Implementazione di una lista circolare.

Esercizio L_10 - Circolare


```
/* visualizza lista dinamica (CIRCOLARE)                                */
void Visualizza(TipoLista lista)
{
        TipoLista testa;

        testa=lista;

        /* verifica lista vuota */
        if (lista==NULL)
                printf("\nLista vuota");
        else
        {
                printf("\nLista: ");

                /*      stampa le singole informazioni                  */
                while (lista->next != testa)
                {
                        printf(" %d ->", lista->info);
                        lista=lista->next;      /* alla prossima cella */
                }

                printf(" %d ...", lista->info);
        }
}
```

Esercizio L_10 - Circolare


```
/* inserimento in coda alla lista dinamica  (CIRCOLARE) */
TipoLista InserisciCoda(TipoLista nuovo, TipoLista lista)
{
      TipoLista temp;
      /* memorizzo la testa della lista dinamica                          */
      temp=lista;
      if (temp==NULL)
      {
            /* collego il nuovo elemento (ultimo) a se stesso             */
            nuovo->next=nuovo;
            return (nuovo); /*      modifica la testa della lista        */
      }
      else
      {     /*      scorre la lista     */
            while (temp->next != lista)
                  temp=temp->next;        /* alla prossima cella          */

            temp->next=nuovo; /* accodo il nuovo elemento dinamico          */

            /* collego il nuovo elemento (ultimo) alla testa              */
            nuovo->next=lista;

            return(lista); /* restituisco la testa della lista (inalterata)*/
      }
}
```

Gestione degli studenti di una classe tramite una lista.

Esercizio L_11 - Studenti


```c
#include <stdio.h>

typedef char stringa [20];
typedef char nomefile [12];

typedef enum {false, true}  boolean;

typedef struct { stringa nome;
                 stringa cognome;
                 int voto;
                 int eta; } tpersona;

typedef struct elemento_lista  {
                   tpersona info;
                   struct elemento_lista * next;
                   }   elem_lista ;

// la classe è implementata come lista dinamica
typedef elem_lista * listaClasse;
```

Esercizio L_11 - Studenti


```
void leggiStudente (tpersona * studletto);
void inserisciInClasse (listaClasse * classe, tpersona studente);
void visualizzaClasse (listaClasse classe);
void salvaClasse (listaClasse classe, nomefile nomef);
void caricaClasse (listaClasse * classe, nomefile nomef);

// restituisce NULL se non trovato
elem_lista* cercaStudente (listaClasse classe, stringa cognomeCercato);
/* NB: punt e prec sono puntatori passati per indirizzo, ovvero doppi
       puntatori!
       boolean cercaStudenteConPrec (listaClasse classe, stringa
                 cognomeCercato, elem_lista **punt, elem_lista **prec);
*/

boolean eliminaStudente (listaClasse *classe, stringa cognomeCercato);

boolean isVuota (listaClasse classe);
void Svuota (listaClasse * classe);
```

Esercizio L_11 - Studenti


```
void main()
 {

   nomefile unFile;
   listaClasse unaClasse;
   tpersona unoStudente;
   stringa cognomeLetto;
   boolean esitoElimina;

   // tipo di ritorno della ricerca senza precedente
   elem_lista * esitoRicerca;

   //boolean esitoAltraRicerca;


   char carLetto;

   unaClasse = NULL;  // inizializzazione di ogni lista;
```

Esercizio L_11 - Studenti


```
do
{
    printf ("*** MENU PRINCIPALE ***\n\n");
    printf ("scegli la tua opzione:\n");
    printf ("1: inserisci uno studente\n");
    printf ("2: visualizza la classe\n");
    printf ("3: cerca uno studente per cognome\n");
    printf ("4: elimina uno studente\n");
    printf ("5: svuota la classe\n");
    printf ("6: test - la classe e' vuota?\n");
    printf ("7: salva la classe su disco\n");
    printf ("8: carica la classe da disco\n");
    printf ("9: uscita \n\n");

    // filtro l'input
    do {
       scanf ("%c",&carLetto);
    }while (carLetto < '1' || carLetto > '9');
```

Esercizio L_11 - Studenti

```
switch (carLetto)
{
   case '1': leggiStudente (&unoStudente);
             inserisciInClasse (&unaClasse, unoStudente);
             break;

   case '2':
        visualizzaClasse (unaClasse);
        break;


   case '3':  // ricerca per cognome
        printf ("inserisci il cognome dello studente da cercare: ");
        scanf ("%s", cognomeLetto);
        esitoRicerca = cercaStudente (unaClasse, cognomeLetto);
        if (esitoRicerca == NULL)
                printf ("\nStudente non trovato'");
        else printf ("\ntrovato!");

        break;
```

Esercizio L_11 - Studenti


```
case '4':  // eliminazione di un elemento (usa la ricerca)
      printf ("inserisci il cognome dello studente da eliminare: ");
      scanf ("%s", cognomeLetto);
      esitoElimina = eliminaStudente (&unaClasse, cognomeLetto);
      if (esitoElimina == false)
            printf ("\nCognome %s non presente", cognomeLetto);
      else
            printf ("\ntrovato e cancellato!");
      break;

 case '5':  // svuotamento
            Svuota (&unaClasse);
            break;

 case '6':
            if (isVuota (unaClasse))
                  printf ("\nLa classe e' vuota");
            else
                  printf ("\nLa classe non e' vuota");

            break;
```

Esercizio L_11 - Studenti


```
case '7':  // salva su disco
            printf ("inserisci il nome del file in cui
                                                scrivere: ");
            scanf ("%s", unFile);
            salvaClasse (unaClasse, unFile);
            break;

case '8':  // carica da disco
            printf ("inserisci il nome del file da caricare: ");
            scanf ("%s", unFile);
            caricaClasse (&unaClasse, unFile);

            break;

case '9': break;
```

Esercizio L_11 - Studenti

```
            // impossibile raggiungere questo caso se il filtro
                                                sull'input funziona
            default: printf ("errore, qualcosa non va");
            break;

      } // fine switch

   } while (carLetto != '9');

   printf ("\n\nFine programma\n");

}
```

Esercizio L_11 - Studenti


```
void leggiStudente (tpersona * studletto)
{


        printf ("\ninserisci nome: ");
        scanf ("%s", studletto->nome);

        printf ("\ninserisci cognome: ");
        scanf ("%s", studletto->cognome);

        printf ("\ninserisci voto: ");
        scanf ("%d", &(studletto->voto));

        printf ("\ninserisci eta': ");
        scanf ("%d", &(studletto->eta));



}
```

Esercizio extra: Inserire i controlli!

Esercizio L_11 - Studenti


```
void inserisciInClasse (listaClasse * classe, tpersona studente)
{


      elem_lista * temp;
      temp = (elem_lista *) malloc (sizeof (elem_lista));

      if (temp == NULL)
      {
         printf ("\nerrore di allocazione della memoria!");
      }
      else
      {
        temp->info=studente;
        temp->next = *classe;
        *classe = temp;
      }
```

Esercizio extra: Prima di allocare memoria, verificare che il cognome non esista già e chiedere conferma all'utente

```
}
```

Esercizio L_11 - Studenti


```
void visualizzaClasse (listaClasse classe)
{
        elem_lista * temp;
        temp = classe;

        if (temp == NULL)
            printf ("\nla classe e' vuota!");

        else
            printf (" *** VISUALIZZO LA CLASSE *** \n\n");

        while (temp != NULL)
        {
                printf ("%s %s di anni %d ha preso %d\n",
                                temp->info.nome,
                                temp->info.cognome,
                                temp->info.eta,
                                temp->info.voto);
                temp = temp->next;
        }
}
```

Esercizio L_11 - Studenti

```
void salvaClasse (listaClasse classe, nomefile nomef)
{
     FILE *pfile;
     elem_lista * temp;
     int contatore = 0;
     temp = classe;

     pfile = fopen (nomef, "wb");
     // apparentemente non è necessario aprire binario

     while (temp != NULL)
     {
          fwrite (&temp->info, sizeof (tpersona), 1, pfile);
          temp = temp->next;
          contatore++;
     }

     printf ("\nho scritto sul disco %d persone", contatore);
     fclose (pfile);

}
```

Esercizio L_11 - Studenti


```
void caricaClasse (listaClasse * classe, nomefile nomef)
{

      FILE *pfile;
      tpersona temp;
      int contatore = 0;

      pfile = fopen (nomef, "rb");
      // attenzione, basilare aprire in lettura binaria

      while( fread (&temp, sizeof (tpersona), 1, pfile) != 0 )
      {
            inserisciInClasse (classe, temp);
            contatore ++;
      }

      printf ("\nho letto dal disco %d persone", contatore);
      fclose (pfile);


}
```

Esercizio L_11 - Studenti


```
elem_lista* cercaStudente (listaClasse classe, stringa cognomeCercato)
{



      elem_lista * temp;
      temp = classe;
      while (temp != NULL)
      {
          if (strcmp (temp->info.cognome, cognomeCercato) == 0)
                    return temp;
          temp = temp->next;
      }


      return NULL; // non ho trovato l'elemento




}
```

Esercizio L_11 - Studenti


```
boolean cercaStudenteConPrec (listaClasse classe,
          stringa cognomeCercato, elem_lista **punt, elem_lista**prec)
{

      *punt = classe;
      *prec = NULL;

      while (*punt != NULL)
      {
          if (strcmp ( (*punt)->info.cognome, cognomeCercato) == 0)
                    return true;   // trovato! punt e prec sono già OK

          *prec = *punt;
          // memorizzo la posizione corrente (che diventa precedente)

          *punt = (*punt)->next;  // avanzo il cursore di uno
      }

      return false;  // non l'ho trovato

}
```

Esercizio L_11 - Studenti


```
boolean isVuota (listaClasse classe)
{

        return (classe == NULL);

}
```

Esercizio L_11 - Studenti


```
void Svuota (listaClasse * classe)
{
        /* NOTA BENE: scrivere semplicemente
                    classe = NULL
           e' un errore perchè genera un sacco di garbage!
        */

        elem_lista * temp;
        int contatore = 0;

        while (*classe != NULL)
        {
            temp = *classe;
            *classe = (*classe)->next;
            free (temp);
            contatore++;
        }

        printf ("\n ho svuotato la lista deallocando %d elementi",
                                                    contatore);
}
```

Esercizio L_11 - Studenti


```
boolean eliminaStudente (listaClasse *classe, stringa cognomeCercato) {
     elem_lista * temp;
     elem_lista * prec;
     temp = *classe;
     // i casi sono due: devo eliminare in testa oppure no
     if (strcmp (temp->info.cognome, cognomeCercato) == 0)  {
          *classe = (*classe)->next;  // elimino in testa, facile
           free (temp);
           return true;
     }
     else   //non devo eliminare in testa, bensì nel mezzo.
         while (temp != NULL) {
            prec = temp;
            temp = temp->next;
            if (strcmp (temp->info.cognome, cognomeCercato) == 0)  {
                prec->next = temp->next;
                free (temp);
                return true;
            }
         }
     return false;  // sono arrivato in fondo e non ho trovato l'elemento
}
```

Ricerca in una lista principale di una sequenza di valori contenuti in una lista secondaria.

Esercizio L_12 - TdE 04/02/05 (modificato)


```
struct elem {
        int valore;
        struct elem * next;
};
int CercaSubList(struct elem * princ, struct elem * sub);

int RiempiListaTesta (struct elem ** L) {

        struct elem * temp;
        int N, i;
        do {
                printf("Inserisci il numero di valori costituenti la lista:");
                scanf("%d",&N);
        } while (N<=0);

        for (i=0; i<N; i++) {
                temp = (struct elem *) malloc(sizeof(struct elem));
                if (temp==NULL)
                        return 0;
                scanf("%d", &temp->valore);
                temp->next = (*L);
                (*L) = temp;
        }
        return 1;
}
```

Esercizio L_12 - TdE 04/02/05 (modificato)


```
void main() {
        struct elem * lista=NULL;
        struct elem * sublist=NULL;

        int pos;

        printf("Inserisci la lista principale:\n");
        if ( RiempiListaTesta(&lista) == 0) {
                printf("Errore nella generazione della lista\n");
                exit(0);
        }

        printf("Inserisci la sottolista da cercare:\n");
        if ( RiempiListaTesta(&sublist) == 0) {
                printf("Errore nella generazione della sotto lista\n");
                exit(0);
        }

        pos = CercaSubList(lista, sublist);
        if (pos >= 0)
                printf("Sottolista trovata a partire dalla posizione %d\n",pos);
        else
                printf("Sottolista non trovata\n");

}
```

Esercizio L_12 - TdE 04/02/05 (modificato)


```
int CercaSubList(struct elem * princ, struct elem * sub) {

        int pos = 1;
        struct elem *subtemp, *princtemp;

        while (princ !=NULL) {
                if (princ->valore == sub->valore) {
                        princtemp = princ;
                        subtemp = sub;

                        while ( (princtemp !=NULL)
                                        && (subtemp!=NULL)
                                        && (princtemp->valore == subtemp->valore))
                        {
                                princtemp = princtemp->next;
                                subtemp = subtemp->next;
                        }
                        if (subtemp==NULL)
                                return pos;
                }
                princ = princ->next;
                pos++;
        }
        return -1;
}
```

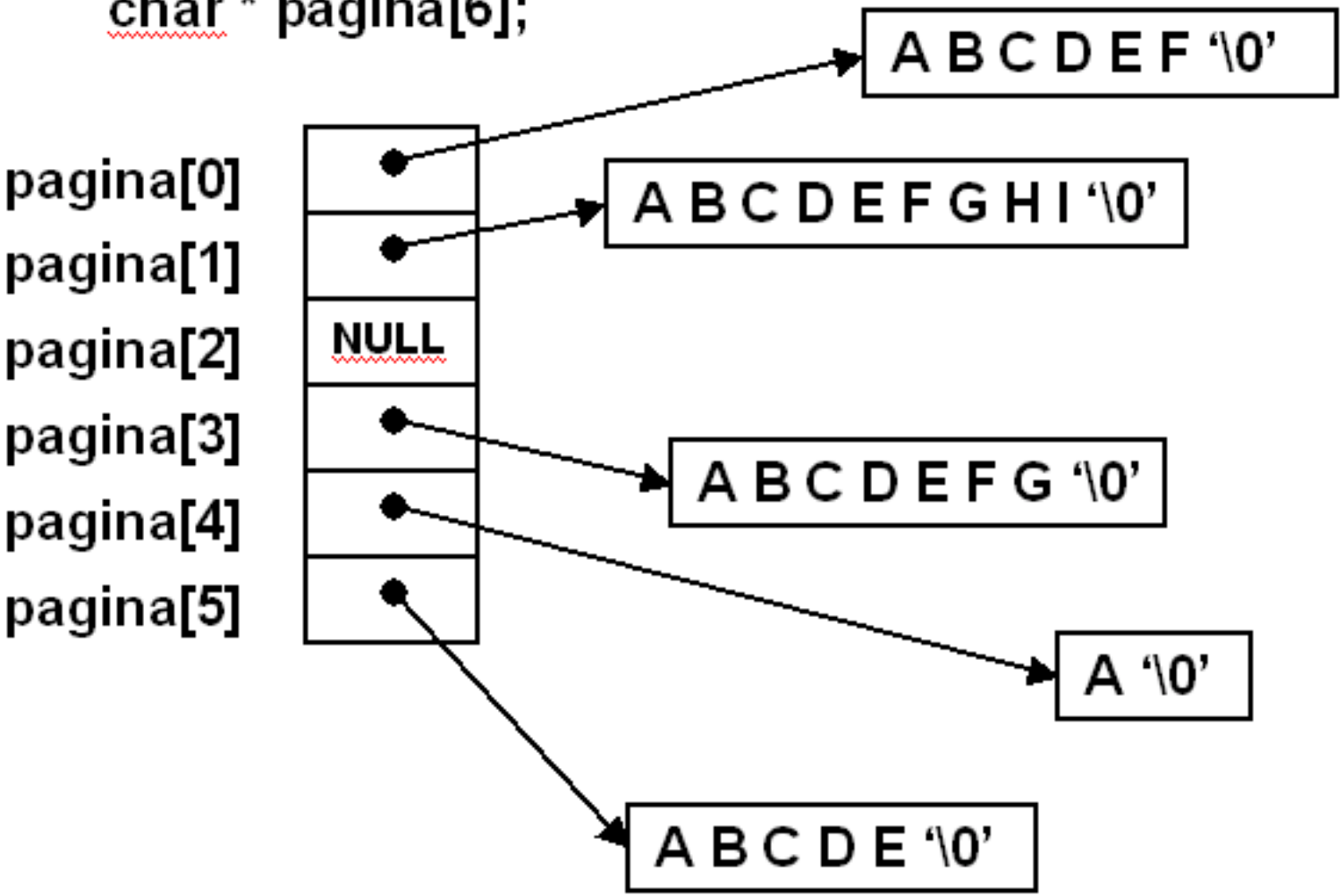
Implementazione di liste di liste.

Allocazione dinamica di stringhe.

Esercizio L_13 - Liste di Liste

Struttura Dati:

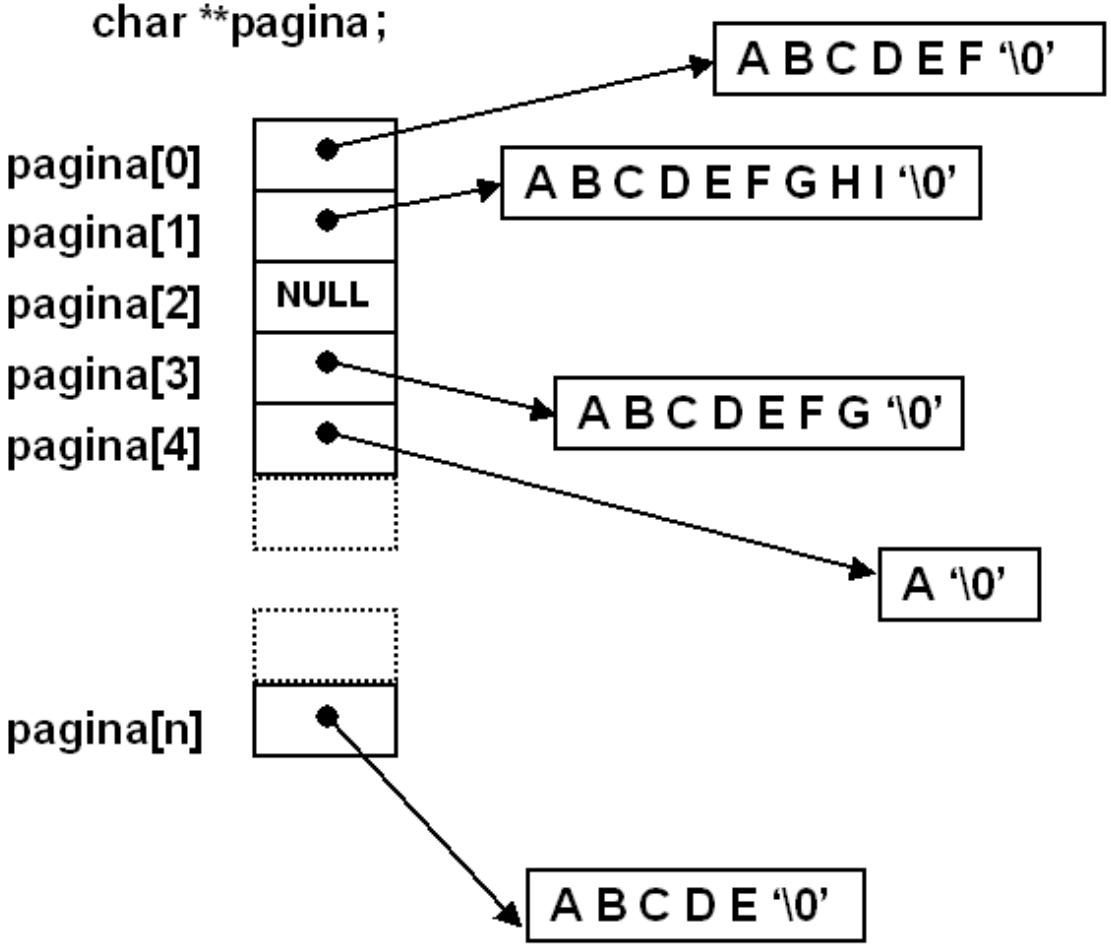
char * pagina[6];



Esercizio L_13 - Liste di Liste

Struttura Dati:

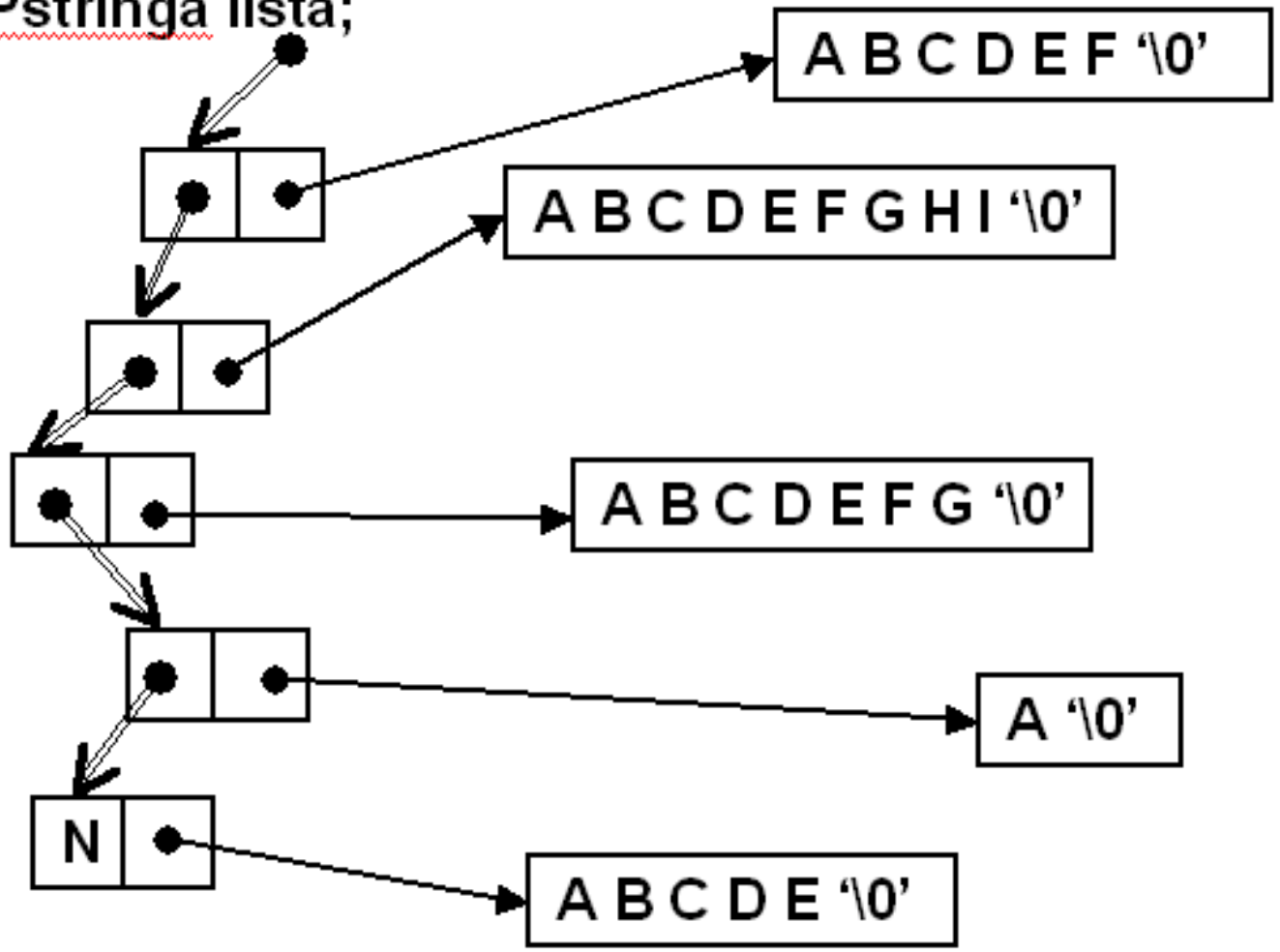
char **pagina;



Esercizio L_13 - Liste di Liste

Struttura Dati:

Pstringa lista;



Esercizio L_13 - Liste di Liste


```c
#define MAXC 81

/* matrice di caratteri non sovradimensionata e dinamica                */

struct elem {
        /* puntatore alla stringa                                       */
        char *stringa;

        /* puntatore al prossimo elemento                               */
        struct elem *prox;
};

/* definiz. del tipo puntatore all'elemento della lista                 */
typedef struct elem *PStringa;
```

Esercizio L_13 - Liste di Liste


```
void main() {


    PStringa lista=NULL;            /* lista delle stringhe ins.     */
    PStringa nuovo, temp;
    char riga[MAXC];                /* stringa temporanea            */

    /* memorizzazione dell'input                                      */
    printf("\nInserisci stringa:");
    gets(riga);

    while (strcmp(riga,"")!=0) {

            /* memorizzazione stringa */

            /* prossima stringa                                       */
            printf("\nInserisci stringa:");
            gets(riga);

    }

    /* visualizzazione delle stringhe inserite e rilascio memoria    */


}
```

A

B

Esercizio L_13 - Liste di Liste


```
/* memorizzazione stringa */

/* allocazione della memoria per il puntatore alla stringa            */
 nuovo = (PStringa) malloc (sizeof(struct elem));
 if (nuovo==NULL)
         printf("\nErrore in allocazione!");
 else
 {
     /* inizializzazione elemento                                      */
     nuovo->prox=NULL;
     nuovo->stringa = mem_str(riga);

     /* aggiungo il nuovo elemento dinamico                            */
     nuovo->prox=lista;

     /* aggiorno testa della lista                                     */
     lista=nuovo;



 }
```

Esercizio L_13 - Liste di Liste

B

```
/* visualizzazione delle stringhe inserite e rilascio memoria*/
while (  lista!=NULL)
{

                    elim_str(&(lista->stringa));

                    temp=lista->prox;

                    free(lista);

                    lista=temp;









}
```

Esercizio L_13 - Liste di Liste


```
/* alloca la memoria e memorizza la stringa passata come parametro */
char *mem_str(char str[])
{

        char *punt;


        /* allocazione della memoria */
        punt=(char *) malloc( (strlen(str)+1) * sizeof(char) );


        if (punt==NULL)
                printf("\nErrore in allocazione!");

        else
                /* copia della stringa inserita */
                strcpy(punt,str);


        /* indirizzo di memoria restituito */
        return (punt);



}
```

Esercizio L_13 - Liste di Liste


```
/* visualizzazione e rilascio delle stringhe inserite */
void elim_str(char *str[])
{


        /* visualizzazione */

        printf("\n%s",*str);



        /* rilascio della memoria */

        free(*str);

        *str=NULL;









}
```

Esempio di implementazione di liste di liste: il ricettario.

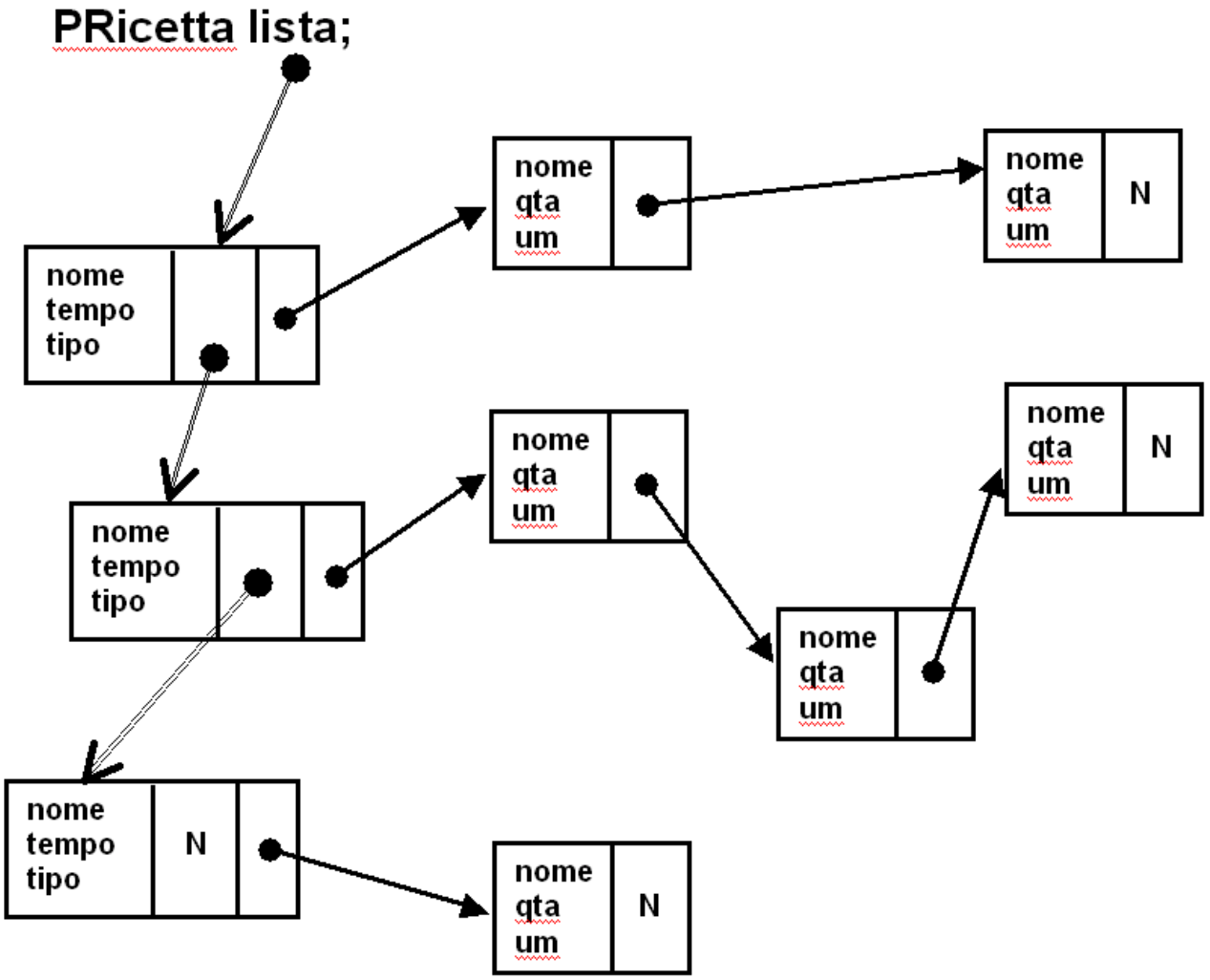
Ogni ricetta è caratterizzata da un nome, un tempo di cottura, un tipo e da un certo numero di ingredienti (MAX 5).

Ciascun ingrediente utilizzato in una certa ricetta è caratterizzato da un nome, da una quantità usata e da un'unità di misura.

Definire le strutture dati necessarie per risolvere il problema.

Esercizio L_14 - Ricettario

Struttura Dati:



Esercizio L_14 - Ricettario


```
#include <stdio.h>
#define MAX_RIC 5
#define MAX_ING 5
/* definizione del tipo di ricetta */
typedef enum {PRIMO, SECONDO, CONTORNO, DOLCE} classe;

/* definizione del tipo ingrediente */
struct s_ingr{
      char nome[20];                        /* ingrediente  */
      int qta;                              /* quantità  */
      char um[20];                          /* unità di misura  */
      struct s_ingr *next;                  /* puntatore al prox ingrediente */
};
/* definizione del tipo puntatore a ingrediente */
typedef struct s_ingr *PIngrediente;

/* definizione del tipo ricetta */
struct s_ric{
      char nome[20];            /* nome ricetta  */
      int     tempo;            /* tempo prepar  */
      classe tipo;              /* tipo di ricetta  */
      PIngrediente ingr;        /* ingredienti  */
      struct s_ric *next;       /* puntatore alla prox ricetta  */
};
typedef struct s_ric *PRicetta;   /* definizione del tipo puntatore a ricetta */
```

Esercizio L_14 - Ricettario


```
void main()
{  PRicetta ric;                          /* puntatore alle ricette */
   char            opz;                   /*  opzione scelta */
   char            cls;                   /* classe di ricette desiderata */
   do {
       printf ("\nScegli: (I)nserisci ricetta, (V)isualizza per classe, (F)ine: ");
       fflush(stdin);
       scanf("%c",&opz);
       switch (opz)
       {      case 'I':                     /* inserimento          */
                     if (ins_ric(&ric)!=0)
                            printf("\nImpossibile inserire la ricetta!");
                     break;
              case 'V':
                     vis_ric(ric);
                     break;
              case 'F':
                     rilascia(ric);
                     printf("\nArrivederci!");
                     break;
              default :
                     printf("\nComando non riconosciuto.");
       }
   } while (opz!='F');
}
```

Esercizio L_14 - Ricettario


```
/* funzione di inserimento di una ricetta  */
int ins_ric(PRicetta *lista_ric)
{
        PRicetta nuovo;
        char    cls;    /* classe di ricetta specificata */

        /* allocazione della memoria per una nuova ricetta */

        nuovo=(PRicetta) malloc(sizeof(struct s_ric));
        if (nuovo==NULL)
                return -1;
        /* richiesta informazioni */
        printf("\nIns. nome ricetta: ");
        fflush(stdin);
        scanf("%s", nuovo->nome);

        printf("\nIns. tempo di preparazione: ");
        fflush(stdin);
        scanf("%d", &nuovo->tempo);
        fflush(stdin);

        printf("\nIns. classe (P)rimo, (S)econdo, (C)ontorno, (D)olce: ");
        scanf("%c", &cls);
```

Esercizio L_14 - Ricettario


```
    switch (cls)
    {
        case 'P':
                    nuovo->tipo=PRIMO;
                    break;
        case 'S':
                    nuovo->tipo=SECONDO;
                    break;
        case 'C':
                    nuovo->tipo=CONTORNO;
                    break;
        case 'D':
                    nuovo->tipo=DOLCE;
                    break;
        default:
                    return (-1);
    }

    /*inserimento ingredienti */
    nuovo->ingr=NULL;
    while (ins_ing(&nuovo->ingr)==0);

    nuovo->next=*lista_ric; /* aggiungo il nuovo elemento in testa alla lista */
    *lista_ric=nuovo; /* aggiorno testa della lista */
    return (0); /* uscita dalla funzione senza errori */
}
```

Esercizio L_14 - Ricettario


```
/* funzione di inserimento di un ingrediente  */
int ins_ing(PIngrediente *lista_ingr)
{
        char risp;        /* risposta dell'utente  */
        PIngrediente nuovo;

        /* verifica richiesta di un nuovo ingrediente           */
        printf("\nVuoi inserire un nuovo ingrediente (S/N) ?");
        fflush(stdin);
        scanf("%c",&risp);
        if (risp=='N')
                return (-1);    /* uscita definitiva  */

        /* allocazione della memoria  */

        nuovo=(PIngrediente) malloc(sizeof(struct s_ingr));
        if (nuovo==NULL)
             return -1;

        /* richiesta informazioni  */
        printf("\nIns. nome ingrediente: ");
        fflush(stdin);
        scanf("%s", nuovo->nome);
```

Esercizio L_14 - Ricettario


```
        printf("\nIns. quantita': ");
        fflush(stdin);
        scanf("%d", &nuovo->qta);

        printf("\nIns. unita' di misura: ");
        fflush(stdin);
        scanf("%s", nuovo->um);

        /* aggiungo il nuovo elemento in testa alla lista  */
        nuovo->next=*lista_ingr;

        /* aggiorno testa della lista  */
        *lista_ingr=nuovo;









        /* uscita con ripetizione dell'inserimento  */
        return (0);
}
```

Esercizio L_14 - Ricettario


```
/* procedura di visualizzazione per classe di ricette */
void vis_ric(PRicetta lista_ric)
{
   while (  lista_ric!=NULL)                /* scansione della lista delle ricette*/
    {
            printf("\n");
            printf("\nNome: %s", lista_ric->nome);
            printf("\nTempo di prep.: %d", lista_ric->tempo);

            switch (lista_ric->tipo)
            {
                    case PRIMO:     printf("\nTipo: Primo");
                                    break;
                    case SECONDO:   printf("\nTipo: Secondo");
                                    break;
                    case CONTORNO:  printf("\nTipo: Contorno");
                                    break;
                    case DOLCE:     printf("\nTipo: Dolce");
                                    break;
            }

            vis_ingr(lista_ric->ingr);
            lista_ric=lista_ric->next; /* elemento successivo */
    }
}
```

Esercizio L_14 - Ricettario


```
/* procedura di visualizzazione degli ingredienti */
void vis_ingr(PIngrediente lista_ingr)
{
        /* scansione della lista degli ingredienti                    */
        while ( lista_ingr!=NULL)
        {


                /* visualizzazione singolo ingrediente  */
                printf("\n\tIns. nome ingrediente: %s", lista_ingr->nome);

                printf("\n\tIns. quantita': %d", lista_ingr->qta);

                printf("\n\tIns. unita' di misura: %s", lista_ingr->um);


                /* elemento successivo  */
                lista_ingr=lista_ingr->next;




        }
}
```

Esercizio L_14 - Ricettario


```
void rilascia(PRicetta lista_ric) /* rilascio della memoria allocata */
{
   PRicetta temp_ric;
   PIngrediente temp_ingr;
   /* scansione della lista delle ricette */
   while (  lista_ric!=NULL)
   {
      /* scansione della lista degli ingredienti */
      while (  lista_ric->ingr!=NULL)
      {
            /* mantengo puntatore all'ingrediente successivo        */
            temp_ingr=(lista_ric->ingr);
            temp_ingr=temp_ingr->next;

            free(lista_ric->ingr); /* rilascio memoria */

            lista_ric->ingr=temp_ingr; /* rispristino la lista degli ingredienti */

       }

      temp_ric=lista_ric->next; /* mantengo puntatore alla ricetta successiva */
      free(lista_ric); /* rilascio la memoria */
      lista_ric=temp_ric; /* rispristino la lista delle ricette */
   }
}
```

Il programma è costituito da un menu in cui si possono inserire le quantità di alcuni prodotti in un magazzino.

Il magazzino è una lista dinamica e l'inserimento è ordinato alfabeticamente in base al nome degli articoli.

E' possibile inserire i dati in due diversi magazzini ed è anche possibile fondere le due liste dinamiche in una terza lista sempre ordinata (ma in questo caso le due liste originali vengono distrutte).

Se lo stesso articolo é presente in entrambe le liste, la quantità della lista risultante sarà correttamente determinata dalla somma delle due quantità.

Esercizio L_15 - Magazzino


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


typedef char stringa[20];

typedef struct { stringa merce;
                 int quantita;
               } tprodotto;

typedef struct elem { tprodotto prodotto;
                      struct elem *link;
                    } telemento;

typedef telemento * tlista;
```

Esercizio L_15 - Magazzino


```
void acquisisci (tprodotto *new);

void inserisci (tlista *lista,tprodotto nuovo);

void insord (tlista *lista, tprodotto nuovoprod);

void stampa (tlista lista);

void estrai (telemento **punt, tlista *l);

tlista fondi (tlista l1, tlista l2);
```

Esercizio L_15 - Magazzino


```
int main () {
    tprodotto nuovoprod;
    tlista elencoprod1=NULL;
    tlista elencoprod2=NULL;
    tlista unione = NULL;
    int carletto;

    do {
        printf ("\n\n\t\t/* MENU' */\n\n");

        printf ("1. Inserisci i dati di un nuovo prodotto del
                 mag1.\n");
        printf ("2. Stampa i prodotti presenti nel magazzino 1.\n");
        printf ("3. Inserisci i dati di un nuovo prodotto del
                 mag2.\n");
        printf ("4. Stampa i prodotti presenti nel magazzino 2.\n");
        printf ("5. Unisci i prodotti presenti nei due magazzini
                 (distrugge le liste)");
        printf ("\n6. Esci dal programma\n");

        carletto=getch();
```

Esercizio L_15 - Magazzino


```
case  '1': acquisisci (&nuovoprod);
                  insord (&elencoprod1, nuovoprod);
                  break;

case  '2': printf ("\n\nElenco dei prodotti nel magazzino 1 :\n");
                  stampa (elencoprod1);
                  break;


case  '3': acquisisci (&nuovoprod);
                  insord (&elencoprod2, nuovoprod);
                  break;


case  '4': printf ("\n\nElenco dei prodotti nel magazzino 2 :\n");
                  stampa (elencoprod2);
                  break;
```

Esercizio L_15 - Magazzino

```
            case  '5': unione = fondi(elencoprod1, elencoprod2);
                       elencoprod1 = elencoprod2 = NULL;
                       // reset necessario, non sono più utilizzabili
                       printf ("\n\nEcco i prodotti presenti nei due
                                                        magazzini:\n");
                       stampa (unione);
                       break;

            case  '6': printf ("\nBye Bye!!!!\n");
                       break;

        }//switch

    }while (carletto != '6');

}//main
```

Esercizio L_15 - Magazzino


```
void acquisisci (tprodotto *new) {


      printf ("Inserisci i dati di questo prodotto:
                              \n\n\tNOME PRODOTTO: ");


      scanf ("%s", new->merce);


      printf ("\n\tQUANTITA': ");


      scanf("%d", &new->quantita);




}//end acquisisci
```

Esercizio L_15 - Magazzino


```
void inserisci (tlista* lista, tprodotto nuovo){



        telemento *temp;

        temp=(telemento*)malloc(sizeof(telemento));

        temp->prodotto = nuovo;

        temp->link = *lista;

        *lista = temp;




}//end inserisci
```

Esercizio L_15 - Magazzino


```
void insord (tlista*lista,tprodotto nuovoprod) {
   telemento*prec;
   telemento*punt;
   telemento*temp;

   punt=*lista;

   if (*lista==NULL ||
            strcmp ( (*lista)->prodotto.merce, nuovoprod.merce) >= 0)
            inserisci (lista, nuovoprod);
   else {
            while (punt!=NULL
               && strcmp (punt->prodotto.merce, nuovoprod.merce) < 0)
            {   prec=punt;
                punt=punt->link;
            }
            temp = (telemento*)malloc(sizeof(telemento));
            temp->prodotto = nuovoprod;
            prec->link = temp;
            temp->link = punt;
   }// chiudo else
}//end insord
```

Esercizio L_15 - Magazzino


```
void stampa (tlista lista){



        telemento *punt;
        punt= lista;

        while (punt!=NULL)
        {

                printf("PRODOTTO: %s\tQUANTITA': %d\n"
                                , punt->prodotto.merce
                                , punt->prodotto.quantita);

                punt=punt->link;

        }



}//end stampa
```

Esercizio L_15 - Magazzino


```
void estrai (telemento **punt, tlista *l) {

        *punt = *l;

        *l = (*l)->link;

}//end estrai
```

Esercizio L_15 - Magazzino


```
tlista fondi (tlista l1, tlista l2) {

    telemento *x = NULL;
    telemento *ultimo = NULL;
    telemento *t = NULL;
    while (l1!=NULL || l2!=NULL)
    {
        if (l2==NULL)
            estrai (&x, &l1);
        else if (l1==NULL)
            estrai (&x,&l2);
        else if (strcmp (l1->prodotto.merce, l2->prodotto.merce) > 0)
            estrai (&x,&l2);
        else if (strcmp (l1->prodotto.merce, l2->prodotto.merce) < 0)
            estrai (&x,&l1);
        else {
            estrai(&x,&l1); // gli articoli sono uguali
            l2->prodotto.quantita += x->prodotto.quantita;
            free (x);
            estrai(&x,&l2);
        }
```

Esercizio L_15 - Magazzino

```
            if (t==NULL)
            {
                    t=x;
                    ultimo=x;
                    t->link=NULL;
            }

            else
            {
                    ultimo->link=x;
                    ultimo=x;
                    x->link=NULL;
            }

    }//end while

    return t;


}//end fondi
```