Politecnico di Milano - Dipartimento di Elettronica, informazione e Bioingegneria

Prof. Mauro Negri

Fondamenti di Informatica **3 marzo 2015**
I appello

Matricola/codice persona_____Cognome_____Nome_____

Durata prova: 1 ora 30 minuti

Istruzioni

LE RISPOSTE DEVONO ESSERE SCRITTE SU QUESTO PLICO (anche sul retro dei fogli)
Si può utilizzare la matita. Vietato l'uso di cellulari, libri, eserciziari, appunti o calcolatrici.
Non è possibile uscire dall'aula durante la prova tranne in casi eccezionali.
La copiatura accertata di parti del compito può comportare la valutazione riprovato dei compiti coinvolti

Esercizio 1 (6punti) _____

Esercizio 2 (3 punti) _____

Esercizio 3 (7 punti) _____

Esercizio 4 (5 punti) _____

Esercizio 4 (5punti) _____

Esercizio 4 (4 punti) _____

Punteggio totale _____

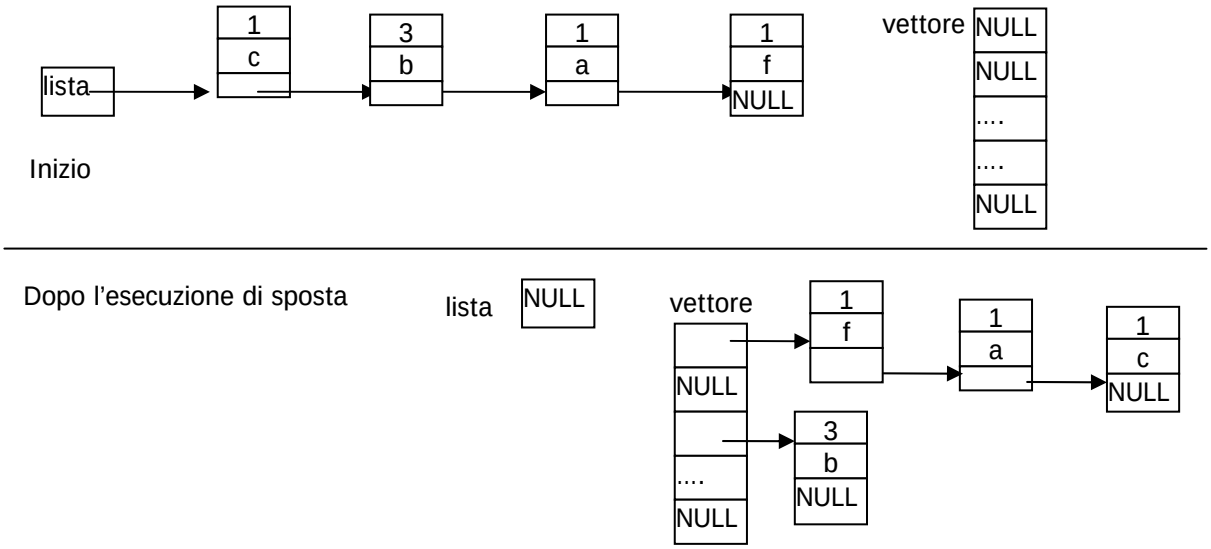
Esercizi 1, 2, 3, 4 Si supponga di avere la seguente lista monodirezionale
struct elemento { int codice; char valore; struct elemento *prossimo;};
struct elemento *lista=NULL;

e di avere la seguente struttura dati vett (vettore di puntatori a liste monodirezionali)
struct elemento *vett[100] `all'inizio dell'esecuzione del programma ogni casella del vettore è inizializzata a NULL.`

ATTENZIONE (nota implementativa)
Le funzioni non devono ipotizzare che i valori dei parametri delle funzioni dedicati al passaggio parametri della lista monodirezionale o di un singolo elemento della lista stessa siano corretti. Pertanto effettuare tutti i controlli necessari per evitare che le funzioni generino errori in esecuzione (come ad esempio referenziare un campo di un elemento che non esiste).
Condizioni garantite e quindi da non controllare nel codice delle funzioni: il campo codice di un elemento della lista contiene sempre un valore compreso tra 1 e 100 e il parametro che rappresenta il vettore è sempre corretto.

1. Scrivere una funzione insert che riceve come parametri il puntatore ad un elemento dinamico di tipo *struct elemento* già esistente, una posizione compresa tra 1 e 100 e il vettore vett; la funzione provvede ad identificare la casella del vettore usando il parametro posizione e ad inserire l'elemento ricevuto, se esiste, nella coda associata a tale casella. L'inserimento avviene in coda alla lista. La funzione ritorna come parametro il vettore modificato. Nel caso in cui ci sia un errore nell'elemento o nella posizione ricevuti la funzione non effettua alcuna operazione.
2. Scrivere una funzione lungh che riceve una lista monodirezionale come parametro e restituisce come parametro il numero di elementi della lista.
3. Scrivere una funzione estrai che riceve una lista monodirezionale come parametro. La funzione sconnette l'ultimo elemento della lista stessa e ritorna come parametri la lista modificata e il puntatore all'elemento estratto; deve inoltre ritornare la lunghezza della lista dopo l'estrazione come parametro passato per valore ritornato; se la lista è vuota la funzione ritorna -1 e se infine i parametri sono errati ritorna -2 (sempre passato per valore ritornato al posto della lunghezza della lista). Solo il valore numerico ritornato può essere specificato nel passaggio parametri per valore ritornato.
4. Scrivere una funzione sposta che riceve come parametri la lista e il vettore globali, esegue l'estrazione di tutti gli elementi della lista monodirezionale uno ad uno a partire dall'ultimo elemento (usare le funzioni sopra definite); la funzione prende il valore del campo codice del singolo elemento dinamico estratto e lo usa per individuare la casella del vettore vett a cui associare l'elemento e poi provvede ad inserirlo in coda alla lista della casella individuata (usare le funzioni sopra definite). Alla fine dell'esecuzione vengono ritornati la lista e il vettore modificati come parametri e il valore 1 come parametro passato per valore ritornato. In caso di errore nei parametri o nel caso in cui la funzione *estrai* ritorni -2, la funzione ritornerà -2 sempre come parametro passato per valore ritornato. Solo il valore numerico ritornato può essere specificato nel passaggio parametri per valore ritornato.

Nella seguente figura si vede lo stato iniziale del vettore e un possibile stato iniziale della coda. Sotto si vede invece lo stato della coda e del vettore dopo l'esecuzione della funzione sposta.



Le soluzioni proposte non descrivono l'unico modo di affrontare gli esercizi.
La novità in questo esercizio era data dal forzare la definizione dei parametri e conseguentemente il controllo della loro correttezza (vedi codice -2).
Altro aspetto riguarda il riutilizzo delle funzioni (lungh in estrai e di estrai e insert in sposta), pertanto sono state penalizzate le soluzioni che reimplementavano al posto del riutilizzo come richiesto dal testo.

```
#define MAX 10
struct elemento {
     int codice;
     char valore;
     struct elemento *prossimo;
};
struct elemento *lista=NULL;
struct elemento *vett[MAX];

void insert (struct elemento *p, int pos, struct elemento *v[MAX])
{ struct elemento *temp;
   if ((pos>0)&&(pos<=100))
     { temp = v[pos-1];
        if (p!=NULL)
            { if (v[pos-1] == NULL) v[pos-1] = p;
               else
                 { while (temp->prossimo!=NULL) temp=temp->prossimo;
                    temp->prossimo=p;
                 }
            }
      }
}

int lungh (struct elemento *l)
{
     int nr=0;
     while (l != NULL)
     {
          l = l->prossimo;
          nr++;
     }
     return nr;
}

int estrai (struct elemento **l, struct elemento **el)
{
       struct elemento *current = *l, *prec=NULL;

       if ((l==NULL)|| (el==NULL)) return -2;
       if (current == NULL)        return -1;

       // 1 solo elemento
        if  (current->prossimo == NULL) *l = NULL;

       // più elementi
       while (current->prossimo != NULL)    {prec=current; current = current->prossimo; }

       prec->prossimo = NULL;
       *el = current;

       return lungh(*l);
}
```

```
int sposta (struct elemento **l, struct elemento *v[MAX])
{
        struct elemento *el = NULL;
        int ris;

        if (l==NULL) return -2;
        ris = estrai(l, &el);
        while(ris>0)
                { insert(el, el->codice, v);
                  ris = estrai(l, &el);
                }
   if (ris == -2) return -2; else return 1;

}
```

Esercizio 5.

1. Data la definizione della variabile float v= - 79,99 descrivere la sua rappresentazione in virgola mobile in base allo standard IEEE754 semplice precisione FP32 descritto a lezione;
   a) Definire prima il numero nel formato normalizzato intermedio (TRONCARE parte decimale alla **seconda** cifra decimale):

$$-1{,}24 * 2^6$$

   b) Codificare poi il numero normalizzato usando il formato FP32:
      S (1 bit di segno): 1
      M (23 bit per la mantissa – fermarsi ai primi 5 bit): 00111
      E (8 bit per l'esponente): 10000101

3. Indicare quale sia il vero valore (in base 10) descritto dalla sequenza di bit memorizzata in S, M ed E del punto b) precedente.

-78

4. Dato il seguente programma C

```
…
I1  struct {int el1; int el2;}  V; int C;
I2  int fgetfq (int p1, int p2, int *p3) {float v;…….}
I3  int main ()
I4     { V.el2=4;
I5       C=getfq (V.el1, V.el2, &C);
          }
```

Tradurre in assembler le definizioni di riga I1e le istruzioni di riga I3, I4 e I5 (i registri R1-R6 sono liberi). La funzione non ha istruzioni.

```
I1
Vel1: . RES 1
Vel2: . RES 1
C: . RES 1
I3
STACK:.RES 1000
MAIN: MOV #STACK, SP
        ADD #999, SP
I4
        MOV #4, Vel2
I5
        ADD #-1,SP
        MOV Vel1, (SP)
        ADD #-1,SP
        MOV Vel2, (SP)
        ADD #-1,SP
        MOV #C, (SP)
        ADD #-1,SP
```

```
        JSR getfq
RTS:    ADD #4, SP
        MOV (SP), C
```

Esercizio 6. Scrivere un programma C che definisce le variabili, int status, pid_t pid e int cont, poi crea due processi figli (figlio1 e figlio2), quindi si mette in attesa della terminazione di figlio1 e quando è risvegliato visualizza il valore di status ricevuto da figlio1. Successivamente passa ad eseguire il programma P.exe (da non implementare) nella cartella \root\miei programmi, passando due stringhe come parametri di nome Q e R rispettivamente. Dopo l'esecuzione del programma P.exe si mette in attesa della terminazione di figlio2 per poi procedere alla visualizzazione del valore ricevuto dal figlio. Infine termina l'esecuzione a propria volta ritornando il valore 111. Figlio1 visualizza al terminale il proprio identificatore di processo e poi termina l'esecuzione ritornando il numero 10 al padre. Figlio2 termina immediatamente la propria esecuzione ritornando il proprio identificatore di processo.
In caso di un errore durante l'esecuzione delle funzioni di sistema operativo terminare l'esecuzione del processo padre con codice di errore -1.
Commentare brevemente eventuali ipotesi che sono state fatte per l'implementazione di quanto richiesto.

```
#include ….
int status; pid_t pid; int cont;
pid_t figlio1; int ris; char Q[10],R[10];
int main ()
    { pid=fork(); if  (pid==-1) exit(-1);
                  if (pid==0)     {printf("%i", getpid()); exit(10);        }

         else
           {figlio1=pid;
          pid=fork(); if  (pid==-1) exit(-1);
              if (pid==0) exit(getpid());
          else
             { pid=waitpid(figlio1,&status,0);
             ris =execl("/root/mieiprogrammi/p.exe", Q,R, NULL);
               if (ris==-1) exit (-1);
                }
         }
     }
```

L'attuale programma eseguito dal processo padre viene sostituito da p.exe dopo la waitpid e quindi la richiesta del testo di aspettare il figlio 2 e ritornare 111 dovrebbe appartenere al programma P.c che però non andava implementato. Unico aspetto da considerare è che l'execl può fallire e allora in questo caso il programma corrente deve prevedere exit(-1) come richiesto dal testo.