Politecnico di Milano - Dipartimento di Elettronica, informazione e Bioingegneria

Prof. Mauro Negri

Fondamenti di Informatica **6 luglio 2015**
II 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 (9 punti) _____

Esercizio 2 (2 punti) _____

Esercizio 3 (3 punti) _____

Esercizio 4 (5 punti) _____

Esercizio 5 (6 punti) _____

Esercizio 6 (5 punti) _____

Punteggio totale _____

Esercizi 1, 2, 3, 4 Si supponga di avere la seguente struttura dati

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define maxc 10
#define maxh 100
struct coppia { int chiave; char valore[maxc]; struct coppia *prossimo;};
struct coppia *HashMap[maxh];
```

Ogni posizione della HashMap rappresenta la testa di una lista monodirezionale.

Esercizio 1. Scrivere una funzione Insert che riceve come parametri un valore (stringa) e una chiave (intero) e la struttura dati Hahmap ed esegue le seguenti operazioni:

- Controlla che il valore sia una stringa di lunghezza < (maxc -2) e in caso negativo ritorna -1, altrimenti estende la stringa aggiungendo il carattere 'C' come prefisso (cioè come primo carattere della stringa).
- Genera un elemento dinamico contenente chiave e il valore modificato nei campi informativi. Se fallisce la funzione ritorna -2
- Invoca una funzione int cerca (int chiave) che riceve un valore di chiave e restituisce un intero compreso tra 0 e maxh-1 per chiavi non negative. Nel caso di valori negativi la funzione ritorna -3.
- Inserisce l'elemento dinamico in testa alla lista associata alla posizione restituita dalla funzione cerca.
- Restituisce la hashmap modificata e il valore 1.

NOTA: Nei casi in cui la funzione non possa completare tutte le operazioni previste nei precedenti punti, la funzione deve terminare la propria esecuzione lasciando le strutture dati e la memoria nello stato in cui era all'atto dell'esecuzione della funzione stessa.

```
int Insert(char * str, int key, struct coppia *hash[])
// o int Insert(char * str, int key, struct coppia **hash)
    {
      int i;int pos; struct coppia *el;

      i = strlen(str);
      if (!(i < (maxc - 2))) return -1;

      while(i > 0) { str[i+1] = str[i];     i--;  }
      str[0] = 'C';

      el = (struct coppia*) malloc(sizeof(struct coppia));
      if (el == NULL) {printf("out of memory\n"); return -2;}

      el->chiave = key;
      strcpy(el->valore, str);
      el->prossimo = NULL;

      pos = cerca(key);
      if (pos == -3) { free(el);return -3;}  //free per lasciare la memoria nello stato pre insert

      el -> prossimo = hash[pos];
      hash[pos] = el;
      return 1;
}
```

Esercizio 2. Indicare come e dove debba essere inizializzata (se serve) la struttura dati hashmap affinchè la funzione di insert funzioni correttamente.

Dove
Nell'area globale dei dati o nell'area locale del main (questo caso possibile perché comunque la struttura viene passata come parametro.
Come
Assegnando NULL pointer a tutti gli elementi del vettore.

Esercizio 3. Scrivere la funzione cerca utilizzata nell'esercizio precedente. Il valore restituito è calcolato in modo che alla chiave 0 restituisce 0, alla chiave 1 restituisce 1,…, alla chiave 99 restituisce 99, alla chiave 100 restituisce 0, alla chiave 101 restituisce 1, alla chiave 199 restituisce 99, alla chiave 200 restituisce 0, alla chiave 201 restituisce 1 e così via.

L'esempio sopra riportato identifica una funzione di calcolo basato sul resto della divisione intera.
Inoltre dall'esercizio 1 si può dedurre che il calcolo non abbia senso per valori negativi.

```
int cerca(int key) {
        if (key < 0) return -3;
        return key % 100;
}
```

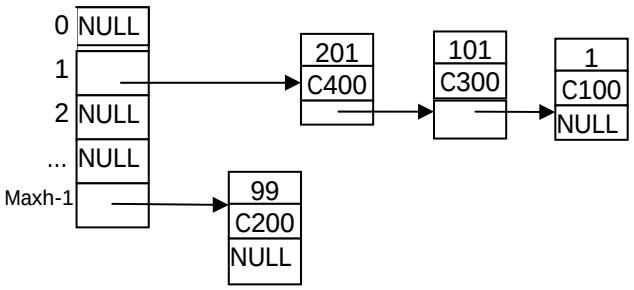
Esercizio 4. Scrivere una funzione main che legge al terminale una sequenza di coppie <chiave, valore> e le inserisce nella hashmap; nel caso in cui l'inserimento fallisca si passa alla coppia successiva. La funzione termina quando il valore di chiave di una coppia è uguale a 3000.

Nella seguente figura si mostra una possibile sequenza di inserimento di coppie <chiave, valore> e lo stato conseguente della coda e del vettore dopo l'esecuzione della funzione sposta.

Coppie ricevute sequenzialmente

Chiave	Valore
1	100
99	200
101	300
201	400

Hashmap prodotta




```
int main() {

        int I,key, res;
        char val[maxc];

        struct coppia * l;
         int found;//test

        for (i = 0; i < maxh; i++) HashMap[i] = NULL;

        printf("chiave? ");
        scanf("%d", &key);
        while(key != 3000) {// ciclo do..while utilizzabile solo con test aggiuntivo su "3000".
```

```
            printf("valore? ");
            scanf("%s", val);
            printf("attivazione insert coppia <%d, %s>\n", key, val);
          switch(Insert(val, key, HashMap)) {
               case -1: printf("stringa troppo lunga, insert NON eseguito\n"); break;
             case -2: printf("no memoria, Insert NON eseguito\n"); break;
             case -3: printf("pos negativa, Insert NON eseguito\n");break;
             case 1: printf("insert avvenuto con successo\n");
            }
                                  // TEST per verifica comportamento
                                     found = 0;
                                     for (i = 0; i < maxh; i++) {
                                       l = HashMap[i];
                                       if (l != NULL) {
                                         if (!found) {printf("liste NON nulle: \n");found = 1; }
                                             printf("POS %d ==> ", i);
                                             while(l != NULL) {
                                                  printf("<%d, %s> -> ",l->chiave, l->valore);
                                               l = l->prossimo;
                                             }
                                                  printf("NULL\n");
                                       }
                                     }
                                     if (!found) {printf("Attualmente tutte le liste sono vuote\n");}
                                  // FINE TEST
            printf("chiave? ");
            scanf("%d", &key);
          }
}
```

Esercizio 5.

1. Data la definizione della variabile float v= - 85,85 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):

$$- 1{,}34*2^6$$

   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): 01010
      E (8 bit per l'esponente): 1000 0101

2. Indicare quale sia il vero valore (in base 10) descritto dalla sequenza di bit memorizzata in S, M ed E del punto b) precedente; si può mettere l'espressione di calcolo utilizzata, ma si DEVE scrivere anche il valore finale prodotto.

$$-84 = (1+1/4+1/16) * 2^6$$

3. La rappresentazione in complemento a 2 di un integer e la rappresentazione FP32 di un numero reale utilizzano entrambi 32 bit. Indicare:

a) l'intervallo di valori rappresentabili in entrambe le rappresentazioni: valore del numero più piccolo e più grande rappresentabili (esprimere tali valori come numero assoluto o come espressione per il loro calcolo)

int: da -2^{31} a $2^{31} -1$

FP32: (bit mantissa tutti a 1) ≅ da $- 2^{128}$ a (bit mantissa tutti a 0) -2^{-126} e da 2^{-126} a 2^{128}

b) quanti numeri interi (in complemento a 2) e reali (in FP32) sono rappresentati nell'intervallo dei valori [2,4[(4 escluso)

int: i valori 2 e 3.
FP32: 1,xx $*2^1$ cioè consuma 1 bit mantissa per la parte intera e lascia 22 bit per la parte frazionaria; ci sono quindi 2^{22} configurazioni di bit possibili (numeri reali rappresentati)

c) come sia possibile che l'intervallo dei numeri reali sia più grande di quello dei numeri interi se si utilizza lo stesso numero di bit per entrambe le rappresentazioni.

Il punto b) asserisce che tra 2 e 4 ci sono moltissimi numeri reali rappresentati e quindi il numero di configurazioni di bit utilizzato è >> delle 2 utilizzate per integer. Se fosse così in tutti gli intervalli l'affermazione sarebbe errata. Tuttavia la distanza tra due numeri reali rappresentati non è costante, ma aumenta all'aumentare nel numero. Quando tutti i bit sono usati per la codifica della sola parte intera la codifica FP comincia ad approssimare anche la parte intera a tal punto da dilatare l'intervallo di rappresentazione come detto al punto a) - vedi lucidi del corso.

Esercizio 6. Scrivere un programma C che definisce le variabili, int status, pid_t pid e FILE *fp, poi apre il file miofile.txt di tipo testo con modalità che permetta la creazione automatica se non esiste il file oppure che ne azzeri il contenuto se esiste e ne permetta comunque la lettura). Poi crea un processo figlio e successivamente scrive nel file la stringa "finito" e si mette in attesa della terminazione del figlio, dopo la quale scrive al terminale "figlio ha letto il file" e termina la propria esecuzione. Il processo figlio accede al file e se legge la stringa "finito" termina la propria esecuzione con codice di ritorno 1. Se viceversa non trova la stringa continua a ripetere l'operazione di lettura . Pertanto il figlio per terminare la propria esecuzione deve aspettare che il padre abbia effettuato la scrittura della stringa nel file.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>


// si possono utilizzare le funzioni di lettura/scrittura dal file che si preferiscono.

int main( int argc, char ** argv )
{int status; pid_t pid; int cont; FILE *fp;char s[100];
 int fine;
 fp=fopen("miofile.txt", "w+");
 if (fp==NULL) exit(0);
 pid=fork();
 if  (pid==-1) exit(-1);
 if (pid==0)
    {printf("\nlfiglio inizia");fflush(stdout);//fflush per forzare scrittura a video
     fine=0;
     while (fine==0)
        {printf("\nlfiglio itera"); fflush(stdout);
         rewind(fp); //per controllare il file dall'inizio
          if  (!(fgets(s, 100, fp)==NULL))
              if(strcmp(s,"finito")==0){printf("figlio ha letto");
                                                fine=1;
                                          exit(1);
                                        }
        }
```

```
}
else
   {sleep(10); //test per vedere figlio che attende
    fputs("finito", fp); fflush(fp);
    printf("padre: scritto nel file");    fflush(stdout);
    wait(&status);
    printf("\nfiglio ha letto il file");
    exit(1);
 }

}
```