Esercitazioni di Prog. II (esercizi su liste d liste)

Chiara Petrioli

‘Glicine’

```
struct nodoramo{
    int valore;
    struct nodoramo *succ;
};

struct elemglicine{
    struct nodoramo *ramo;
    struct elemglicine *next;
};
typedef struct nodoramo NODO;
typedef NODO *RAMO;
typedef struct elemglicine GLICINE2;
typedef GLICINE2 *GLICINEPTR;

void stampaglicine(GLICINEPTR);
int sommapari(GLICINEPTR);
int sommapariramo(RAMO);

main()
.....
```

‘Glicine’

```
main()
{
	int insramo;
	int i;
	int valore;
	int dimramo;
	GLICINEPTR GG=NULL;
	RAMO RR;
	GLICINEPTR tempg;
	RAMO tempr;
	printf("vuoi inserire un nuovo ramo? se si scrivi 1 altrimenti 0 \n");
	scanf("%d",&insramo);
	while (insramo)
	{
		tempg=malloc(sizeof(GLICINE2));
		tempg->next=GG;
		tempg->ramo=NULL;
		GG=tempg;
		RR=NULL;
		printf("numero di elementi del ramo?\n");
		scanf("%d",&dimramo);
		for (i=0;i<dimramo;i++)
		{
			printf("inserisci valore primo elemento \n");
			scanf("%d",&valore);
			tempr=malloc(sizeof(NODO));
			tempr->valore=valore;
			tempr->succ=RR;
			RR=tempr;
		}
		GG->ramo=RR;
		printf("vuoi inserire un nuovo ramo? se si scrivi 1 altrimenti 0 \n");
		scanf("%d",&insramo);
	}
	printf("stampa del glicine \n");
	stampaglicine(GG);
	printf("il numero di occorrenze dei valori pari e' %d \n", sommapari(GG));
}
```

Esercizio 1

```
/*Post: stampa i valori contenuti in un glicine*/

void stampaglicine (GLICINEPTR G)
{
    RAMO r;
    if (G!=NULL)
    {
        printf("[ ] ->");
        r=G->ramo;
        while(r!=NULL)
        {
            printf("[[%d]]->",r->valore);
            r=r->succ;
        }
        printf("\n | \n");
        stampaglicine(G->next);
    }
}
```

Si scriva una procedura che, dato un glicine Ne stampi gli elementi

Esercizio 2

```
/*Post:Restituisce il numero dei valori pari nel ramo corrente*/

int sommapariramo(RAMO R)
{
    if(R==NULL)
        return 0;
    else
        return((R->valore%2)?sommapariramo(R->succ):(1+sommapariramo(R->succ)));
}
/*Post: restituisce il numero di valori pari nel glicine*/

int sommapari(GLICINEPTR G)
{
    if (G==NULL)
        return 0;
    else
        return(sommapariramo(G->ramo)+sommapari(G->next));
}
```

Si scriva una funzione Che dato un GLICINE calcoli La somma degli elementi Pari nel GLICINE

Esercizio 2

```
/*Post:Restituisce il numero dei valori pari nel ramo corrente*/

int sommapariramo(RAMO R)
{
    if(R==NULL)
        return 0;
    else
        return((R->valore%2)?sommapariramo(R->succ):(1+sommapariramo(R->succ)));
}
/*Post: restituisce il numero di valori pari nel glicine*/

int sommapari(GLICINEPTR G)
{
    if (G==NULL)
        return 0;
    else
        return(sommapariramo(G->ramo)+sommapari(G->next));
}
```

Si scriva una funzione Che dato un GLICINE calcoli La somma degli elementi Pari nel GLICINE

Esercizio 3

```
int elementiramo(RAMO R)
{
      if(R==NULL)
                return 0;
      else
                return(1+elementiramo(R->succ)));
}


int profondita(GLICINEPTR G)
{     int temp, temp1;
       if (G==NULL)
                return 0;
       else
     {
        temp=profondita(G->next);
      temp1=elementiramo(G->ramo);
      if (temp>=temp1) return temp;
      else return temp1;
}
}
```

Si scriva una funzione Che dato un GLICINE ne calcoli la Profondita' (lunghezza Ramo piu' lungo)

Esercizio 4

```
int elegante(GLICINEPTR G)
{   if ((G==NULL)|| (G->next==NULL))
          return 1;
    else
    {
     return ((elementiramo(G->ramo)<= elementiramo(G->next->ramo))
      &&(elegante(G->next));
}
}
```

Si scriva una funzione che dato un GLICINE verifichi se e' 'elegante'
Un glicine si dice elegante se i suoi Rami hanno una lunghezza crescente