Práctica 2 - Manejo de estructuras de datos y punteros

Organización del Computador 2

1er Cuatrimestre 2017

1. Estructuras estáticas: Vectores y Matrices

Ejercicio 1

Para cada uno de los siguientes ítems, indicar el valor en bytes del desplazamiento en memoria (*offset*) requerido para direccionar sucesivamente los elementos del vector en cuestión. Indique, para cada uno, al menos 3 formas de acceder al n-ésimo elemento del vector.

a) vector de enteros de 8 bits sin signo.
b) vector de enteros de 16 bits con signo.
c) vector de enteros de 32 bits sin signo.
d) vector de enteros de 64 bits sin signo.

Ejercicio 2

Escriba una función en lenguaje ensamblador que devuelva el máximo elemento de un vector de números de 64 bits con signo de dimensión n (n de 16 bits sin signo), cuyo prototipo sea:

```
long maximo(long *v, unsigned short n);
```

Ejercicio 3

Escribir una función en lenguaje ensamblador que, dado un vector v de números de 64 bits sin signo y de dimensión n, calcule la suma de todos los elementos de v. Puede asumirse que esta suma no supera los 64 bits. El prototipo de la función es el siguiente:

```
long sumarTodos(long *v, unsigned short n);
```

¿En qué cambiaría el código si el vector contuviese números de 32 bits?

Ejercicio 4

Escriba una función en lenguaje ensamblador que calcule el producto entre un vector v de números de 64 bits con signo de dimensión n (n de 16 bits sin signo) y un escalar k de 32 bits con signo. El resultado debe devolverse en el vector w. El prototipo de la función es el siguiente:

```
void productoEscalar(long* v, int k, unsigned short n, superlong* w);
```

Ejercicio 5

Escriba una función en lenguaje ensamblador que:

a) Sume dos vectores de números de 64 bits con signo de dimensión n (n de 16 bits sin signo), almacenando el resultado en el primer vector, cuyo prototipo sea:
   `void suma(long *v1, long *v2, unsigned short n);`
b) Idem ejercicio anterior, pero con números de 128 bits.

Ejercicio 6

Escriba una función en lenguaje ensamblador que calcule el producto interno entre dos vectores de números de 32 bits sin signo de dimensión n (n de 16 bits sin signo), cuyo prototipo sea:

```
unsigned long productoInterno(unsigned int *v1, unsigned int *v2,
unsigned short n);
```

Ejercicio 7

Sea M_1 una matriz de $n \times m$ números de 64 bits sin signo almacenada por filas y M_2 otra matriz de $n \times m$ números de 64 bits sin signo pero almacenada por columnas.

a) ¿Qué offset se debería sumar a la dirección de memoria de M_i si pretendemos direccionar el primer elemento de la j-ésima columna? ($1 \leq i \leq 2$ y $0 \leq j < m$)
b) ¿Qué offset se debería sumar a la dirección de memoria de M_i si pretendemos direccionar el primer elemento de la j-ésima fila? ($1 \leq i \leq 2$ y $0 \leq j < n$)
c) Generalizar las respuestas de los ítems previos para direccionar el k-ésimo elemento de la respectiva fila o columna.

Ejercicio 8

Se tiene una matriz M que almacena números de 64 bits con signo, cuya dimensión es $n \times n$ (n de 16 bits sin signo). Escriba en lenguaje ensamblador:

a) Una función que indique si la matriz M es simétrica. El prototipo de la función es el siguiente:
   `long esSimetrica(long *M, unsigned short n);`
b) Una función que indique si la suma de los elementos de la diagonal principal es igual a la suma de elementos de la diagonal traspuesta. El prototipo de la función es el siguiente:
   `long diagonalesIguales(long *M, unsigned short n);`
c) Una función que indique si la matriz M es diagonal dominante. Formalmente, se dice que la matriz M de dimensión $n \times n$ es diagonal dominante cuando se satisface,

$$|m_{i,i}| \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} |m_{i,j}| \quad \forall i = 1 \ldots n$$

   El prototipo de la función es el siguiente:
   `long diagonalDominante(long *M, unsigned short n);`

Ejercicio 9

Escriba una función en lenguaje ensamblador que sume dos matrices de números de 64 bits con signo de dimensión $n \times n$ (n de 32 bits sin signo), almacenando el resultado en la primera matriz, cuyo prototipo sea:

```
void suma(long *A1, long *A2, unsigned int n);
```

Ejercicio 10

Escriba una función en lenguaje ensamblador que dada una matriz A de números de 64 bits con signo de dimensión $n \times n$ (n de 16 bits sin signo), devuelva otra matriz B, cuyos elementos sean el promedio de los vecinos de cada elemento de la matriz A.
Es decir:

$$b_{i,j} = \frac{a_{i,j-1} \ + \ a_{i,j+1} \ + \ a_{i-1,j} \ + \ a_{i+1,j}}{4}$$

Determinar el rango en el cual se puede realizar esta cuenta, para evitar indefiniciones. Fuera del rango la matriz resultante debe ser igual a la matriz original ($b_{i,j} = a_{i,j}$).

El prototipo de la función es el siguiente:
`void promedioVecinos(long *A, unsigned short n, long *B);`

Ejercicio 11

Escriba una función en lenguaje ensamblador que multiplique una matriz A de números de 16 bits sin signo de dimensión $m \times n$ (m y n de 16 bits sin signo) por un vector v de números de 16 bits sin signo de dimensión n. El resultado debe devolverse en el vector w.
El prototipo de la función es el siguiente:
`void productoMatrizVector(unsigned short *A, unsigned short *v,`
`unsigned short n, unsigned short m, unsigned int *w);`

Ejercicio 12

Escriba una función en lenguaje ensamblador que dada una matriz de números de 64 bits con signo de dimensión $n \times n$ (n de 16 bits sin signo) retorne la norma uno de la misma. Recordemos que:

$$||A||_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |a_{i,j}|$$

El prototipo de la función es el siguiente:
`superlong normaUno(long *A, unsigned short n);`

2. Estructuras dinámicas

Ejercicio 13

Supongamos que contamos con tres vectores distintos tales que cada uno de ellos contiene respectivamente instancias de las siguientes estructuras:

```
struct A {
        int n;
        int m;
        struct A *s;
} __attribute__((packed));


struct B {
        unsigned int n;
        unsigned short v[3];
        struct B *s;
} __attribute__((packed));


struct C {
        unsigned char c;
        struct C *v[10];
} __attribute__((packed));
```

a) Indicar en cada caso el offset requerido para direccionar sucesivamente los elementos del vector.
b) Indicar, para cada `struct`, los offsets de cada uno de sus miembros.
c) Volver al ejercicio 5 y pensar cómo sería la respuesta al ítem c) si se dispusiera de tres matrices de $n \times m$ conteniendo respectivamente instancias de dichas estructuras.

Ejercicio 14

Dado un nuevo tipo de datos llamado `lista` para manejar enteros con signo, cuya definición en C es la siguiente:

```
typedef struct nodo_t {
        long    dato;                   /* dato del nodo (un entero) */
        struct  nodo_t *prox;           /* puntero al siguiente */
} nodo;

typedef struct lista_t {
        nodo*   primero;                /* puntero al primer nodo */
} lista;
```

a) Escriba una función en lenguaje ensamblador que dada una lista y un entero i devuelva el i-ésimo elemento de la lista. El prototipo de la función es el siguiente:
`int iesimo(lista l, unsigned long i);`

b) Escriba una función en lenguaje ensamblador que dada una lista y un entero n devuelva la posición en la que se encuentra dicho entero en la lista. El prototipo de la función es el siguiente:
`unsigned int buscar(lista l, long n);`

c) Escriba una función en lenguaje ensamblador que dada una lista y un entero n lo agregue al comienzo de la lista. El prototipo de la función es el siguiente:
`void agregarAd(lista* l, long n);`

d) Escriba una función en lenguaje ensamblador que dada una lista y un entero n lo agregue al final de la lista. El prototipo de la función es el siguiente:
`void agregarAt(lista* l, long n);`

e) Escriba una función en lenguaje ensamblador que dada una lista ordenada en forma ascendente y un entero n lo agregue de forma que la lista se mantenga ordenada. El prototipo de la función es el siguiente:
`void agregarOrd(lista* l, long n);`

f) Escriba una función en lenguaje ensamblador que dada una lista y un entero n borre la primera aparición de n en la lista. El prototipo de la función es el siguiente:
`void eliminar(lista* l, long n);`

g) Escriba una función en lenguaje ensamblador que dada una lista y un entero n borre todas las apariciones de n en la lista. El prototipo de la función es el siguiente:
`void eliminarApariciones(lista* l, long n);`

h) Escriba una función en lenguaje ensamblador que dada una lista elimine los elementos repetidos. El prototipo de la función es el siguiente:
`void eliminarRepetidos(lista* l);`

i) Escriba una función en lenguaje ensamblador que elimine todos los nodos de dicha lista. El prototipo de la función es el siguiente:
`void eliminarLista(lista* l);`

j) Escriba una función en lenguaje ensamblador que dadas dos listas ordenadas, agregue en la primera los elementos de la segunda de forma ordenada. El prototipo de la función es el siguiente:
`void agregarOrdenado(lista* l, lista l1);`

Ejercicio 15

Consideremos una variante de las listas introducidas en el ejercicio anterior en la cual el valor almacenado en cada nodo pasa a ser, a su vez, una lista de strings:

```
typedef struct _nodo_t {
        char                    *str;
        struct nodo_t           *prox;
} nodo;

typedef struct lista_t {
        nodo    *primero;
} lista;
```

```
typedef struct _nodo_lista_de_lista_t {
        lista                           *dato;
        struct _nodo_lista_de_lista_t   *prox;
} nodo_lista_de_lista;

typedef struct _lista_de_lista_t {
        nodo_lista_de_lista     *primero;
} lista_de_lista;
```

Si interpretamos a cada string como una palabra, podemos imaginar que una lista de strings representa una oración y, a su vez, una secuencia de oraciones representa un párrafo. El objetivo de este ejercicio es imprimir por pantalla el párrafo denotado por una `lista_de_lista` dada. Tener en cuenta que se permite utilizar la función `printf` a lo largo de todo el ejercicio.

a) Escribir una función en lenguaje ensamblador que, dado un puntero a `nodo`, imprima por pantalla la palabra allí contenida. El prototipo de la función es el siguiente:
`void imprimirPalabra(nodo* node);`

b) Escribir una función en lenguaje ensamblador que, dado un puntero a `lista`, imprima por pantalla la oración que ésta representa. Considerar que entre una palabra y otra de una oración hay que agregar un espacio "␣". El prototipo de la función es el siguiente:
`void imprimirOracion(lista* list);`

c) Escribir una función en lenguaje ensamblador que, dado un puntero a `lista_de_lista`, imprima por pantalla el párrafo que ésta representa.
Considerar que entre una oración y otra hay que agregar un punto seguido por un espacio ".␣" y al final del párrafo hay que imprimir un punto y aparte ".\n".
El prototipo de la función es el siguiente:
`void imprimirParrafo(lista_de_lista* l);`

Ejercicio 16

Dado el siguiente tipo de datos `dicc` (diccionario sobre arbol binario) para manejar claves y significados, se pide escribir las funciones descriptas a continuación:

```
typedef struct _nodo_t {
        char                    *clave;
        char                    *significado;
        struct nodo_t   *izq;   /* puntero al arbol izquierdo */
        struct nodo_t   *der;   /* puntero al arbol derecho */
} nodo;

typedef struct _dicc_arb_bin_t {
        nodo    *raiz;
} dicc_arb_bin;
```

Invariante de representación: el árbol binario siempre esta ordenado (para todo nodo vale que las claves en el árbol izquierdo son menores a la clave del nodo actual; y las claves en el árbol derecho son mayores a la clave del nodo actual).

Se supone dada la función `int esMayor(char* clave1, char* clave2)` que devuelve 1 si clave1 $\geq$ clave2 y 0 si no (si clave2 $>$ clave1); utilizando el orden lexicográfico de las palabras. Se supone dada la función `esIgual(char *clave1, char *clave2)` que devuelve 1 cuando clave1 = clave2 y 0 en caso contrario.

a) Una función que dado un diccionario y una palabra devuelva el significado de dicha palabra. El prototipo de la función es el siguiente:
`char* significado(dicc d, char *clave);`
b) Una función que dado un diccionario y una palabra agregue la definición de esa palabra en el diccionario. El prototipo de la función es el siguiente:
`void agregarPalabra(dicc *d, char *clave, char *significado);`
c) Una función que dado un diccionario, una palabra y una definición, cambie el significado de esa palabra en el diccionario. El prototipo de la función es el siguiente:
`void cambiarSignificado(dicc d, char *clave, char *significado);`
d) Una función en lenguaje ensamblador que dado un diccionario y una palabra, elimine esa palabra del diccionario. El prototipo de la función es el siguiente:
`void eliminarPalabra(dicc *d, char *clave);`

Ejercicio 17

Dado un nuevo tipo de datos `conj` (conjunto sobre listas) para manejar conjuntos de números enteros con signo, cuya definición en C es la siguiente:

```
typedef struct _nodo_t{
        long elem;                      /* elemento (un entero con signo) */
        struct _nodo_t  *siguiente;     /* puntero al siguiente */
} dato;

typedef struct _conj_t {
        nodo* primero;          /*puntero al primer elemento del conjunto */
} conj;
```

Invariante de representación: el conjunto no tiene elementos repetidos y esta representado sobre una lista ordenada en forma ascendente.

a) Escriba una función en lenguaje ensamblador que determine si un elemento está en el conjunto. El prototipo de la función es el siguiente:
`int esta(conj c, long elem);`
b) Escriba una función en lenguaje ensamblador que agregue un elemento en el conjunto. El prototipo de la función es el siguiente:
`void agregarElemento(conj* c, long elem);`
c) Escriba una función en lenguaje ensamblador que elimine un elemento en el conjunto. El prototipo de la función es el siguiente:
`void borrarElemento(conj* c, long elem);`
d) Escriba una función en lenguaje ensamblador que realice la intersección entre dos conjuntos c y c_1, devolviendo el resultado en el conjunto c (es decir, $c = c \cap c_1$). El prototipo de la función es el siguiente:
`void interseccion(conj* c, conj c1);`
e) Escriba una función en lenguaje esamblador que realice la unión entre dos conjuntos c y c_1, devolviendo el resultado en el conjunto c (es decir, $c = c \cup c_1$). El prototipo de la función es el siguiente:
`void union(conj* c, conj c1);`
f) Escriba una función en lenguaje esamblador que calcule el promedio de los elementos de conjunto. Tener en cuenta que si bien el resultado final entra en un entero de 64 bits, la suma parcial de todos los elementos debe calcularse con doble precisión (en 128 bits). El prototipo de la función es el siguiente:
`int promedio(conj c);`

Ejercicio 18

Dado un nuevo tipo de datos `hash` para manejar datos de números enteros con signo, cuya definición en C es la siguiente:

```
typedef struct_nodo_t {
        long    valor;
        struct _nodo _t *prox;
} nodo;

typedef struct _hash_t {
        nodo *arreglo[20] ;
} hash;
```

De cada posición del arreglo cuelga una lista de números enteros con signo. Este hash en particular maneja números enteros entre -1000 y 1000 ordenándolos por centena sin repetidos. Dado un número entero con signo x, si -1000 $\leq x <$ −900 entonces x se va a almacenar en la primera lista, si -900 $\leq x <$ -800 se va a almacenar en la segunda y así sucesivamente.

a) Escriba una función en lenguaje ensamblador que dado un hash y un entero con signo indique si el entero está en el hash. El prototipo de la función es el siguiente:
`long esta(hash* ph, long n);`
b) Escriba una función en lenguaje ensamblador que dado un hash y un entero con signo lo agregue en el hash. El prototipo de la función es el siguiente:
`void agregar(hash* ph, long n);`
c) Escriba una función en lenguaje ensamblador que dado un hash y un entero con signo que está en el hash lo borre. El prototipo de la función es el siguiente:
`void eliminar(hash* ph, long n);`
d) Escriba una función en lenguaje ensamblador que calcule el promedio de todos los números enteros del hash. El prototipo de la función es el siguiente:
`long promedio(hash* ph);`

Ejercicio 19

Dado un *File System* sencillo cuya definición viene dada por las siguientes estructuras:

```
enum entry_type { TYPE_FILE, TYPE_DIR };

typedef struct dir_entry {
        enum entry_type this_entry_type;
        char name[8+3];
        void *first_entry;
        void *next_entry;
} __attribute__((packed)) *ptr_dir_entry;

typedef struct file_entry {
        enum entry_type this_entry_type;
        char name[8+3];
        unsigned long attr;
        unsigned long size;
        void *data;
        void *next_entry;
} __attribute__((packed)) *ptr_file_entry;
```

donde el árbol de directorios se arma de la siguiente manera:

- Se tiene un `dir_entry` inicial.
- `first_entry` apunta al primer elemento (archivo o directorio) incluido en él. `NULL` si no corresponde.
- `next_entry` apunta al siguiente elemento dentro del directorio actual. `NULL` si no corresponde.
- `data` apunta a los datos del archivo, guardados en binario.
- `size` indica el tamaño del archivo en bytes

Escriba una función en lenguaje ensamblador que recorra un *File System* pasado como parámetro y calcule el tamaño promedio de los archivos. El prototipo de la función es el siguiente:

```
unsigned long calcularPromedioSize(ptr_dir_entry root_folder);
```

Nota: tener en cuenta que la sumatoria de los tamaños de todos los archivos no necesariamente entra en 64 bits.