E12 – Esercizi su Strutture dati dinamiche in C

Esercizio 1. Si vuole realizzare una lista caratteri in C utilizzando una rappresentazione semplicemente collegata. Scrivere una struttura **nodo** che contiene due campi: **info** di tipo **char** e **next** di tipo puntatore a **nodo**. Scrivere poi le seguenti funzioni per la gestione della lista:

- **stampa**, che dato un puntatore al nodo iniziale della lista ne stampa il contenuto secondo la notazione parentetica.
- **inserisci**, che dato un puntatore al nodo iniziale della lista (passato per riferimento) e un carattere **c**, inserisce **c** in testa alla lista (cioè in prima posizione).
- **cerca**, che dato un puntatore al nodo iniziale della lista e un carattere **c**, restituisce la posizione di **c** all'interno della lista, se **c** è presente nella lista, **-1** in caso contrario.
- **rimuovi**, che dato un puntatore al nodo iniziale della lista (passato per riferimento) e un carattere **c**, rimuove la prima occorrenza di **c** dalla lista, se **c** è presente nella lista, non fa nulla altrimenti.

Scrivere poi un programma che permetta di testare le funzioni scritte.

Esercizio 2. Aggiungere al programma dell'esercizio precedente una funzione **concatena**, che dati **l1** ed **l2,** puntatori ai nodi iniziali di due liste distinte, crea una nuova lista concatenando le due liste date. Il puntatore iniziale alla nuova lista sarà **l1** (che quindi va passato per riferimento).

Esercizio 3. Modificare la funzione **inserisci** dell'Esercizio 1 in maniera che la lista sia mantenuta ordinata alfabeticamente, cioè i caratteri presenti nella lista appaiano ordinati alfabeticamente. Come possono le funzioni **cerca** e **rimuovi** sfruttare l'ordinamento alfabetico della lista? Come è necessario modificare la funzione **concatena** per far sì che, date due liste ordinate, la lista concatenata sia anch'essa ordinata alfabeticamente?

Soluzioni

Esercizio 1 - svolgimento.

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

struct nodo{
      char info;
      struct nodo * next;
};

typedef struct nodo node;
typedef node * nodePtr;

void stampa(nodePtr list); // stampa l'intera lista
void inserisci(nodePtr *list, int value);  // inserisce value in testa
                                           // alla lista puntata da *list
int cerca(nodePtr list, int value);   // cerca value nella lista puntata
                                      // da list e restituisce il
                                      // puntatore al nodo che lo
                                      // contiene (NULL se value non è
                                      // presente nella lista)
void rimuovi(nodePtr *list, int value);    // rimuove la prima occorrenza
                                           // di value nella lista
                                           // puntata da *list (non fa
                                           // nulla se value non è
                                           // presente nella lista)

int main(){
      nodePtr head=NULL; // lista vuota

      printf("Opzioni:\n");
      printf("1 - Inserisci un elemento\n");
      printf("2 - Cerca un elemento\n");
      printf("3 - Rimuovi un elemento\n");
      printf("4 - Esci\n");

      int opzione;
      printf("Scegli un'opzione\n");
      scanf("%d", &opzione);
      while(opzione!=4){
            if(opzione==1){
                  char value;
                  printf("Dammi l'elemento da inserire\n");
                  scanf(" %c", &value);
                  inserisci(&head, value);
                  stampa(head);
                  printf("\n");
            }else if(opzione==2){
                  char value;
                  printf("Dammi l'elemento da cercare\n");
                  scanf(" %c", &value);
                  int i=cerca(head, value);
                  if(i!=-1)
                        printf("L'elemento si trova in pozione %d\n", i);
                  else
                        printf("L'elemento non e' presente nella lista\n");
```

```
            stampa(head);
            printf("\n");
        }else if(opzione==3){
            char value;
            printf("Dammi l'elemento da rimuovere\n");
            scanf(" %c", &value);
            rimuovi(&head, value);
            stampa(head);
            printf("\n");
        }else if(opzione!=4){
            printf("Opzione non valida");
        }
        printf("Scegli un'opzione\n");
        scanf("%d", &opzione);
    }

}

void stampa(nodePtr list){
    printf("("); // stampa la parentesi aperta
    nodePtr p=list; // puntatore per scorrere la lista

    while(p!=NULL){ // finché la lista non è finita
        printf("%c",p->info); // stampa l'elemento corrente...
        p=p->next; // ...e avanza
        if(p!=NULL)     // se il prossimo elemento esiste
            printf(", "); // stampa una virgola
    }

    printf(")"); // stampa la parentesi chiusa
}

void inserisci(nodePtr *list, int value){
    nodePtr newPtr=malloc(sizeof(node)); // alloca memoria per un nuovo
                                         // nodo
    if(newPtr!=NULL){     // se l'allocazione è andata a buon fine
        newPtr->info=value; // scrivo il campo info del nuovo nodo...
        newPtr->next=*list;  // e il suo campo next (che punterà
                             // all'attuale primo nodo)
        *list=newPtr; // il puntatore iniziale punta al nuovo nodo
    }
}

int cerca(nodePtr list, int value){
    nodePtr p=list; // puntatore per scorrere la lista
    int r=-1;  // valore da restituire (posizione di value nella lista)

    int i=0; // indice per contare le posizioni
    while(p!=NULL && r==-1){   // finché la lista non è terminata e non
                               // ho ancora trovato l'elemento
        if(p->info==value)     // se il nodo corrente contiene
                               // l'elemento cercato
            r=i;  // poni r pari al nodo corrente
        p=p->next; //avanza...
        i++; // ...ed incrementa l'indice
    }
```

```
	return r;

}

void rimuovi(nodePtr *list, int value){
	if(*list!=NULL){// se la lista non è vuota
		if((*list)->info==value){ // se l'elemento da rimuovere è il
					  // primo
			nodePtr p =(*list); // puntatore al nodo da rimuovere
			*list=(*list)->next; // il puntatore *list viene fatto
					     // puntare al nodo successivo (o a
					     // NULL se c'era un solo nodo)
			free(p); // il nodo da rimuovere viene deallocato
		}else{ // el. da rimuovere in posizione successiva alla prima
			nodePtr p=*list; // puntatore per scorrere la lista

			// finché non si è raggiunto l'ultimo nodo e il
			// successore non contiene il valore da rimuovere
			while(p->next!=NULL && p->next->info!=value)
					p=p->next; // avanza

			if(p->next!=NULL){ // se non sono arrivato sull'ultimo
					//(quindi il successivo è da rimuovere)
				nodePtr q=p->next; // punt. al nodo da rimuovere
				p->next=p->next->next; // il predecessore del nodo
						       // da rimuovere viene fatto
						       // puntare al successore del
						       // nodo da rimuovere
				free(q); // il nodo da rimuovere viene deallocato
			}
		}
	}
}
```

Esercizio 2 - svolgimento.

```
void concatena(nodePtr *l1, nodePtr l2){
	if(*l1==NULL) //se la prima lista è vuota
		*l1=l2;    // la lista concatenata coincide con la seconda
	else{
		nodePtr p=*l1;
		//scorre la lista fino all'ultimo elemento
		while(p->next!=NULL)
			p=p->next;

		p->next=l2; //l'ultimo nodo della prima lista
			    // punta al primo della seconda
	}

}
```

Esercizio 3 – svolgimento

```
void inserisci(nodePtr *list, int value){
      nodePtr newPtr=malloc(sizeof(node)); // alloca memoria per un nuovo
                                           // nodo
      if(newPtr!=NULL){     // se l'allocazione è andata a buon fine
            newPtr->info=value;   // scrivo il campo info del nuovo nodo
            if(*list==NULL){// se la lista è vuota
                  newPtr->next=NULL; // il nuovo nodo non avrà successore
                  *list=newPtr;   // *list deve puntare al nuovo nodo
            }else if((*list)->info>=value){ // inserimento in testa
                  newPtr->next=*list;
                  *list=newPtr;
            }else{     // inserimento in altra posizione
                  nodePtr p=*list; // nodo per scandire la lista
                  nodePtr prec=NULL; //nodo che punta il predecessore di p
                  while(p!=NULL && p->info<value){//finché l'elemento
                                                  // corrente esiste ed ha
                                                  //valore minore di value
                        prec=p; //avanzo con prec
                        p=p->next; //e con p
                  }
                  newPtr->next=p; // il successore del nuovo nodo è p
                  prec->next=newPtr; // il succ. di prec è il nuovo nodo

            }
      }
}
```

Le funzioni cerca e rimuovi possono essere modificate in maniera da scorrere la lista cercando l'elemento di interesse solo finché l'elemento corrente è minore di quello cercato.

```
int cerca(nodePtr list, int value){
      nodePtr p=list; // puntatore per scorrere la lista
      int r=-1;  // valore da restityire (posizione di value nella lista)
      int i=0; // indice per contare le posizioni
      while(p!=NULL && p->info<value){// finché la lista non è terminata
                                      // e l'elemento corrente è minore
                                      // di quello che sto cercando
            p=p->next; //avanza
            i++; // ed incrementa l'indice
      }
      if(p!=NULL && p->info==value) // se non ho scandito tutta la lista
                                    // e l'elemento corrente è quello che
                                    // sto cercando
            r=i; // ho trovato l'elemento
      return r;

}

void rimuovi(nodePtr *list, int value){
      if(*list!=NULL){// se la lista non è vuota
            if((*list)->info==value){ // se l'elemento da rimuovere è il

                                      // primo
                  nodePtr p =(*list);  // puntatore al nodo da rimuovere
                  *list=(*list)->next; // il puntatore *list viene fatto
                                       // puntare al nodo successivo (o a
```

```
                                        // NULL se c'era un solo nodo)
                free(p); // il nodo da rimuovere viene deallocato
          }else{ // elemento da rimuovere in posizione successiva alla
                 // prima
                nodePtr p=*list; // puntatore per scorrere la lista

                // finché non si è raggiunto l'ultimo nodo e il
                // successore contiene un valore minore di value
                while(p->next!=NULL && p->next->info<value)
                      p=p->next; // Avanza

                if(p->next!=NULL && p->next->info==value){
                // se non sono arrivato sull'ultimo e il successivo è da
                // rimuovere
                      nodePtr q=p->next; //puntatore al nodo da rimuovere
                      p->next=p->next->next; //il predecessore del nodo
                                             // da rimuovere viene fatto
                                             // puntare al successore del
                                             // nodo da rimuovere
                      free(q); // il nodo da rimuovere viene deallocato
                }
          }
     }
}
```

Un modo per modificare la funzione **concatena** garantendo che la funzione risultato sia ordinata è di creare una nuova lista in cui vengono inseriti, tramite la funzione **inserisci**, tutti gli elementi delle due liste date. La funzione inserisci garantisce che gli elementi inseriti siano inseriti mantenendo la lista ordinata. È necessario gestire la deallocazione delle due vecchie liste.

```
void concatena(nodePtr *l1, nodePtr l2){
     nodePtr l=NULL; // puntatore alla lista risultato

     // scandisce la prima lista e copia tutti gli elementi nella lista
     // risultato
     nodePtr p=*l1;
     while(p!=NULL){
          inserisci(&l,p->info);
          p=p->next;
     }

     // scandisce la seconda lista e copia tutti gli elemnti nella lista
     // risultato
     p=l2;
     while(p!=NULL){
          inserisci(&l,p->info);
          p=p->next;
     }

     // dealloca le vecchie liste
     nodePtr q;
     p=*l1;
     while(p!=NULL){
          q=p;
          p=p->next;
          free(q);
```

```
        }

        p=l2;
        while(p!=NULL){
              q=p;
              p=p->next;
              free(q);
        }

        *l1=l; // aggiorna il puntatore *l1;

}
```