

Fondamenti di Informatica T-1 modulo 2

Laboratorio 10: ***Esempio di esame - Soluzioni***

1

Esercizio 1

2

Header funzionalità comuni (common.h/common.c)

```
#ifndef common
#define common
    #include <stdio.h>
    typedef enum{false, true} Boolean;
    int readField(char buffer[], char sep, FILE *f);
#endif
```

common.c implementa la readField come da lucidi.

3

Parte 1 – Strutture dati e funzionalità di agenda (agenda.h)

```
#ifndef agenda
#define agenda
    #define DIM_NAME 21
    #define MAX_EVENTS 20
    #include "common.h"

    typedef struct {
        int h;
        int m;
    } Time;
    typedef struct {
        Time startTime;
        Time endTime;
    } Event;
    ...
```

4

Parte 1 – Strutture dati e funzionalità di agenda (agenda.h)

```
...
typedef struct {
    int code;
    char name[DIM_NAME];
    Event events[MAX_EVENTS];
    int nEvents;//dimensione logica di events
} Worker;
//manipolazione tempi (Es. 1)
int timeDifference(Time t1, Time t2);
Time getTranslatedTime(Time t, int displacement);
//scrittura delle strutture dati a video
void printTime(Time t);
void printEvent(Event e);
void printWorker(Worker w);
...

```

5

Parte 1 – Strutture dati e funzionalità di agenda (agenda.h)

```
...
//scrittura delle strutture dati su file (di testo)
void writeTime(FILE* f, Time t);
void writeEvent(FILE* f, Event e);
void writeWorker(FILE* f, Worker w);
//lettura delle strutture dati da file (di testo) (Es. 2)
Boolean readTime(FILE* f, Time* t);
Boolean readEvent(FILE* f, Event* e);
Boolean readWorker(FILE* f, Worker* w);
//funzioni per l'inserimento ordinato di eventi (Es. 4)
int compare(Event e1, Event e2);
Boolean insert(Event* events, Event e1, int dim, int* elCount);
#endif

```

6

Parte 1 – Strutture dati e funzionalità di richiesta (request.h)

```
#ifndef req
#define req

typedef struct {
    int workerCode;
    int minutes;
} Request;

//stampa a video di una richiesta e di un vettore di richieste
void printRequest(Request r);
void printRequests(Request* reqs, int dim);
//lettura di un vettore di richieste da file binario (Es. 2)
Request* readRequests(char* fileName, int* dim);

#endif
```

7

Parte 1 – Implementazione funzioni sui tempi (agenda.c)

```
#include "agenda.h"
#include "math.h"

int timeDifference(Time t1, Time t2)
{
    return abs(t1.m - t2.m + (t1.h - t2.h)*60);
}

Time getTranslatedTime(Time t, int displacement)
{
    Time t2;
    t2.m = (t.m + displacement) % 60;
    t2.h = t.h + (t.m + displacement) / 60;
    return t2;
}
```

8

Parte 2 – Lista di Workers (agenda.h)

```
#include "agenda.h"  
typedef Worker Element;
```

9

Parte 2 – header gestione agende (agendaManagement.h)

```
#include "list.h"  
#include "request.h"  
//Stampa lista di workers a video  
void printWorkers(List l);  
//Lettura lista di workers da file di testo (Es. 2)  
List readWorkers(char* fileName);  
//Inserimento richiesta (Es. 3)  
Boolean insertNewEvent(Request r, List workers);  
// Gestione e stampa richieste (Es. 4)  
void handleRequests(List workers, Request* reqs, int dimReqs);  
void writeWorkers(char* fileName, List workers);
```

10

Parte 2 – lettura Time e Event (agenda.c)

```
Boolean readTime(FILE *f, Time* t)
{
    return fscanf(f, "%d:%d", &t->h, &t->m) == 2;
}
```

```
Boolean readEvent(FILE *f, Event* e)
{
    if(! readTime(f, &e->startTime))
        return false;
    if(! readTime(f, &e->endTime))
        return false;
    else
        return true;
}
```

11

Parte 2 – lettura singolo Worker (agenda.c)

```
Boolean readWorker(FILE *f, Worker* w)
{
    int i = 0;
    Boolean ok;
    if(fscanf(f, "%d", &w->code) != 1)
        return false;
    if(readField(w->name, ';', f) != ';')
        return false;
    do {
        ok = readEvent(f, &(w->events[i]));
        if(ok) i++;
    }
    while(ok && fgetc(f) == ';');
    w->nEvents = i;
    return true;
}
```

12

Parte 2 – lettura Workers (agendaManagement.c)

```
List readWorkers(char* fileName)
{
    List workers = emptyList();
    Worker w;
    FILE* f = fopen(fileName, "r");
    if(f == NULL)
        abort();
    while(readWorker(f, &w))
        workers = cons(w, workers);
    fclose(f);
    return workers;
}
```

13

Parte 2 – Stampa Time e Event (agenda.c)

```
void printTime(Time t)
{
    printf("%02d:%02d", t.h, t.m);
}

void printEvent(Event e)
{
    printTime(e.startTime);
    printf("-");
    printTime(e.endTime);
}
```

14

Parte 2 – Stampa singolo Worker (agenda.c)

```
void printWorker(Worker w)
{
    int i;
    printf("%d) %s", w.code, w.name);
    printf("\n\t");
    for(i = 0; i < w.nEvents; i++)
    {
        printf(" ");
        printEvent(w.events[i]);
    }
}
```

15

Parte 2 – Stampa Workers (agendaManagement.c)

```
void printWorkers(List l)
{
    while(!empty(l))
    {
        printWorker(head(l));
        printf("\n");
        l = tail(l);
    }
}
```

16

Parte 2 – Lettura richieste (request.c)

```
Request* readRequests(char* fileName, int* dim)
{
    FILE* f = fopen(fileName, "rb");
    Request* reqs;
    fread(dim, sizeof(int), 1, f);
    reqs = (Request *) malloc(*dim * sizeof(Request));
    fread(reqs, sizeof(Request), *dim, f);
    return reqs;
}
```

17

Parte 2 – Stampa richieste (request.c)

```
void printRequest(Request r) {
    printf("Worker: %d, amount: %d",
           r.workerCode, r.minutes);
}

void printRequests(Request* reqs, int dim) {
    int i;
    for(i = 0; i < dim; i++)
    {
        printRequest(reqs[i]);
        printf("\n");
    }
}
```

18

Parte 3 – Comparazione tra eventi (agenda.c)

```
//Comparazione tra eventi (per l'inserimento
ordinato). La comparazione è effettuata sui
tempi di inizio degli eventi
int compare(Event e1, Event e2)
{
    int comp = e1.startTime.h - e2.startTime.h;
    if(comp == 0)
        return e1.startTime.m - e2.startTime.m;
    else
        return comp;
}
```

19

Parte 3 – Inserimento ordinato di eventi (agenda.c)

```
Boolean insert(Event* events, Event e1, int dim, int* elCount)
{
    int i = 0, j;
    Boolean found = false;
    if(*elCount < dim)
    {
        while(i < *elCount && !found)
        {
            if(compare(e1, events[i]) < 0) ←
                found = true;
            else
                i++;
        }
        if(found)
            for(j = *elCount; j >= i; j--)
                events[j] = events[j-1];
            events[i] = e1;
            (*elCount)++;
            return true;
        }
    return false;
}
```

20

Parte 3 – Inserimento richiesta (agendaManagement.c)

```
Boolean insertNewEvent(Request r, List workers)
{
    int i = 0;
    Boolean slotFound = false;
    int freeTime;
    Time tEndPrec;
    Time tStartNext;
    Worker* w;
    tEndPrec.h = 9;
    tEndPrec.m = 0; // si parte dalle 9:00
    w = findWorker(r.workerCode, workers);
    if(w == NULL) //dipendente non trovato
        return false;
    ...
}
```

21

Parte 3 – Inserimento richiesta (agendaManagement.c)

```
...
while(i <= w->nEvents && !slotFound)
{
    //l'ultimo controllo considera le 18:00
    if(i == w->nEvents)
    {
        tStartNext.h = 18;
        tStartNext.m = 0;
    }
    else
        tStartNext = w->events[i].startTime;
    freeTime = timeDifference(tStartNext, tEndPrec);
    slotFound = freeTime >= r.minutes;
    ...
}
```

22

Parte 3 – Inserimento richiesta (agendaManagement.c)

```
...
if(slotFound)
{
    Event e;
    e.startTime = tEndPrec;
    e.endTime = getTranslatedTime(tEndPrec, r.minutes);
    insert(w->events, e, MAX_EVENTS, &w->nEvents);
}
if(i < w->nEvents)
    tEndPrec = w->events[i].endTime;
i++;
}
return slotFound;
}
```

23

Parte 4 – Gestione richieste (agendaManagement.c)

```
void handleRequests(List workers,
                    Request* reqs, int dimReqs)
{
    Boolean ok;
    int i;
    for(i = 0; i < dimReqs; i++)
    {
        ok = insertNewEvent(reqs[i], workers);
        if(!ok)
            printf("Il dipendente %d non ha tempo
                   sufficiente per gestire un impegno di
                   %d minuti\n",
                   reqs[i].workerCode, reqs[i].minutes);
    }
}
```

24

Parte 4 – Stampa Time e Event su file (agenda.c)

```
void writeTime(FILE* f, Time t)
{
    fprintf(f, "%02d:%02d", t.h, t.m);
}

void writeEvent(FILE* f, Event e)
{
    writeTime(f, e.startTime);
    fputc(' ', f);
    writeTime(f, e.endTime);
}
```

25

Parte 4 – Stampa singolo Worker su file (agenda.c)

```
void writeWorker(FILE* f, Worker w)
{
    int i;
    fprintf(f, "%d%s", w.code, w.name);
    for(i = 0; i < w.nEvents; i++)
    {
        fputc(';', f);
        writeEvent(f, w.events[i]);
    }
}
```

26

Parte 4 – Stampa Workers su file (agendaManagement.c)

```
//La stampa rispetta l'ordine del file di lettura!  
void writeWorkersRecursive(FILE* f, List workers)  
{  
    if(!empty(workers))  
    {  
        writeWorkersRecursive(f, tail(workers));  
        writeWorker(f, head(workers));  
        fputc('\n', f);  
    }  
}  
  
void writeWorkers(char* fileName, List workers)  
{  
    FILE* f = fopen(fileName, "w");  
    writeWorkersRecursive(f, workers);  
}
```

27

Esercizio 2

28

Parte 1 – Strutture dati e funzionalità ingredienti (listino.h)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#ifndef LISTINO
#define LISTINO
#define MAX 25
    typedef struct ingrediente
    {
        char nome[MAX];
        float prezzo;
    } Ingrediente;
//lista di ingredienti
typedef struct item_struct
{
    Ingrediente value;
    struct item_struct *next;
} item;
typedef item *list;
```

29

Parte 1 – Strutture dati e funzionalità ingredienti (listino.h)

```
//Lettura ingredienti (parte 1)
list leggiIngredienti(char *filename);
//Scrittura ingredienti (parte 1)
int scriviIngredienti(char *filename, list l);
//Ricerca prezzo (parte 2)
float trovaPrezzo(char *nome, list l);
//Aggiornamento prezzo (parte 2)
list aggiornaPrezzo(list l, char *nome, float nuovoPrezzo);

//Funzioni ADT lista (di ingredienti)
list emptyList();
int isEmpty(list l);
list cons(Ingrediente newValue, list l);
Ingrediente head(list l);
list tail(list l);
#endif
```

30

Parte 1 – Lettura ingredienti (listino.c)

- Nota: per l'implementazione delle funzionalità relative alla lista si consultino le slide

```
list leggiIngredienti(char *filename)
{
    FILE * fp;
    char nome[MAX];
    float prezzo;
    Ingrediente temp;
    list result = emptyList();
    if ((fp = fopen(filename, "r")) == NULL)
        return result;
    while (fscanf(fp, "%s %f", nome, &prezzo) == 2)
    {
        strcpy(temp.nome, nome);
        temp.prezzo = prezzo;
        result = cons(temp, result);
    }
    fclose(fp);
    return result;
}
```

31

Parte 1 – Scrittura ingredienti (listino.c)

```
int scriviIngredienti(char *filename, list l)
{
    FILE * fp;
    if ((fp = fopen(filename, "w")) == NULL)
    {
        return -1;
    }
    while (!isEmpty(l))
    {
        fprintf(fp, "%s %f\n", head(l).nome, head(l).prezzo);
        l = tail(l);
    }
    fclose(fp);
    return 0;
}
```

32

Parte 2 – Ricerca prezzo (listino.c)

```
float trovaPrezzo(char *nome, list l)
{
    while (!isEmpty(l))
    {
        if (strcmp(head(l).nome, nome) == 0)
        {
            return head(l).prezzo;
        }
        else
        {
            l = tail(l);
        }
    }
    return -1;
}
```

33

Parte 2 – Aggiornamento prezzo (listino.c)

```
list aggiornaPrezzo(list l, char *nome, float nuovoPrezzo)
{
    list temp = l;
    int trovato = 0;
    while (!isEmpty(temp) && !trovato)
    {
        if (strcmp(head(temp).nome, nome) == 0)
        {
            trovato = 1;
            temp->value.prezzo = nuovoPrezzo;
        }
        else
        {
            temp = tail(temp);
        }
    }
    return l;
}
```

34

Parte 3 – Strutture dati e funzionalità pizze (pizza.h)

```
#ifndef PIZZA
#define PIZZA

#include <stdio.h>
#include <string.h>
#include "listino.h"
typedef struct pizza
{
    char nomeCliente[65];
    int numeroIngredienti;
    Ingrediente topping[10];
} Pizza;

float calcolaPrezzo(Pizza p, list l); //Parte 3
Pizza leggiPizza(FILE * fp); //Supporto in parte 4

#endif
```

35

Parte 3 – Calcolo prezzo (pizza.c)

```
float calcolaPrezzo(Pizza p, list l)
{
    int i;
    float result = 0;
    float temp;
    for (i = 0; i < p.numeroIngredienti; i++)
    {
        temp = trovaPrezzo(p.topping[i].nome, l);
        if (temp < 0)
        {
            return -1;
        }
        result = result + temp;
    }
    return result;
}
```

36

Parte 4 – Funzionalità ordini (ordini.h)

```
#ifndef ORDINI
#define ORDINI

#include <stdio.h>
#include <string.h>
#include "pizza.h"

Pizza * leggiOrdini(char *filename, int *numPizze);
void naiveSortR(Pizza a[], int dim);

#endif
```

37

Parte 4 – Lettura singola pizza da file binario (pizza.c)

```
Pizza leggiPizza(FILE *fp)
{
    Pizza result;
    if (!feof(fp))
    {
        fread(&result, sizeof(Pizza), 1, fp);
    }
    return result;
}
```

38

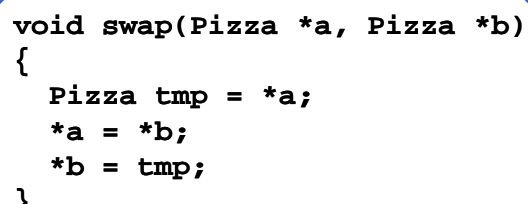
Parte 4 – Lettura ordini (ordini.c)

```
Pizza* leggiOrdini(char *filename, int *numPizze)
{
    FILE *fp;
    Pizza *result;
    int temp;
    *numPizze = 0;
    if ((fp=fopen(filename, "rb")) == NULL)
        return NULL;
    fread(&temp, sizeof(int), 1, fp);
    result = (Pizza *) malloc(sizeof(Pizza) * temp);
    while (*numPizze < temp)
    {
        result[*numPizze] = leggiPizza(fp);
        *numPizze = *numPizze + 1;
    }
    fclose(fp);
    return result;
}
```

39

Parte 4 – Ordinamento ordini con naïve sort (ordini.c)

```
void naiveSortR(Pizza a[], int dim)
{
    int i, posmin;
    Pizza min;
    if (dim == 1)
        return;
    for (posmin = 0, min = a[0], i = 1; i < dim; i++)
    {
        if (strcmp(a[i].nomeCliente, min.nomeCliente) < 0)
        {
            posmin = i;
            min = a[i];
        }
    }
    if (posmin != 0)
        swap(&a[0], &a[posmin]);
    naiveSortR(&a[1], dim - 1);
}
```



```
void swap(Pizza *a, Pizza *b)
{
    Pizza tmp = *a;
    *a = *b;
    *b = tmp;
}
```

40

Parte 5 – Calcolo spesa clienti (main.c)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "listino.h"
#include "pizza.h"
#include "ordini.h"

int main(void)
{
    list l;
    Pizza *p;
    int numPizze;
    int i;
    char *nome;
    float totale;
    l = leggiIngredienti("ingredienti.txt");
    l = aggiornaPrezzo(l, "salame_piccante", 2.00);
    scriviIngredienti("ingredienti.txt", l);
    ...
}
```

41

Parte 5 – Calcolo spesa clienti (main.c)

```
...
p = leggiOrdini("ordini.bin", &numPizze);
naiveSortR(p, numPizze);
for (i=0; i<numPizze; )
{
    totale = calcolaPrezzo(p[i], l);
    nome = p[i].nomeCliente;
    i++;
    while (strcmp(nome, p[i].nomeCliente) == 0)
    {
        totale = totale + calcolaPrezzo(p[i], l);
        i++;
    }
    printf("%s deve pagare %f euro\n", nome, totale);
}
system("PAUSE");
return (0);
}
```

42