ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Licenciatura em Engenharia Informática e de Computadores

Programação em Sistemas Computacionais

Teste Global, 27 de Janeiro de 2011

Considere em todas as questões a utilização de uma arquitectura a 32 bit *little-endian*.

1. [2]

a) [1] Considerando o programa no ficheiro fonte `macro.c`, descreva o resultado da sua execução, justificando os valores apresentados.

b) [1] Uma cache associativa de 4 vias (*ways*) com 16 K bytes e 32 bits de endereço físico usa 5 bits para offset no bloco. Quantos bits são usados para a *tag*? Justifique os cálculos apresentados.

macro.c
```
int a[3] = {0x040201,0x08040201};
int main() {
  void *p;
#define M(T) for(p=a ; *(T*)p ; p=((T*)p)+1) \
          printf("%X ",*(T*)p); putchar('\n')
  M(char); M(short); M(int);
  return 0;
}
```

2. [4] Considere os ficheiros `a.c`, `b.c`, `p.s` e `makefile`.

a) [1] Quais os ficheiros gerados com o comando `make`?

b) [1] Indique os símbolos indefinidos do módulo `a.o`.

c) [1] Descreva, justificando, o *output* do programa `psc`.

d) [1] Altere a função `f` implementada em `b.c` para fechar a biblioteca dinâmica antes de retornar.

a.c
```
#include <stdio.h>
extern int x;
int f(int);
static int y = 10;

int main() {
  int z=x+y;
  printf("%d\n",f(z));
  return 0;
}
```

makefile
```
psc: a.o b.o p.so
  gcc -ldl -o $@ a.o b.o
p.so: p.o
  gcc -shared -o $@ $^
%.o: %.c
  gcc -c $^
%.o: %.s
  gcc -c $^
```

b.c
```
#include <dlfcn.h>
int x=10;
typedef int (*Pf)(int*);
int f(int a) {
  void * l= dlopen("./p.so",RTLD_NOW);
  return ((Pf)dlsym(l,"p"))(&a);
}
```

p.s
```
.intel_syntax noprefix
.globl p
p:
  mov   eax, [esp+4]
  mov   eax, [eax]
  inc   eax
  ret
```

3. [8] Um evento é uma entidade que anuncia (dispara) uma determinada ocorrência. No evento podem-se registar funções interessadas na ocorrência (*listeners*). No disparo do evento são invocadas todas as funções previamente registadas (por ordem de registo). Para implementar um sistema de eventos foram definidos os tipos e funções presentes em `events.h`. O programa apresentado em `tevents.c` é um teste ao sistema de eventos cujo resultado é apresentado. O módulo `events.c` implementa parcialmente as funções do sistema de eventos.

tevents.c
```
#include "events.h"

Event tempEvent;

static void l1(char *evtName, void *arg) {
  int temp = *(int *) (arg);
  printf("l1, %s:%d degrees\n", evtName, temp);
}
static void l2(char *evtName, void *arg) {
  int temp = *(int *) (arg);
  printf("l2, %s:%d degrees\n", evtName, temp);
}
static void simulateTemperatureEvent(int temp) {
  triggerEvent(&tempEvent, &temp);
}

int main() {
  createEvent(&tempEvent, "tempEvent");
  registListener(&tempEvent, l1);
  registListener(&tempEvent, l2);
  simulateTemperatureEvent(26);
  clearListeners(&tempEvent);
  simulateTemperatureEvent(24);
  return 0;
}
```

```
l1, tempEvent:26 degrees
l2, tempEvent:26 degrees
```

events.h
```
/* especifica a assinatura de listeners */
typedef void (*Listener)(char *eventName, void *arg);

typedef struct EventListener {
  Listener func;
  struct EventListener *previous;
} EventListener;

typedef struct Event {
  char name[32]; /* nome do evento */
  int chainSize; /* dimensão da cadeia de listeners*/
  EventListener *chain; /* listeners registados*/
} Event;

void registListener(Event *evt, Listener fp);
Listener *getListeners(Event *evt, int *size);
void triggerEvent(Event *evt, void *arg);
void createEvent(Event *evt, char *name);
void clearListeners(Event *evt);
```

a) [1,5] Implemente em IA-32 a função **`simulateTemperatureEvent`** que já está implementada no ficheiro `tevents.c`.

b) [2] Implemente em C a função **`clearListeners`** que coloca no estado inicial (sem *listeners*) o evento passado por parâmetro.

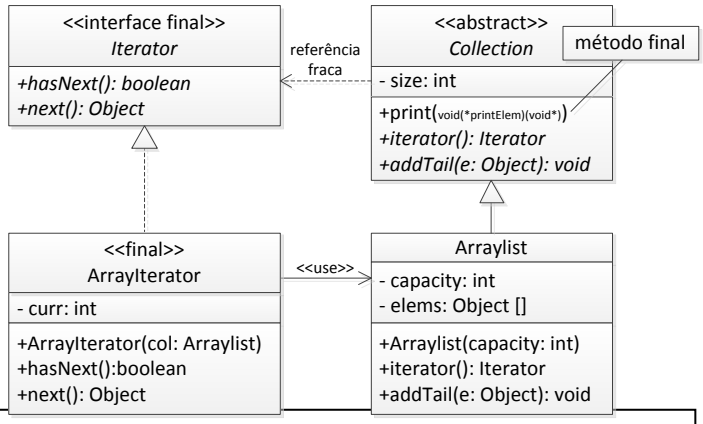
events.c
```
void createEvent(Event *evt, char *name) {
  strcpy(evt->name, name);  evt->chainSize=0;  evt->chain = NULL;
}

void registListener(Event *evt, Listener fp) {
  EventListener *el =(EventListener *) malloc(sizeof(EventListener));
  el->previous = evt->chain; evt->chain = el;
  el->func = fp; evt->chainSize++;
}

void invokeListener(char *evtName, EventListener *el, void *arg) {
  if (el->previous!=NULL) invokeListener(evtName, el->previous, arg);
  el->func(eventName, arg);
}

void triggerEvent(Event *evt, void *arg) {
  if (evt->chain != NULL) invokeListener(evt->name, evt->chain, arg);
}

void clearListeners(Event *evt) { … }
Listener *getListeners(Event *evt, int *size) { … }
```

c) [2] Implemente em IA-32 a função **`invokeListener`** que já está implementada no ficheiro `events.c`.

d) [2,5] Realize a função **`Listener *getListeners(Event *evt, int *size)`** que retorna um *array*, alocado dinamicamente com a dimensão mínima necessária, com todos os *listeners* registados no evento referido por **`evt`**. O número de *listeners* registados é colocado no inteiro referido por **`size`**.

4. [6] Considere o código apresentado a seguir e o diagrama de classes Java:

O diagrama de classes implementa uma hierarquia de colecções iteráveis. O código corresponde a uma transformação da hierarquia `Collection` em linguagem C e a uma aplicação `app.c` que usa a colecção `Arraylist`.
Nota: o código de pré-processamento `#include` e `#ifndef` não foi incluído para efeitos de legibilidade.



arraylist.h

```
typedef struct {
  Collection super;
  int capacity;
  void ** elems;
} Arraylist;
void initArraylist(Arraylist *this, int capacity);
```

arraylist.c

```
/* criação, iniciação de um ArrayIterator */
Iterator *iteratorAL(Arraylist *this) {…}
/* adição de elem no fim da colecção this */
void addTailAL(Arraylist *this, void *elem) {…}

static CollectionVTable vtbl = {
  (IteratorFP)iteratorAL,
  (AddTailFP)addTailAL
};
void initArraylist(Arraylist *this, int cap) {
  initCollection(&this->super);
  this->super.vptr = &vtbl;
  this->capacity = cap;
  this->elems = (void**)malloc(sizeof(void*)*cap);
}
```

collection.h

```
typedef struct _collection Collection;
typedef Iterator* (*IteratorFP)(Collection *this);
typedef void (*AddTailFP) (Collection *this, void *elem);
typedef struct {
  IteratorFP iterator;
  AddTailFP addTail;
} CollectionVTable;
struct _collection {
  CollectionVTable * vptr;
  int size;
};
void initCollection(Collection *this);
typedef void (*PrintFP) (void *elem);
void printCollection(Collection *this, PrintFP printElem);
```

collection.c

```
void initCollection(Collection *this) { this->size = 0;}

void printCollection(Collection *this, PrintFP prtElem) {
  Iterator * it = this->vptr->iterator(this);
  while (it->vptr.hasNext(it)) prtElem(it->vptr.next(it));
  free(it);
}
```

app.c

```
const char * words[]= {"word 1","word 2","word 3"};

Arraylist al;
Collection * create_collection() {
  initArraylist(&al, 10); return (Collection*)&al;
}

void print_string(const char * e) { printf("%s\n", e); }
int main() {
  int i, n = sizeof(words)/sizeof(words[0]);
  Collection *c;
  c = create_collection();
  for (i = 0; i < n; i++)
    c->vptr->addTail(c, (void*)words[i]);
  printCollection(c, (PrintFP)print_string);
  return 0;
}
```

a) [4] Realize em C uma versão equivalente da hierarquia **`Iterator`** indicando os módulos que criar. Implemente igualmente a função **`iteratorAL`** do módulo `arraylist.c` que cria, inicia e retorna um objecto `ArrayIterator`. Tome em atenção o código sublinhado na função `printCollection` do módulo `collection.c` que obriga a que a tabela virtual fique no espaço de memória de cada objecto que implemente `Iterator`.

b) [2] Actualize os módulos da hierarquia `Collection` e da aplicação `app.c` de forma a libertar os recursos alocados dinamicamente no processo de criação da colecção antes de retornar da aplicação.
Indique os módulos onde realiza as actualizações.

Duração: 2 horas e 30 minutos
Bom teste!