Sistemi di Calcolo (A.A. 2016-2017)
Corso di Laurea in Ingegneria Informatica e Automatica
Sapienza Università di Roma

B

Compito di esonero – Durata 1h 30'

Inserire nome, cognome e matricola nel file `studente.txt`.

Parte 1 (programmazione IA32)

Si traduca in assembly IA32 la seguente funzione C scrivendo un modulo `es1B.s`:

```
int media(int, int);
int test(const char* x, const char* y) {
    return media(*x,*y) <= *x || *y <= media(*x,*y);
}
```

L'unico criterio di valutazione è la correttezza, cioè l'equivalenza semantica tra il programma tradotto e quello C di partenza. Per i test, usare il programma di prova `es1B-main.c` e il modulo `es1B-media.s`, entrambi inclusi.

Generare un file eseguibile `es1B` compilato con `gcc –m32`.

Parte 2 (programmazione IA32)

Si traduca in assembly IA32 la seguente funzione C scrivendo un modulo `es2B.s`:

```
void concatReverse(char* s1, char* s2, char* dest,
                   int len_s1, int len_s2) {
    while (*s1) *dest++ = *s1++;
    int i = 0;
    for (; i < len_s2; i++) *dest++ = s2[len_s2-i-1];
    *dest = 0;
}
```

L'unico criterio di valutazione è la correttezza, cioè l'equivalenza semantica tra il programma tradotto e quello C di partenza. Per i test, usare il programma di prova `es2B-main.c` incluso.

Generare un file eseguibile `es2B` compilato con `gcc –m32`.

Parte 3 (ottimizzazione work)

Si crei nel file `lista-opt.c` una versione **ottimizzata** del seguente modulo `lista.c`:

```
#include <stdlib.h>
#include "lista.h"

typedef struct nodo nodo;

struct nodo { // nodo lista
    int   elem;
    nodo* next;
};

struct lista {
    nodo* first; // lrimo nodo della lista
};

lista* lista_new(){ // crea lista vuota
    return calloc(1, sizeof(lista));
```

```
}

void lista_addLast(lista* l, int x){ // aggiunge in coda
    nodo* last = l->first;
    if (last != NULL)
        while (last->next != NULL) last = last->next;
    nodo* n = malloc(sizeof(nodo));
    n->elem = x;
    n->next = NULL;
    if (l->first == NULL) l->first = n;
    else last->next = n;
}

int lista_removeFirst(lista* l, int* x){ // rimuove dalla testa
    if (l->first == NULL) return -1;
    nodo* dead = l->first;
    if (x != NULL) *x = dead->elem;
    l->first = dead->next;
    free(dead);
    return 0;
}

void lista_del(lista* l){ // dealloca lista
    while (l->first != NULL) lista_removeFirst(l, NULL);
    free(l);
}
```

Il modulo implementa un **tipo di dato lista utilizzando una lista collegata**.

Compilare due versioni del programma, usando `gcc` **a 32 bit** con livello di ottimizzazione 1 e lo stesso modulo `main.c`:

1. non ottimizzata manualmente: eseguibile `lista`;
2. ottimizzata manualmente: eseguibile `lista-opt`.

Ai fini dell'ottimizzazione:

1. usare `gprof` per identificare le porzioni più onerose computazionalmente. Per evitare confusione, chiamare l'eseguibile usato per la profilazione `lista-pg` e il report del profiler `lista-pg.txt`;
2. esaminare il modulo `lista.s` generato a partire da `lista.c` con `gcc –S -O1` (già fornito) per capire quali ottimizzazioni siano già state effettuate dal compilatore.

Rispondere alle seguenti domande:

1. Descrivere le ottimizzazioni applicate **manualmente** e dire perché si ritiene che siano efficaci.
2. Riportare la media dei tempi di esecuzione (real) di `lista` e di `lista-opt` su tre run usando il comando `time` e dire di quante volte è più veloce l'**eseguibile** `lista-opt` rispetto a `lista` (speedup).
3. Riportare il flat profile del programma `lista` usando `gprof`.

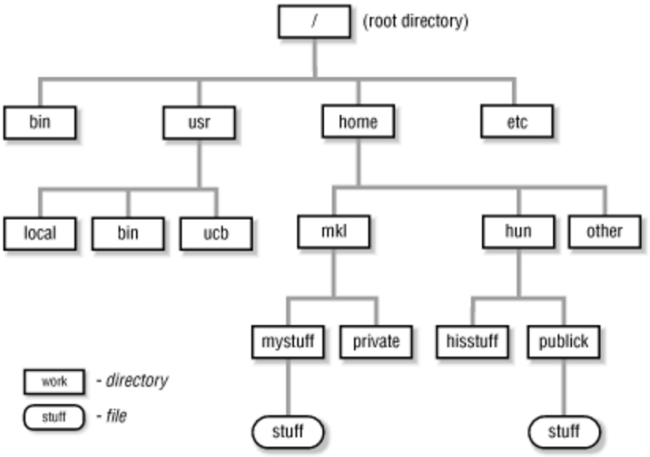
Inserire le risposte nel file `es3B.txt`. Alla fine del compito, **non cancellare** `gmon.out` e gli altri eseguibili creati.

Parte 4 (quiz)

Si risponda ai seguenti quiz, inserendo le risposte (A, B, C, D o E per ogni domanda) nel file `es4B.txt`. **Una sola risposta è quella giusta**. Rispondere E equivale a non rispondere (0 punti).

Domanda 1 (file system)

Si consideri la seguente struttura di directory:



Assumendo di essere l'utente `hun` e che la directory corrente sia `hisstuff` quale se seguenti comandi risulta valido:

A	`cd ~/home/hun/../`	**B**	`ls ../~/hisstuff`
C	`cat ~/hisstuff/../publick/./stuff`	**D**	`rm /home/hun/./hisstuff/../publick/`

Motivare la risposta nel file `M1.txt`. **Risposte non motivate saranno considerate nulle.**

Domanda 2 (parametri main)

Se un programma C viene compilato (producendo il binario `prog`) ed eseguito utilizzando il comando `./prog abc 1 def`, quale delle seguenti espressioni booleane risulta vera nel momento in cui inizia l'esecuzione della sua funziona main:

A	`argc <= 3`	**B**	`strcmp(argv[0], "abc") == 0`
C	`argv[2] == 1`	**D**	`strcmp(argv[3], "def") == 0`

Motivare la risposta nel file `M2.txt`. **Risposte non motivate saranno considerate nulle.**

Domanda 3 (analisi delle prestazioni del software)

Dato un programma con tre funzioni A, B e C, esse occupano rispettivamente il 60%, 20% e 20% del tempo di esecuzione. Sapendo che C è invocata soltanto da B, il compilatore può fare inlining di C in B ed ottimizzare la nuova versione di B. Qual è lo speedup massimo ottenibile per il nuovo programma se A rimane invariata?

A	1.33x	**B**	1.50x
C	1.67x	**D**	2.00x

Motivare la risposta nel file `M3.txt`. **Risposte non motivate saranno considerate nulle.**