
char
char
...

char (*daytab)[13]

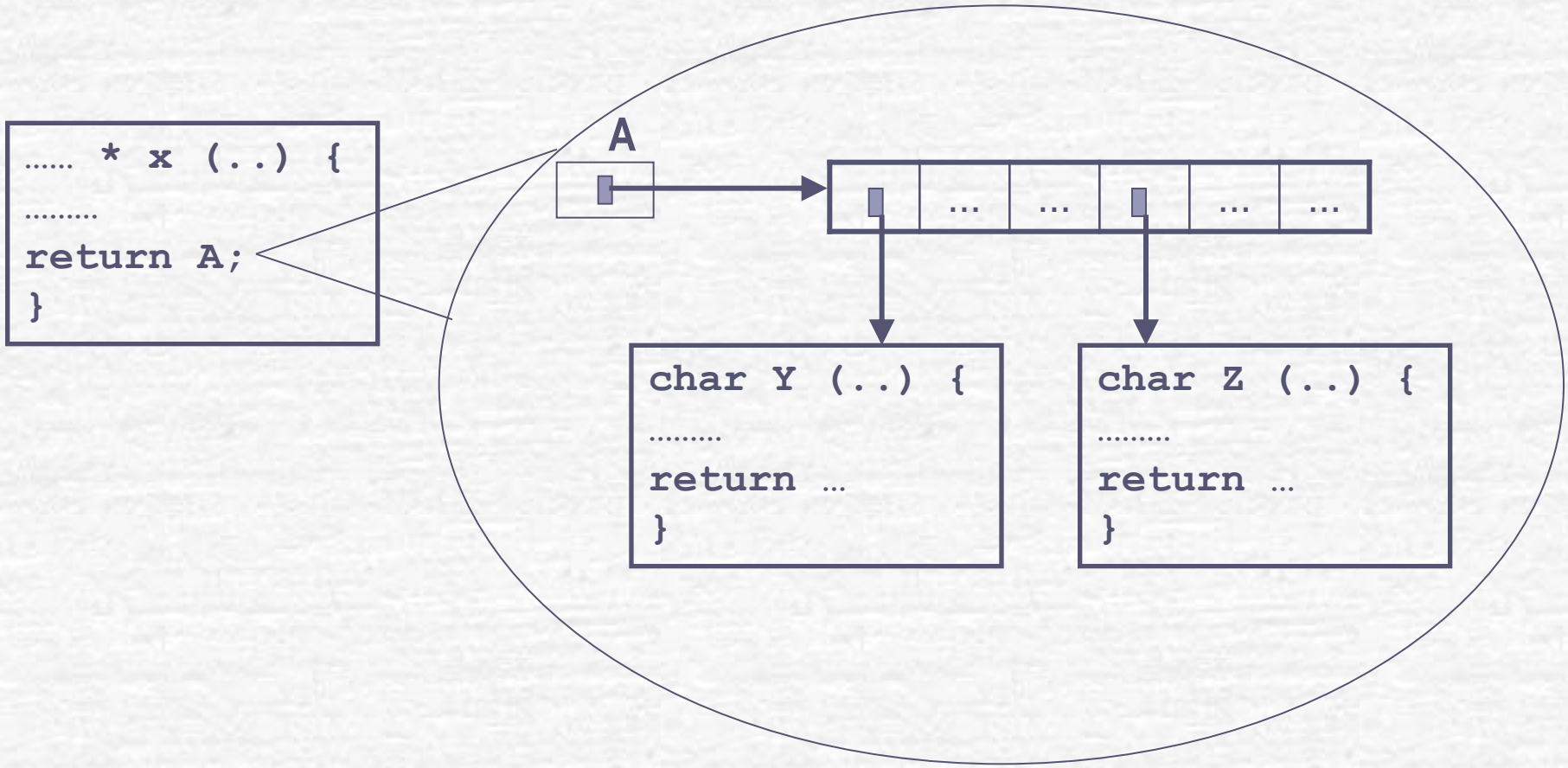
char *daytab[13]



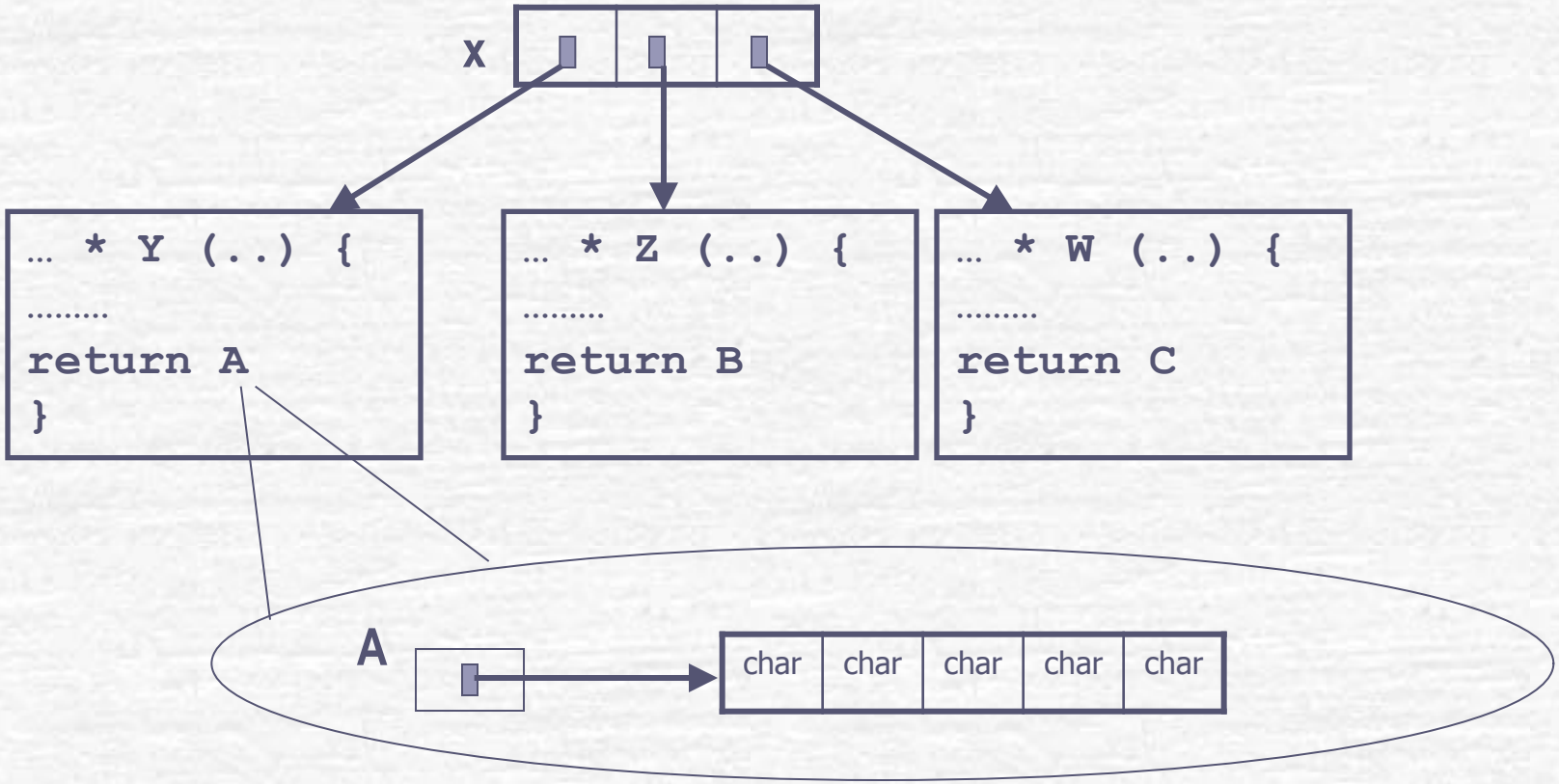
void (*comp)()



char (*(*x())[])()



char (*(*x[3])())[5]



```
/* linked list example */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

/* definition of a data node for holding student information */
struct node {
   char name[20];
   int  id;
   struct node *next;
};/* function prototypes */
 struct node * initnode( char *, int );
 void printnode( struct node * );
 void printlist( struct node * );
 void add( struct node * );
 struct node * searchname( struct node *, char *
       );
 void deletenode( struct node * );
 void insertnode( struct node * );
 void deletelist( struct node * );
```

```
/* head points to first node in list, end points to last node in list */
/* initialise both to NULL, meaning no nodes in list yet */
struct node *head = (struct node *) NULL;
struct node *end = (struct node *) NULL;

/* this initialises a node, allocates memory for the node, and returns   */
/* a pointer to the new node. Must pass it the node details, name and id */
struct node * initnode( char *name, int id )
{
   struct node *ptr;
   ptr = (struct node *) calloc( 1, sizeof(struct node ) );
   if( ptr == NULL )                    /* error allocating node?      */
      return (struct node *) NULL;      /* then return NULL, else      */
   else {                               /* allocated node successfully */
      strcpy( ptr->name, name );        /* fill in name details        */
      ptr->id = id;                     /* copy id details             */
      return ptr;                       /* return pointer to new node  */
   }
}

/* this prints the details of a node, eg, the name and id                */
/* must pass it the address of the node you want to print out            */
void printnode( struct node *ptr )
{
   printf("Name ->%s\n", ptr->name );
   printf("ID   ->%d\n", ptr->id );
}
```

```
/* this prints all nodes from the current address passed to it. If you    */
/* pass it 'head', then it prints out the entire list, by cycling through */
/* each node and calling 'printnode' to print each node found             */
void printlist( struct node *ptr )
{
   while( ptr != NULL )          /* continue whilst there are nodes left */
   {
      printnode( ptr );          /* print out the current node           */
      ptr = ptr->next;           /* goto the next node in the list       */
   }
}

/* this adds a node to the end of the list. You must allocate a node and  */
/* then pass its address to this function                                 */
void add( struct node *new )  /* adding to end of list */
{
   if( head == NULL )      /* if there are no nodes in list, then         */
      head = new;          /* set head to this new node                   */
   if (end!=NULL) end->next = new;        /* link in the new node to the end of the list */
   new->next = NULL;       /* set next field to signify the end of list   */
   end = new;              /* adjust end to point to the last node        */
}

/* search the list for a name, and return a pointer to the found node     */
/* accepts a name to search for, and a pointer from which to start. If    */
/* you pass the pointer as 'head', it searches from the start of the list */
struct node * searchname( struct node *ptr, char *name )
{
    while( strcmp( name, ptr->name ) != 0 ) {    /* whilst name not found */
       ptr = ptr->next;                          /* goto the next node    */
       if( ptr == NULL )                         /* stop if we are at the */
          break;                                 /* of the list           */
    }
    return ptr;                                  /* return a pointer to   */
}                                                /* found node or NULL    */
```

```
/* deletes the specified node pointed to by 'ptr' from the list          */
void deletenode( struct node *ptr )
{
   struct node *temp, *prev;
   temp = ptr;    /* node to be deleted */
   prev = head;   /* start of the list, will cycle to node before temp    */

   if( temp == prev ) {                  /* are we deleting first node  */
      head = head->next;                   /* moves head to next node     */
      if( end == temp )                   /* is it end, only one node?   */
         end = end->next;                  /* adjust end as well          */
      free( temp );                      /* free space occupied by node */
   }
   else {                               /* if not the first node, then */
      while( prev->next != temp ) {       /* move prev to the node before*/
         prev = prev->next;               /* the one to be deleted       */
      }
      prev->next = temp->next;             /* link previous node to next  */
      if( end == temp )                   /* if this was the end node,   */
         end = prev;                     /* then reset the end pointer  */
      free( temp );                      /* free space occupied by node */
   }
}
```

```
/* pass it the address of the new node to be inserted, with details all  filled in  */
void insertnode( struct node *new )
{
   struct node *temp, *prev;              /* similar to deletenode      */

   if( head == NULL ) {                   /* if an empty list,          */
      head = new;                         /* set 'head' to it           */
      end = new;
      head->next = NULL;                  /* set end of list to NULL    */
      return;                             /* and finish                 */
   }

   temp = head;                           /* start at beginning of list */
                  /* whilst currentname < newname to be inserted then */
   while( strcmp( temp->name, new->name) < 0 ) {
         temp = temp->next;               /* goto the next node in list */
         if( temp == NULL )               /* dont go past end of list   */
            break;
   }

   /* we are the point to insert, we need previous node before we insert  */
   /* first check to see if its inserting before the first node!          */
   if( temp == head ) {
      new->next = head;                   /* link next field to original list   */
      head = new;                         /* head adjusted to new node          */
   }
   else {     /* okay, so its not the first node, a different approach    */
      prev = head;   /* start of the list, will cycle to node before temp */
      while( prev->next != temp ) {
         prev = prev->next;
      }
      prev->next = new;                   /* insert node between prev and next  */
      new->next = temp;
      if( end == prev )                   /* if the new node is inserted at the */
        end = new;                        /* end of the list the adjust 'end'   */
   } }
```

```
/* this deletes all nodes from the place specified by ptr                */
/* if you pass it head, it will free up entire list                      */
void deletelist( struct node *ptr )
{
  struct node *temp;

  if( head == NULL ) return;   /* dont try to delete an empty list       */

  if( ptr == head ) {      /* if we are deleting the entire list         */
     head = NULL;          /* then reset head and end to signify empty   */
     end = NULL;           /* list                                       */
  }
  else {
     temp = head;          /* if its not the entire list, readjust end   */
     while( temp->next != ptr )        /* locate previous node to ptr    */
        temp = temp->next;
     end = temp;                       /* set end to node before ptr     */
  }

  while( ptr != NULL ) {   /* whilst there are still nodes to delete     */
    temp = ptr->next;      /* record address of next node                */
    free( ptr );           /* free this node                             */
    ptr = temp;            /* point to next node to be deleted           */
  }
}
```

```
/* this is the main routine where all the glue logic fits                */
main()
{
  char name[20];
  int id, ch = 1;
  struct node *ptr;

   system("clear");     //        clrscr();
  while( ch != 0 ) {
    printf("1 add a name \n");
    printf("2 delete a name \n");
    printf("3 list all names \n");
    printf("4 search for name \n");
    printf("5 insert a name \n");
    printf("0 quit\n");
    scanf("%d", &ch );
    switch( ch )
    {
       case 1:  /* add a name to end of list */
                printf("Enter in name -- ");
                scanf("%s", name );
                printf("Enter in id -- ");
                scanf("%d", &id );
                ptr = initnode( name, id );
                add( ptr );
                break;
       case 2:  /* delete a name */
                printf("Enter in name -- ");
                scanf("%s", name );
                ptr = searchname( head, name );
                if( ptr ==NULL ) {
                    printf("Name %s not found\n", name );
                }
                else
                   deletenode( ptr );
                break;
```

```
case 3:  /* list all nodes */
                printlist( head );
                break;

        case 4:  /* search and print name */
                printf("Enter in name -- ");
                scanf("%s", name );
                ptr = searchname( head, name );
                if( ptr ==NULL ) {
                    printf("Name %s not found\n", name );
                }
                else
                   printnode( ptr );
                break;
        case 5:  /* insert a name in list */
                printf("Enter in name -- ");
                scanf("%s", name );
                printf("Enter in id -- ");
                scanf("%d", &id );
                ptr = initnode( name, id );
                insertnode( ptr );
                break;
        default: break;

     }
   }
   deletelist( head );
}
```

Επεξεργασία Αρχείων στη C

- Με τη χρήση `FILE *` αναπαριστάνουμε δείκτη σε ένα αρχείο.
- `fopen` χρησιμοποιείται για να ανοίγει ένα αρχείο. Επιστρέφει την ειδική τιμή `NULL` για να δείξει ότι δεν μπορεί να ανοίξει ένα αρχείο.

```
FILE *fptr;
char filename[]= "file2.dat";
fptr= fopen (filename,"w");
if (fptr == NULL) {
    fprintf (stderr, "ERROR");
     /* DO SOMETHING */
}
```

Τρόποι Ανοίγματος Αρχείου

- FILE * fopen(const char * filename, const char * mode)
- Το δεύτερο όρισμα της `fopen` καθορίζει τον *τρόπο* με τον οποίο ανοίγουμε ένα αρχείο.

  Υπάρχουν οι εξής επιλογές:
- "r" ανοίγει ένα αρχείο για διάβασμα
- "w" ανοίγει ένα αρχείο για εγγραφή
- "a" ανοίγει ένα αρχείο για προσθήκη / εγγραφή
- “b” δυαδικά δεδομένα (binary)
- rb, wb, ab, r+, w+, a+, rb+,wb+,ab+

Διαχείριση Αρχείων

- Κλείσιμο αρχείων
  int *fclose* (FILE * fp);
- Είσοδος – έξοδος χαρακτήρων
  int *putc*(int c, FILE * fp);
  (ισοδύναμα *fputc*())
  int *getc*(FILE * fp);
  (ισοδύναμα *fgetc*())
- Είσοδος – έξοδος συμβολοσειρών
  int *fputs* (const char * s, FILE * fp);
  char * *fgets* (char * s, int n, FILE * fp);

- int *feof(FILE* *fp)

  Not 0: end of file
- void *rewind* (FILE *fp);
- int *fflush(FILE* *fp)

  0: success, EOF: error
- int *fseek*(FILE *fp, long int apostasi, int thesi)

  thesi=SEEK_SET (0), SEEK_CUR (1), SEEK_END (2)

  int *ftell*(FILE *fp)
- int fread(void *buffer, int arbytes, int fores, FILE *fp)

  Διάβασε fores στοιχεία, μεγέθους arbytes το καθένα, στο buffer array
- int fwrite(void *buffer, int arbytes, int fores, FILE *fp)

  Γράψε fores στοιχεία, μεγέθους arbytes το καθένα, από το buffer array

Παράδειγμα (1)

```
#define getchar()  getc(stdin)
#define putchar(c) putc((c), stdout)

char *fgets(char *s, int n, FILE *iop)
 { register int c; register char *cs; cs=s;
    while (--n>0 && (c=getc(iop))!=EOF)
                if ((*cs++=c)==‘\n’) break;
    *cs=‘\0’; return (c==EOF && cs==s)?NULL:s;
}


int fputs(char *s, FILE *iop)
{ int c;
  while (c=*s++) putc(c,iop);
  return ferror(iop)?EOF:0;
}
```

Παράδειγμα (2)

```
#include <stdio.h>
void filecopy(FILE *, FILE *);

main (int argc, char *argv[])
{    FILE *fp;
     if (argc==1) filecopy(stdin, stdout);
     else
          while (--argc>0)
          if ((fp=fopen(*++argv, "r"))==NULL) {
        printf("cat: δεν μπορώ να ανοίξω το %s\n", argv)
          return 1;
                                                          }
          else {filecopy(fp, stdout); fclose(fp);}
        return 0;
}

void filecopy(FILE *ifp, FILE *ofp)
{    int c;
     while ((c=getc(ifp))!=EOF)
        putc(c,ofp);
}
```

Γράφοντας σε Αρχείο με *fprintf*

- Η *fprintf* δουλεύει όπως η *printf* και η *sprintf* με την διαφορά ότι το πρώτο όρισμα είναι ένας δείκτης σε αρχείο.

```
FILE *fptr;
fptr= fopen ("file.dat","w");
/* Check it's open */
fprintf (fptr,"Hello World!\n");
```

- Θα μπορούσαμε επίσης να διαβάσουμε αριθμούς από ένα αρχείο χρησιμοποιώντας *fscanf* – αλλά υπάρχει και καλύτερος τρόπος.

Διαβάζοντας από Αρχείο με *fgets*

- *fgets* είναι ένας καλύτερος τρόπος ανάγνωσης από αρχείο
- Μπορούμε να διαβάσουμε από αρχείο με *fgets*

```
FILE *fptr;
char line [1000];
/* Open file and check it is open */
while (fgets(line,1000,fptr) != NULL) {
   printf ("Read line %s",line);
```

fgets έχει 3 ορίσματα, μία συμβολοσειρά, ένα μέγιστο αριθμό χαρακτήρων προς ανάγνωση και ένα δείκτη αρχείου. It returns NULL if there is an error (such as EOF)

fgets(char*s, int n, FILE *fstream); (διαβάζει το πολύ n-1 χαρακτήρες σταματώντας αν συναντήσει χαρακτήρα νέας γραμμής)

Κλείνοντας ένα Αρχείο

- Μπορούμε να κλείσουμε ένα αρχείο απλώς χρησιμοποιώντας *fclose* και το δείκτη αρχείου. Ακολουθεί παράδειγμα "hello files".

```
FILE *fptr;
char filename[]= "myfile.dat";
fptr= fopen (filename,"w");
if (fptr == NULL) {
    printf ("Cannot open file to write!\n");
    exit(-1);
}
fprintf (fptr,"Hello World of filing!\n");
fclose (fptr);
```

Παράδειγμα (3)

Να υλοποιηθούν και να χρησιμοποιηθούν οι ακόλουθες συναρτήσεις:

1. int **createRandFile**(char *fileName,int N): Δημιουργεί ένα αρχείο με όνομα fileName στο οποίο γράφει N τυχαίους αριθμούς. Επιστρέφει 1 σε επιτυχία και -1 σε αποτυχία
2. int ***loadData**(char *fileName,int N): Διαβάζει από το αρχείο fileName N τυχαίους αριθμούς και τους επιστρέφει σε ένα δείκτη ακεραίων.
3. int **writeArray2File**(char *fileName,int *array,int N): Γράφει τους αριθμούς του διανύσματος array στο αρχείο fileName. Επιστρέφει 1 σε επιτυχία και -1 σε αποτυχία.

```
int createRandFile(char *fileName,int N)
    {
        FILE *fp;
        int i;

        fp = fopen(fileName,"w");
        if (fp)
        {
            srand((unsigned)time(NULL));
            for (i=0;i<N;i++)
            {
                fprintf(fp,"%d\n",(MAX*rand())/RAND_MAX);
            }
            fclose(fp);
        return 1;
        }
        else
            return -1;
    }
```

```
int *loadData(char *fileName,int N)
    {
        int *array = (int*)malloc(sizeof(int)*N);
        FILE *fp;
        int i;

        fp = fopen(fileName,"r");
        if (fp)
        {
            for (i=0;i<N;i++)
            {
                fscanf(fp,"%d",&array[i]);
            }
            fclose(fp);
            return array;
        }
        else
            return NULL;
    }
```

```
int writeArray2File(char *fileName,int *array,int N)
    {
        FILE *fp;
        int i;

        fp = fopen(fileName,"w");
        if (fp)
        {
            for (i=0;i<N;i++)
            {
                fprintf(fp,"%d\n",array[i]);
            }
            fclose(fp);
            return 1;
        }
        else
        return -1;
    }
```