TECNÓLOGO EN INFORMÁTICA

PRINCIPIOS DE PROGRAMACIÓN

Segundo Parcial, 23 de Junio de 2010

Por favor siga las siguientes indicaciones:

- **Escriba con lápiz.**
- **Escriba su nombre y número de documento en todas las hojas que entregue.**
- **Numere las hojas e indique el total de hojas en la primera de ellas.**
- **Comience cada ejercicio en una hoja nueva.**
- **El total máximo de puntos del parcial es 60.**
- **La prueba es individual y sin material.**
- **Solo se contestan dudas acerca de la letra de los ejercicios.**

Ejercicio 1 (25 puntos)

En el contexto del laboratorio del curso (batalla naval), suponga que se definió una estructura llamada Ataque_Tablero, la cual contiene un tablero enemigo-al principio inicializado con todas sus posiciones ocultas-, y donde se irán descubriendo las posiciones de los distintos barcos. Esta estructura contiene campos contadores, para guardar la cantidad de aciertos a los distintos tipos de barcos enemigos. Se definió además, otra estructura llamada JugadaGanadora, donde se guardarán los datos del usuario ganador; y por último también se definió otra estructura llamada Coordenadas, para referenciar las posiciones de los elementos del tablero.

Dichas estructuras están definidas de la siguiente manera:

```
struct Ataque_Tablero{
        tipo_celda tableroEnemigo[ ][TAM_TABLERO];  //contiene la ubicación de //los barcos enemigos.
        int cantidadCeldasHundidasTipoA;
        int cantidadCeldasHundidasTipoB;
        int cantidadCeldasHundidasTipoC;

};

const int MAXNOMBRE=…;
struct JugadaGanadora{
        char  nombre[MAXNOMBRE+1];
        int cantidadTiros;
};

struct coordenadas{
        int fila;
        int columna;
};
```

Nota: Recordar que el tipo_celda está definido de la siguiente manera:

typedef enum {OCULTA='*',AGUA='|',BUQUE1='A', BUQUE2='B',PORTA_AVIONES='C'} tipo_celda;

Se pide:

a) Implementar la función

bool ataque(tipo_celda tableroEnemigo[][], coordenadas coord, AtaqueTablero &tab, bool &tirarDeNuevo);

Que recibe como parámetros

tableroEnemigo: Es un tablero que contiene descubiertas las posiciones de los barcos enemigos. Es utilizado para verificar si se ha producido un acierto en el ataque.

coord: indica la fila y columna, donde se quiere atacar al Enemigo.

tab: estructura que representa las posiciones hasta ahora descubiertas en el tablero enemigo por el jugador que está atacando, y contiene contadores acerca de cuantos aciertos han ocurrido hasta el momento a los distintos tipos de barcos. Estos contadores son utilizados para saber cuándo se han hundido todos los baros enemigos.

tirarDeNuevo: parámetro retornado por la función, cuyo valor se establece en verdadero sí y solo sí la posición indicada en coord. ya fue atacada.

Comportamiento esperado de la función:

La función debe actualizar el parámetro 'tab', en la posición indicada en coord.(debe descubrir en el campo tablero que se encuentra en dicha estructura la posición atacada). Además si se encuentra un barco, debe también actualizar alguno de los campos de la estructura AtaqueTablero, correspondientes a cantidadCeldasHundidasTipoX, según sea el tipo de barco que se acertó.

En caso de que se hayan hundido todos los tipos de barcos, la función debe retornar verdadero, en cualquier otro caso la función debe retornar falso.

Si no se produce ningún acierto, simplemente se actualiza el campo tableroEnemigo de la estructura AtaqueTablero descubriendo la posición(la celda del tablero, en ese caso debería establecerse como de tipo AGUA).

El parámetro tirarDeNuevo, debe retornarse en verdadero, si solo si, ocurra que la posición indicada en coord. ya haya sido atacada(fue atacada si en la posición indicada la celda, en el campo tableroEnemigo, de la estructura AtaqueTablero es de cualquier tipo distinto a OCULTA), en cualquier otro caso este parámetro debe retornarse con valor falso.

Solución

bool ataque(tipo_celda tableroEnemigo[][], coordenadas coord, AtaqueTablero &tab, bool &tirarDeNuevo)

```
{
   if(tab.tableroEnemigo[coord..fila][coord..columna]!=OCULTA){
        tirarDeNuevo=true;
       return false;
   }else{
```

```
  tirarDeNuevo=false; //no hay que tirar de nuevo

   tipo_celda tmp=tableroEnemigo[coord.fila][coord.columna];
    tab.tableroEnemigo[coord..fila][coord..columna]=tmp;
  //si se  acerto a algún tipo de barco
    if(tmp=='A') tab. cantidadCeldasHundidasTipoA++;
   else if(tmp=='B') tab. cantidadCeldasHundidasTipoB++;
   else if(tmp=='B') tab. cantidadCeldasHundidasTipoC++;

//verifica si se termino la partida
if(cantidadCeldasHundidasTipoA==2 && cantidadCeldasHundidasTipoB==2 &&
cantidadCeldasHundidasTipoC==3) return true;
else return false;



  }
}//fin func
```

b) Implementar el procedimiento

void ordenar(JugadaGanadora jugadas[], int largo);

Que ordena según el campo cantidadTiros de la estructura JugadaGanadora, el arreglo jugadas. El arreglo deberá ser ordenado de forma creciente.
En el parámetro largo se indica la cantidad de elementos del arreglo jugadas.

Solución

```
//SE UTILIZA EL ALGORITMO DE BURBUJA
void ordenar(JugadaGanadora jugadas[], int largo){

        JugadaGanadora  temp;

        for (int i = 0; i <largo ; i++) {
                for (int j = i + 1; j <=largo; j++) {
                        if (jugadas[i] > jugadas[j]) {
                                temp.cantidadTiros = v[i] .cantidadTiros;
                                strcpy(temp.nombre, v[i].nombre);
                                v[i].cantidadTiros = v[j] .cantidadTiros;
                                strcpy(v[i].nombre, v[j].nombre);
                                v[j].cantidadTiros = temp .cantidadTiros;
                                strcpy(v[j].nombre, temp.nombre);



                }
                }
        }

}
```

Ejercicio 2 (15 puntos)

Hacer una función que tiene como cabezal ***bool validacionTarjeta(int código[16]***) que debe comprobar si el código que existe en el arreglo corresponde a una tarjeta de crédito válida o no. Si no es válida la función debe regresar falso (0) de lo contrario true (1). El código de la tarjeta está en el parámetro código.

El numero se compone de tres partes principales (es a título informativo):

1.- Los 4 primeros dígitos componen el identificativo del banco que cede la tarjeta. Hay un número diferente para cada banco (esto se busca en internet).

2.- El 5º dígito es el tipo de tarjeta e indica qué entidad financiera gestiona esta tarjeta. Ejemplos: Visa (8) y Master Card (9).

3.- Los 10 dígitos siguientes componen el número de usuario e identifican a este de manera única. Digito de control es el último número y se obtiene aplicando un algoritmo especial al resto del número.

El formato es el siguiente 1111 2333 3333 3334

Algoritmo de codificación para validar la tarjeta.

Se realiza en tres pasos. Si tenemos el numero de la tarjeta **4539 4512 0398 735**6 y queremos comprobar que es válido:

1. Multiplicar por dos los números de las posiciones impares (4-3-4-1-0-9-7-5) y dejarlos con un solo digito.

4*2=8
3*2=6
4*2=8
1*2=2
0*2=0
9*2=18-->1+8=9
7*2=14--->1+4=5
5*2=10--->1+0=1

2. Sumar los dígitos de las posiciones pares y los nuevos de las posiciones impares.

5+9+5+2+3+8+3+6+ 8+6+8+2+0+9+5+1=80

3. Si el resultado es múltiplo de 10 entonces el número es válido.

Solución

```
bool validacionTarjeta( int código[16] ){
   for(int i=0;i<16;i=i+2){//los posiciones impares en el número, son las pares en el arreglo
      código[i]=código[i]*2;
     if(código[i]>9){
       codigo[i]=1 + (codigo[i]%10);
     }
  }//for
//suma digitos
int suma=0;
for(int i=0;i<16;i++){
  suma+=codigo[i];
}//fin for
if(suma%10==0)return true;
```

```
 else return false;
 }//fin func
```

Ejercicio 3 (10 puntos)

Implementar el algoritmo de búsqueda lineal en arreglos no ordenados (y sin elementos repetidos). La **función** deberá devolver **-1 si la búsqueda no es exitosa** y el lugar del arreglo ocupado por el número buscado en caso de que la búsqueda sea exitosa.
Use el siguiente cabezal:

```
int busquedaNoOrdenado( int arreglo[], int tamaño, int número);
```

Solución

```
int busquedaNoOrdenado( int arreglo[], int tamaño, int número){
   bool encontré=false;
   int i=0;
  while(i<tamaño && !encontre){
     if(arreglo[i]==numero) encontre=true;
    else i++;
  }//fin while
 if(encontre)return i;
 else return -1;
}
```

Ejercicio 4 (10 puntos)

Implementar una función recursiva que dado un entero como entrada devuelva:

0 si la entrada es 0 o 1
1 si la entrada es 2
2 * f(n-1) + 3 * f(n-2) + 4 * f(n-3) para cualquier n mayor que 2

Solución

```
int f(int n){
    if(n==0 || n==1) return 0;
    else if(n==1) return 2;
    else return 2*f(n-1) + 3*f(n-2) + 4*f(n-3)
}
```