Estructuras de Datos Estáticas y Dinámicas.

Temas:

- Algoritmos sobre arreglos: búsqueda del mínimo, máximo, ordenación por selección e inserción, etc.
- Arreglos paralelos. Algoritmos sobre arreglos paralelos.
- Concepto de tipo de dato definido por el programador (struct o registro).
- Arreglos de registros. Algoritmos sobre arreglos de registros.
- Arreglos dinámicos.

Búsqueda de la posición del menor elemento en un arreglo a partir de una posición dada.

Algoritmo:

1. Se almacena en una variable auxiliar el elemento del arreglo correspondiente a una posición dada (se lo presupone como el menor), y en otra variable la posición dada.
2. Se compara este valor con el resto de los valores del arreglo, y cada vez que se encuentra un elemento menor al guardado en la variable auxiliar, se actualiza este valor con el encontrado.

```
int posicionMenor(int a[], int cantVal, int pos){
    int menor = a[pos];
    int posMenor = pos;
    int index = pos + 1;
    while (index < cantVal){
        if(menor > a[index]){
            menor = a[index];
            posMenor = index;
        }
        index++;
    }
    return posMenor;
}
```

Ordenación por el método de selección

Algoritmo:

1. Se busca la posición del menor elemento de un arreglo comenzando desde la posición i, y se intercambia el menor elemento con el de la posición i.
2. Se repite el primer paso comenzando con i = 0 y terminando con i = N-1; siendo N el tamaño del arreglo.

19	**23**	**45**	**33**	**18**	**1**	**12**	**9**
0	1	2	3	4	5	6	7

i

1	23	45	33	18	**19**	12	9

i

1	23	45	33	18	**19**	12	9

i

1	**9**	45	33	18	**19**	12	**23**

i

!!

```
void ordenacionSeleccion(int a[ ], int cantVal){
    int posMenor;
    int i = 0;
    while(i< cantVal - 1){ /// llego hasta la anteúltima posición
        posMenor = posicionMenor(a, cantVal, i);
        aux = a[posMenor];
        a[posMenor] = a[i];
        a[i] = aux;
        i++;
    }
}
```

Las instrucciones

```
    aux = a[posMenor];
    a[posMenor] = a[i];
    a[i] = aux;
```

pueden cambiarse por una función que realice la tarea de intercambio.

```
void intercambio(int A[ ], int i, int j){
    int aux = A[i];
    A[i] = A[j];
    A[j] = aux;
}
```

```
void intercambio (int *a, int *b){
    int aux;
    aux = *a;
    *a = *b;
    *b = *aux;
}

void ordenacionSeleccion(int a[ ], int cantVal){
    int posMenor;
    int i = 0;
    while(i<cantVal - 1){ //llego hasta la anteúltima posición
        posMenor = posicionMenor(a, cantVal, i);
        intercambio(&a[porMenor], &a[i]);
        i++;
    }
}
```

Inserción de un elemento en un arreglo ordenado (manteniendo el orden)

Supongamos que debemos insertar un nuevo elemento en un arreglo ordenado. Se supone que el arreglo no ha sido usado en forma completa.

10	20	30	40				
			u				

u marca la posición de la última celda ocupada con información útil.
Y sea **dato** una variable con el valor a insertar.

Se recorre el arreglo desde la última posición a la primera buscando el lugar en donde insertar el nuevo dato. Mientras esto se hace, se realiza también un “corrimiento” de los elementos del arreglo, para crear un espacio en donde ubicar el nuevo valor.

Supongamos que se desea insertar el valor 25.-

10	20	30	40				
			u				

10	20	25	30	40			
				u+1			

```
void insertar(int a[ ], int u, int dato){
    int i= u; //ultima pos valida izq
    while (i >= 0 && dato < a[i]){
        a[i+1] = a[i];
        i--;
    }
    a[i+1] = dato;
}
// Se debe incrementar el valor de u para indicar que ha aumentado el
tamaño del arreglo.
```

Ordenación por inserción

Supongo un arreglo completo y desordenado.

```
void ordenacionInsercion(int a[ ], int cantVal){
    int u=0;
    while (u < cantVal - 1){
     //llega hasta la posición del anteúltimo elemento del arreglo.
        insertar(a, u, a[u+1]);
        u++;
    }
}
```

A medida que el algoritmo evoluciona, se marca en el arreglo una “zona ordenada” a la izquierda, separada de otra “zona desordenada” a la derecha. Se va tomando el siguiente elemento (a[u+1]) de la zona desordenada y se lo inserta dentro de la zona ordenada (se utiliza también el lugar u+1, para hacer crecer la zona ordenada).

Arreglos paralelos

Los arreglos paralelos (dos o más) son aquellos que están relacionados por un mismo índice, o sea, las celdas ubicadas en la misma posición (para distintos arreglos) tienen un vínculo entre sus datos. Se utilizan generalmente para asociar datos de distintos tipos y almacenarlos en arreglos de tipos de datos básicos.

En realidad no representa un nuevo tipo de datos, sino una forma en particular de trabajar con un grupo de arreglos.

Por ejemplo:

Almacenar los datos (número de legajo, nombre y año) de un grupo de alumnos. Se crean tres arreglos del mismo tamaño, cada uno del tipo correspondiente para que pueda almacenar uno de los datos.

legajos	121	122	123	124	125
nombres	Alvarez	Perez	Lopez	Garcia	Simpson
años	2	1	2	3	3
	0	1	2	3	4

Como es de suponer, los datos de igual número de celda de cada uno de los arreglos están vinculados, por pertenecer a una misma “entidad de datos”. Esto tiene que ver con el diseño, o sea con el significado que el programador le da a los datos (semántica).

En nuestro ejemplo tenemos almacenados los datos de cinco alumnos; en la celda número uno (subíndice 0) está almacenado el alumno Alvarez cuyo legajo es 121 y cursa el 2 año, y así con los demás.

Inserción ordenada en arreglos paralelos

```
void insertar(int id_windows[ ],int b[ ], int c[ ] int validos, int id, int
datob, int datoc){
    int i= validos-1; //ultima pos valida
    while (i >= 0 && id < id_windows[i]){
        id_windows[i+1] = id_windows[i];
        b[i+1] = b[i];
        c[i+1] = c[i];
        i--;
    }
    id_windows[i+1] = id;
    b[i+1] = datob;
    c[i+1] = datoc;
}

void carga(int id_windows[],int b[], int c[], int* validos)
{
…
do{
  printf(“ingrese id windows”);
  scanf(“%i”, &id);
  …
  insertar(id_windows,b,c,*validos,id,datob,datoc);
  (*validos)++; // *validos = *validos + 1;
}while(*validos < CANTMAX && continuarr == ‘s’);

}
// Se debe incrementar el valor de validos para indicar que ha aumentado el
tamaño del arreglo.
```

Ordenación sobre arreglos paralelos

Para ordenar arreglos paralelos se debe establecer uno de ellos como el arreglo que determinará el criterio de ordenación. Todos los arreglos deberán modificar la ubicación de sus elementos según la ordenación establecida por uno de ellos.

Por ejemplo, si se desea ordenar el conjunto de arreglos paralelos del ejemplo anterior, por orden de legajo, según el algoritmo de ordenación por selección, se tiene:

```
int posicionMenor(int leg[ ], int cantVal, int u){
    int menor = leg[u];
    int posMenor = pos;
    for(int index = u + 1; index < cantVal; index ++){
        if(menor < leg[index]){
            menor = leg[index];
            posMenor = index;
        }
    }
    return posMenor;
}
```

```c
void intercambioPalabra(char pal[ ][30], int i, int j){
    char aux[30];
    strcpy(aux, pal[i]);
    strcpy(pal[i], pal[j]);
    strcpy(pal[j], aux);
}
```

```c
void intercambioEntero(int numero[ ], int i, int j){
    int aux = numero[i];
    numero[i] = numero[j];
    numero[j] = aux;
}
```

```c
void ordenacionSeleccion(int leg[ ], char nombre[ ][30], int anio[ ],
int cantVal ) {
//los tres arreglos tienen la misma cantidad de elementos válidos
    int posMenor;
    int i=0;
    while(i<cantVal - 1){ /// llego hasta la anteúltima posición
        posMenor = posicionMenor(leg, i);
        intercambioPalabra(nombre, posMenor, i);
        intercambioEntero(leg, posMenor, i);
        intercambioEntero(anio, posMenor, i);
        i++;
    }
}
```

Queda para el alumno resolver el algoritmo de ordenación por inserción.

NOTA: como ya vimos en el apunte “Arreglos en C”, un **arreglo de palabras** se define igual que una matriz de caracteres, en donde el primer subíndice determina la cantidad de palabras del arreglo y el segundo subíndice determina la cantidad máxima de caracteres de cada palabra.

Para trabajar cada celda del arreglo de palabras simplemente se usa el nombre de la matriz de caracteres y solamente el primer subíndice, por ejemplo, sea un arreglo de 10 palabras de longitud 30 cada una:

```c
char palabras[10][30];
```

para acceder a la palabra de la celda i (0 <= i < 10) escribimos: **palabras[i]** dentro de la instrucción correspondiente.

Arreglos dinámicos

Se denomina arreglo dinámico a aquel que su tamaño es definido en tiempo de ejecución. En el arreglo estático el tamaño es definido en tiempo de compilación.

Para crear un arreglo en tiempo de ejecución se utiliza la instrucción **malloc**.
Sintaxis:

```
        tipo * variable;
        int cantidad = ...;
        variable = (tipo *) malloc (cantidad * (sizeof(tipo));
```

La instrucción sizeof(tipo) retorna la cantidad de bytes en memoria que ocupa un dato del tipo indicado. Ahora la cantidad de celdas del arreglo es un dato que se puede almacenar en una variable (cantidad). O sea que puede ser ingresada por el usuario por teclado o calculada.

Esta instrucción crea un arreglo que se puede utilizar de la misma manera que los arreglos estáticos, con la ventaja de posponer hasta el tiempo de ejecución la determinación de su tamaño.

La instrucción malloc busca en la memoria disponible una zona de tamaño igual a cantidad * sizeof(tipo), la reserva y retorna la dirección del comienzo de esta zona. Por lo tanto, la variable que determina el arreglo, en realidad es una variable de tipo puntero con la dirección de memoria de la primer celda del arreglo. Esto es válido para los arreglos estáticos también.

Ejemplo:

```
void main() {
    int * a;
    int cantidad;
    printf("ingrese el tamaño de su arreglo: ");
    scanf("%d", &cantidad);
    a = (int *) malloc (cantidad * sizeof(int));
    a[3] = ....
    …
}
```

Esto crea un arreglo llamado a, de tamaño (cantidad de celdas) igual al valor ingresado por el usuario en la variable cantidad.
Son muy útiles para ahorrar memoria.
Ver también las funciones calloc() y realloc().