SIMULAZIONE Fondamenti di Informatica T-1
Prova di Laboratorio - 14 Dicembre 2009
Compito A

Prima di cominciare: si scarichi il file **`StartKit0A.zip`** contenente i file di esempio. *(Qualora non fosse disponibile lo startkit, creare i files di testo/binari opportuni al fine di verificare il programma)*
Avvertenze per la consegna: nominare i file sorgenti come richiesto nel testo del compito, apporre all'inizio di ogni file sorgente un commento contenente i propri dati (**cognome, nome, numero di matricola**) e il **numero** della prova d'esame. Al termine, **consegnare tutti i file sorgente** ed i file contenuti nello StartKit.
Rispettare le specifiche, in particolare inserire le funzioni nei file specificati fra parentesi dopo il nome della funzione. Chi non rispetta le specifiche sarà opportunamente penalizzato. **NON SARANNO CORRETTI** gli elaborati che presenteranno un numero "non ragionevole" di errori di compilazione.
Consiglio: per verificare l'assenza di *warnings*, effettuare di tanto in tanto un *Rebuild All*.

Gli ***impegni giornalieri*** dei dipendenti di un'azienda devono essere aggiornati con una serie di nuove richieste.
Gli ***impegni già fissati sono memorizzati in un file di testo***, all'interno del quale ogni riga rappresenta gli impegni di un dipendente, secondo il seguente formato:

- Codice numerico del dipendente (intero)
- Nome del dipendente (al più di 20 caratteri), seguito da un ';'
- Sequenza di impegni del dipendente, costituita da
  - Ora di inizio dell'impegno, con formato HH:MM (HH e MM interi), seguita da uno spazio
  - Ora di fine dell'impegno, con formato HH:MM, seguita da un ';' (a meno che non si tratti dell'ultimo impegno, in questo caso c'è un fine linea)

Si fissi come ***ipotesi*** che la ***sequenza di impegni sia ordinata rispetto al tempo***, e che ogni lavoratore abbia un massimo di 20 impegni. Nota: ogni impegno ha un orario di inizio superiore o uguale alle 09:00 e un orario di fine inferiore o uguale alle 18:00

Un secondo ***file binario*** contiene le informazioni relative ad ***ulteriori richieste di impegni da eseguire (possibilmente) in giornata***. Il primo campo del file riporta il numero di richieste contenute, seguito dalle varie richieste. Ogni richiesta è costituita dal ***codice del dipendente*** a cui si riferisce e da un intero rappresentate i ***minuti da dedicare*** per l'impegno richiesto.

Esercizio 1 - Strutture dati e relative operazioni

Definire delle strutture atte a contenere i dati relativi a dipendenti, impegni e riunioni:

- Il tipo **event** deve contenere i dati di un impegno (orario di inizio e di fine) sfruttando una seconda struttura dati chiamata **time** (contenente ore e minuti).
- Il tipo **worker** contiene i dati relativi ad un dipendente: codice numerico, nome e sequenza di impegni da modellare come un vettore di **event *allocato staticamente***
  Nota: ricordarsi che servirà memorizzare anche la dimensione logica di tale vettore
- Infine, il tipo **request** memorizza i dati di una richiesta di impegno, ovvero codice del dipendente e numero di minuti.

Dichiarare e definire le seguenti funzioni:

- **timeDifference** deve prendere in ingresso due orari e restituirne la differenza in minuti (valore assoluto, ricordandosi di effettuare le opportune conversioni minuti/ore… la differenza tra 10:20 e 8:50 è pari a 90 minuti)
- **getTranslatedTime** deve prendere in ingresso un orario e un valore in minuti e restituire l'orario che si ottiene aggiungendo i minuti dati all'orario (ricordarsi di effettuare le opportune conversioni minuti/ore)

SIMULAZIONE Fondamenti di Informatica T-1
Prova di Laboratorio - 14 Dicembre 2009
Compito A

Esercizio 2 - Lettura dai file

Realizzare una funzione **readWorkers** che prenda in input il nome del file di testo contenente i dati dei dipendenti, lo apra in lettura e ***legga i dati di tutti i dipendenti memorizzandoli in una lista di worker***
Realizzare una funzione **readRequests** che prenda in input il nome del file binario contenente i dati delle richieste, lo apra in lettura e legga i dati di tutte le richieste memorizzandoli in un ***vettore di meeting allocato dinamicamente della dimensione strettamente necessaria***. Si ricorda che il numero di richieste è memorizzato come primo campo del file.
Realizzare due funzioni di stampa verificando la corretta lettura dei dati.

Esercizio 3 - Inserimento richiesta

Realizzare una funzione **insertNewEvent** che prenda in input una richiesta di impegno e la lista di dipendenti, ottenendo il dipendente coinvolto e cercando di inserire la richiesta nella sua agenda. Si adotti la seguente politica per l'inserimento:

- la richiesta deve essere inserita ***prima possibile tra i vari impegni già presi dal dipendente*** (ovvero appena il dipendente ha un tempo libero capace di contenere il nuovo impegno), senza ovviamente travalicare i limiti della giornata lavorativa (09:00 – 18:00); al fine di individuare l'ammontare dei tempi liberi, si utilizzi la funzione **timeDifference** sviluppata al punto 1.
- se è possibile inserire la richiesta in agenda, la funzione si deve incaricare di creare effettivamente il nuovo impegno (l'impegno partirà o alle 09:00 o al tempo di fine dell'impegno immediatamente precedente) e di inserirlo nel vettore degli impegni del dipendente sfruttando **l'inserimento ordinato**.

La funzione deve restituire un booleano che attesti se l'inserimento è stato effettuato o meno.

Parte 4 - Gestione delle richieste

Realizzare una funzione **handleRequests** che prenda in input una lista di dipendenti e un vettore di richieste, cercando di inserirle nelle varie agende. A tal fine, si scandisca il vettore di richieste, utilizzando la funzione **insertNewEvent** per l'inserimento. Se la richiesta non è stata inserita, si stampi a video un messaggio di errore del tipo "il dipendente XXX non ha tempo sufficiente per gestire un impegno di YYY minuti".
Realizzare una funzione **printWorkers** che stampi su un file di testo una lista di dipendenti e relativi impegni, con lo stesso formato utilizzato in lettura.

Nel main, si utilizzino le funzioni sviluppate per acquisire i dati in input, gestire le richieste e stampare su file la lista dei dipendenti dopo l'aggiornamento.

SIMULAZIONE Fondamenti di Informatica T-1
Prova di Laboratorio - 14 Dicembre 2009
Compito A

"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.

SIMULAZIONE Fondamenti di Informatica T-1
Prova di Laboratorio - 14 Dicembre 2009
Compito A

"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;

	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);

	//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 el, int dim, int* elCount);
#endif
```

SIMULAZIONE Fondamenti di Informatica T-1
Prova di Laboratorio - 14 Dicembre 2009
Compito A

"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;
}

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

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

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]);
	}
}

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);
}

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]);
```

```
	}
}

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;
}

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;
}

//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;
}

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

```
        }
        return false;
}
```

SIMULAZIONE Fondamenti di Informatica T-1
Prova di Laboratorio - 14 Dicembre 2009
Compito A

"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
```

SIMULAZIONE Fondamenti di Informatica T-1
Prova di Laboratorio - 14 Dicembre 2009
Compito A

"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");
	}
}

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;
}
```

SIMULAZIONE Fondamenti di Informatica T-1
Prova di Laboratorio - 14 Dicembre 2009
Compito A

“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);
```

SIMULAZIONE Fondamenti di Informatica T-1
Prova di Laboratorio - 14 Dicembre 2009
Compito A

"agendaManagement.c":

```
#include "agendaManagement.h"

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;
}

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

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;
	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;
		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;
}
```

SIMULAZIONE Fondamenti di Informatica T-1
Prova di Laboratorio - 14 Dicembre 2009
Compito A

```
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);
	}
}

//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);
}
```

SIMULAZIONE Fondamenti di Informatica T-1
Prova di Laboratorio - 14 Dicembre 2009
Compito A

“element.h”:

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