Principios de Programación

El lenguaje C

1. Arreglos

Los **arreglos** son estructuras de datos consistentes en un conjunto de datos del mismo tipo. Los arreglos tienen un tamaño que es la cantidad de objetos del mismo tipo que pueden almacenar. Los arreglos son entidades estáticas debido a que se declaran de un cierto tamaño y conservan éste todo a lo largo de la ejecución del programa en el cual fue declarado.

Decimos arreglo o array indistintamente.

1.1. Declaración

Ejemplo de declaración:

```
int arreglo1[30]
```

declara que arreglo1 es un arreglo que puede contener 30 enteros.

```
#define TAMANIO 100

int arreglo2[TAMANIO]
```

declara que arreglo2 es un arreglo que puede contener TAMANIO enteros. La ventaja de esta última declaración es que todos los programas que manipulen arreglo2 utilizarán como tamaño TAMANIO y si quiero cambiar el tamaño del array alcanza con cambiar la definicion de TAMANIO.

Observar que en la declaración se especifica: tipo de los elementos, número de elementos y nombre del arreglo.

Un arreglo consta de posiciones de memoria contiguas. La dirección más baja corresponde al primer elemento y la más alta al último. Para acceder a un elemento en particular se utiliza un índice.

En C, todos los arreglos usan cero como indice para el primer elemento y si el tamaño es n, el indice del último elemento es n-1.

Ejemplo de programa que utiliza un array:

```
main ()
{
      int arreglo2[TAMANIO];
```

```
/* cargo array con valor igual al indice mas 10*/
      for (int i=0;i<TAMANIO;i++)
          arreglo2[i]=i+10;

/* imprimo contenido del arreglo */
      for (int i=0;i<TAMANIO;i++)
          printf("Elemento %d del array es %d\n",i+1,arreglo2[i]);
}
```

Del programa anterior podemos extraer que :

1. para cargar un elemento de un array coloco el nombre de array seguido de un indice entre parentesis rectos y asocio el valor que desee.
2. para acceder al valor de un elemento del array coloco el nombre de array seguido de un indice entre parentesis rectos.
3. los indices con los que accedo al array varían entre 0 y la cantidad de elementos menos 1.

Los nombres de arrays siguen la misma convención que los nombres de variable.

Ejemplos de declaraciones:

```
int notas[8] /* almacena ocho notas */

char nombre[21] /* almacena nombres de largo menor o igual a 20 */

int multiplos[n] /* donde n tiene un valor, declara un arreglo de tamaño n*/
```

Los indices pueden ser cualquier expresión entera. Si un programa utiliza una expresión como subindice esta se evalua para determinar el índice. Por ejemplo si a=5 y b=10, el enunciado arreglo2[a+b] +=2, suma 2 al elemento del arreglo número 15.

Puedo utilizar un elemento del arreglo en las mismas expresiones que variables del tipo correspondiente. En el caso de arreglo2, puedo utilizar cualquiera de sus elementos en expresiones donde pueda utilizar una variable entera, por ejemplo

```
printf(" %d", arreglo2[0]+arreglo2[15]+arreglo2[30]);
```

Los arreglos pueden ser declarados para que contengan distintos tipos de datos. Por ejemplo un arreglo del tipo char puede ser utilizado para almacenar una cadena de caracteres.

1.2. Inicialización

Los elementos de un arreglo pueden ser inicializados en la declaración del arreglo haciendo seguir a la declaración un signo de igual y una lista entre llaves de valores separados por comas.

Por ejemplo

int n[10]={32, 27, 64, 18, 95, 24, 90, 70, 8, 3};

Si en la declaración hay menos inicializadores que el tamaño del array, los elementos son inicializados a cero. Puedo entonces inicializar todo un array en 0 con la declaración:

int n[10]={0};

Declarar más inicializadores que el tamaño del arreglo es un error de sintaxis.

Si en una declaración con una lista inicializadora se omite el tamaño del arreglo el número de elementos del arreglo será el número de elementos incluidos en la lista inicializadora.

Por ejemplo

int a1 [] = {1,2,3,4,5};

crea un arreglo de 5 elementos.

No se puede asignar un arreglo en otro, se tiene que copiar posición a posición.

1.3. Operaciones frecuentes:

Recorrido de un arreglo de tamaño n:

- El índice puede ir del primero al último elemento:

```
for (i=0;i < n;i++)
    {
     proceso
    }
```

- El índice puede ir del último al primer elemento:

```
for (i=n-1;i >= 0;i–)
    {
     proceso
    }
```

1.4. Cómo pasar arreglos a funciones, ejemplos: funciones en arreglos de enteros

Supongamos quiero definir funciones para modularizar el manejo de arreglos de enteros. Definiremos funciones que leen datos y los almacenan en arreglos, definiremos funciones que imprimen los datos almacenados en un arreglo, definiremos una función que halla el máximo en un arreglo, definiremos una función que suma los elementos en un arreglo y usando ésta última una función que calcula el promedio de un arreglo.

```
/* Funcion leo_arreglo, lectura de un arreglo de enteros */
void leo_arreglo(int a[],int n)
{
      for (int i=0;i < n;i++)
          {
           printf("Ingrese elemento :");
           scanf ("%d",&a[i]);
           printf("\n");
          }
}
```

invocamos la función anterior pasando como argumento el nombre del arreglo y su tamaño, por ejemplo:

leo_arreglo(array2,TAMANIO)

Estudiemos la función. Recibe como parametros un arreglo (parámetro a) y un entero (parámetro n) que representa el tamaño del arreglo.

C pasa de forma automática los arreglos a las funciones utilizando simulación de llamadas por referencia (cualquier modificación que realice en la función al array tendra efecto en el array que se pasa como parámetro). Esto es asi porque el nombre del arreglo coincide con la dirección del primer elemento del arreglo.

A pesar de que se pasan arreglos completos simulando llamadas por referencia, los elementos individuales de arreglo se pasan en llamadas por valor.

Para que una función reciba un arreglo en una llamada de función la lista de parametros de función debe especificar que se va a recibir un arreglo (lo hacemos colocando tipo, nombre y los parentesis rectos). No es obligatorio (pero es útil) pasar como argumento el tamaño del array.

Si se indica el tamaño del arreglo dentro de los paréntesis rectos el compilador lo ignorará.

Como el pasaje del array es por referencia, cuando la función utiliza el nombre del arreglo **a** se estará refiriendo al arreglo real en el llamador.

Veamos las otras funciones:

```
/* Funcion imprimo_arreglo, impresion de un arreglo de enteros */
```

```
void imprimo_arreglo(int a[],int n)
{
      for (int i=0;i<n;i++)
          {
           printf("Elemento numero %d = %d", i+1, a[i]);
           printf("\n");
          }
}
```

```
/* funcion buscar_maximo: halla el maximo en un arreglo */
int buscar_maximo(double valores[], int n)
{
    int maximo_pos = 0;
    for (int i = 1; i < n; i++) {
        if (valores[i] > valores[maximo_pos]) {
          maximo_pos = i;
        }
    }
    return maximo_pos;
}
```

```
/* Funcion sumo_arreglo, devuelve la suma de un arreglo de enteros */
int sumo_arreglo(int a[],int n)
{
    int suma=0;

    for (int i=0;i<n;i++)
        suma += a[i];
    return suma;
}
```

```
/* Funcion promedio: devuelve el promedio de un arreglo */
float promedio(int a[],int n)
{
      return sumo_arreglo(a,n)/n;
}
```

1.5. Ejemplos

1. De funciones y programas que computan arreglos:

   - Función que inicializa un arreglo de n lugares con los valores del 1 al n:

     void inicializo(int a[],int n)

```
{
      for(int i=0;i<n;i++) a[i]=i+1;
}
```

- Función que calcula la suma de dos arreglos de n elementos reales:

```
void sumo(float a[],float b[],float c[],int n)
{
      for (int i=0;i <n ; i++)
          c[i]=a[i]+b[i];
}
```

- Función que calcule los primeros n números de Fibonacci:

```
void Fibonacci(int a[],int n)
{
      a[0]=0;
      a[1]=1;
      for (int i=2;i<n;i++)
          a[i]=a[i-1]+a[i-2];
}
```

- Función que invierte un arreglo :

```
void invierto(int b[],int n)
{
      int temp;

      for (int i=0;i < n/2;i++)
          {
            temp = b[i];
            b[i]=b[n-i-1];
            b[n-i-1] = temp;
          }
}
```

De otra manera:

```
void invierto(int b[],int n)
{
      int i,j,temp;

      for (i=0,j=n;i < j;i++,j–)
          {
            temp = b[i];
            b[i]=b[j];
            b[j] = temp;
          }
}
```

- Programa que cuenta número de dígitos, espacios y otros.

```
void main()
{
      int c, i, nblancos, notros;
      int ndigitos[10];

      nblancos = notros = 0;
      for (i = 0; i < 10; ++i) ndigitos[i] = 0;

      while ((c = getchar()) != EOF)
            if (c >= '0' && c <= '9')
              ++ndigitos[c - '0'];
            else if (c == ' ' || c == '\n' || c == '\t')
                  ++nblancos;
                else
                     ++notros;

      printf("digitos = ");
      for (i = 0; i < 10; ++i) printf(" %d ", ndigitos[i]);
      printf("blancos = %d, otros = %d\n", nblancos, notros);
      system("PAUSE");
}
```

- Programa que cuenta número de vocales y otros en la entrada.

```
main ()
{
      int c, i, notros;
      int nvocales[6];

      notros = 0;
      for (i = 0; i < 5; ++i) nvocales[i] = 0;

      while ((c = getchar()) != EOF)
            switch (c) {
                  case 'a' : case 'A': ++nvocales[0];break;
                  case 'e' : case 'E': ++nvocales[1];break;
                  case 'i' : case 'I': ++nvocales[2];break;
                  case 'o' : case 'O': ++nvocales[3];break;
                  case 'u' : case 'U': ++nvocales[4];break;
                  default: ++notros; }

      printf("vocales = ");
      for (i = 0; i < 5; ++i)
          printf(" %d", nvocales[i]);
      printf("otros = %d\n", notros);
      system("PAUSE");
}
```

2. De uso de funciones sobre arreglos:

   - Programa que lee e imprime notas finales de 10 estudiantes (utilizando las funciones anteriores) :

```
main ()
{
        int notas[10];

        printf("Ingrese las notas finales de los 10 estudiantes\n");
        leo_arreglo(notas,10);
        printf("Las notas de los estudiantes son: \n");
        imprimo_arreglo(notas,10);
        system("PAUSE");
```

   - Programa que lee notas finales de 10 estudiantes e imprime su promedio (utilizando las funciones anteriores) :

```
main ()
{
        int notas[10];

        printf("Ingrese las notas finales de los 10 estudiantes\n");
        leo_arreglo(notas,10);
        printf("El promedio de las notas de los estudiantes es: %f", promedio(notas,10));
        system("PAUSE");
```

   - Programa que lee un valor n, almacena los primeros n números de Fibonacci y los imprime:

```
main ()
{
        int n;

        printf("Ingrese numero :");
        scanf(" %d", &n);
        putchar('\n');
        int fib[n];

        Fibonacci(fib,n);
        printf("Primeros %d numeros de Fibonacci son :\n");
        imprimo_arreglo(fib,n);
        system("PAUSE");
```

1.6. Arreglos constantes

Podrı'an existir situaciones en las cuales no queremos permitir que una función modifique un arreglo.

C proporciona un calificador llamado **const** que se puede utilizar para evitar la modificación de variables en particular de arreglos.

Cuando en un parametro de una función que representa un arreglo se antecede la declaración del parametro con la palabra clave **const**, los elementos del arreglo se convierten en constantes y cualquier intento de modificar un elemento del arreglo dentro de la función da como resultado un error en tiempo de compilación.

Ejemplo:

```
void trato_de_modificar(const int b [])
{
     b[0] = 1;
}
```

dará error de sintaxis.

1.7. Arreglos de caracteres o strings

Hasta ahora solo nos hemos ocupado de arreglos de enteros. Sin embargo los arreglos son capaces de contener datos de cualquier tipo. Estudiaremos el almacenamiento de **strings** o **cadenas** en arreglos de caracteres.

Un arreglo de caracteres puede ser inicializado utilizando una cadena, por ejemplo, la siguiente declaración:

```
char string1[] = "primero";
```

inicializa los elementos del arreglo string1 con las letras de la palabra primero. El tamaño del arreglo queda determinado por la cantidad de letras de primero.

Es importante notar que la cadena "primeroçontiene 7 caracteres más un caracter especial que indica la terminación de la cadena que es el '\0' o caracter nulo. Entonces primero realmente consta de 8 caracteres y este es el tamaño del array string1.

La declaración anterior es equivalente a :

```
char string1 [] = {'p', 'r', 'i', 'm', 'e', 'r', 'o', '\0'}
```

Dado que una cadena es un arreglo de caracteres podemos tener acceso a los caracteres individuales de una cadena utilizando la notación de nombre de arreglos con subindices.

1.7.1. Lectura e impresion de cadenas

Podemos imprimir y leer una cadena con el especificador de conversión " %s".

Los enunciados

```
printf(" %s","Hola");
printf(" %s",string1);
```

imprimiran Hola y primero. El enunciado:

```
char string2[20];

scanf(" %s", string2);
```

lee una cadena del teclado y la coloca en string2. Notar que el nombre del arreglo se pasa a scanf sin colocar el & que en otros casos se usa. El & es utilizado por lo regular para darle a scanf la localización de una variable en la memoria. Como el nombre de un arreglo es la dirección de inicio del arreglo el & no es necesario.

Es responsabilidad del programador asegurarse que el arreglo al cual se lee la cadena sea capaz de contener cualquier cadena que el usuario escriba en el teclado. La función scanf lee caracteres del teclado hasta encontrarse con el primer caracter de espacio en blanco. Se almacenará la cadena ingresada y el caracter nulo.

printf imprime hasta que encuentra el caracter nulo.

Si leemos una cadena con getchar(), es nuestra responsabilidad colocar el '\0' al final del arreglo.

Ejemplo:

```
/* leo cadena con getchar() */
main ()
{
      char ca[15];
      int i=0;
      while ((ca[i]=getchar())!=EOF) i++ ;
      ca[i]='\0';
}
```

1.8. Casos de Estudio, funciones con arreglos de caracteres

Ejemplo 1

Escribamos un programa que lee un conjunto de lineas e imprime la más larga. Un esquema del programa es:

```
mientras haya otra linea
        si es mas larga que la anterior mas larga
           salvar la linea y su largo
imprimir linea mas larga
```

Este esquema muestra que el programa se puede dividir naturalmente en funciones. Una pieza lee una nueva linea, main testea si es más larga, otra función salva la linea y main realiza el resto.

Empezemos escribiendo una función **getline** que lee una linea de la entrada. Esta función lee hasta que encuentra EOF o nueva linea.Retorna el largo de la linea leida y coloca el '\0' al final del string leido. Como el EOF se ingresa en una linea nueva, cuando se ingrese EOF se devolverá largo cero y el string consistirá solo en el '\0'.

Para salvar la linea utilizamos la función copiar.

Las funciones son las siguientes:

```
/* funcion getline */
int getline(char s[],int n)
{
    int c, i=0;

    while (--n > 0 && (c=getchar())!=EOF && c!='\n')
          s[i++] = c;
    if (c == '\n') s[i++]=c;
    s[i]='\0';
    return i;
}


/* funcion copiar, reliza una copia del array que se ingresa */
void copiar(char s1[], char s2[])
{
     int i=0;

     while ((s2[i]=s1[i])!='\0') i++;
}
```

la función main es:

```
#define MAX 1000

main ()
{
      int largo,maximo=0;
      char linea[MAX];
      char copia[MAX];

      while ((largo=getline(linea,MAX))>0)
            if (largo > maximo)
               {
                maximo=largo;
                copiar(linea,copia);
               }
      if (maximo > 0) printf(" %s", copia); }
```

Ejemplo 2

Veamos un programa que lee un conjunto de lineas e imprime cada linea de la entrada que contiene un patron particular. Por ejemplo

Es el tiempo
para todo el bien
el hombre va en busca
de nuevas sensaciones.

Si buscamos el patrón **el**, lo encontramos en las primeras tres lineas.

La estructura básica del programa que resuelve el problema es:

```
mientras haya otra linea
        si la linea contiene el patron
            imprimo la linea
```

Si bien es posible poner todo el código del programa en la instrucción main, una manera mejor de definir el programa es hacer de cada operación una función separada.

Utilizaremos la función **getline** del ejemplo anterior.

Definiremos una función **indice** que regresa la posición en la linea de entrada en la cual se encontró el patrón, retorna -1 si el patrón no se encontró.

Por último main imprime cada linea que contiene el patrón.

```
/* funcion indice, retorna indice de t en s, -1 si no se encuentra */
int indice(char s[], char t[],int n)
{
    int i,j,k;

    for (i=0;s[i]!='\0';i++)
        {
          for (j=i,k=0; t[k]!='\0' && s[j]==t[k]; j++, k++) ;
          if (t[k] == '\0') return i;
        }
    return -1;
}
```

la función main que llama estas funciones es la siguiente:

```
#define MAX 1000
#define largo 10
main ()
{
       char linea[MAX];
       char patron[largo];

       printf("Ingrese patron");
       scanf(" %s", patron);
       while (getline(linea,MAX) > 0)
```

```
            if (indice(linea,patron) >= 0)
              printf(" %s",linea);
}
```

2. Arreglos con múltiples subindices

En C los arreglos pueden tener múltiples subindices. Podemos utilizarlos por ejemplo en tablas de valores que almacena información arreglada en filas y columnas. Para identificar un elemento de la tabla debemos especificar dos subindices, el primero especifica el renglón del argumento y el segundo identifica la columna.

Podemos especificar arreglos con más de 2 subindices.

Un arreglo de multiple subindice puede ser inicializado en su declaración en forma similar a los de un subindice. Por ejemplo un arreglo de doble subindice puede ser declarado e inicializado con:

```
int b[2][3] = { {1,2,3}, {3,4,5}}
```

si para un renglón dado no se proporcionan suficientes inicializadores los elementos restantes de dicho inicializador se setarán en cero.

Veamos una función imprimo_arreglo que imprima los valores de un array de dos dimensiones por fila.

```
/* imprimo array de dos dimensiones por fila */

void imprimo_arreglo(int a [][3],int n)
{
    int i,j;

    for(i=0;i < n;i++)
    {
       printf("Fila %d\n", i);
       for (j=0;j < 3;j++)
           printf(" %d ", a[i][j]);
       printf("\n");
    }
}
```

Notar que la definición de la función especifica el parámetro del arreglo como int a[][3]. El primer subindice de un arreglo de multiples subindices no se requiere pero todos los demás subindices son requeridos. El compilador utiliza estos subindices para determinar las localizaciones en memoria de los elementos en los arreglos de multiples subindices. Los elementos del arreglo son almacenados en memoria de forma consecutiva renglón despues de renglón.

Si queremos generalizar el programa, podemos definir la cantidad de columnas en una constante del siguiente modo:

```
/* imprimo array de dos dimensiones por fila */

#define nro_columnas 3

void imprimo_arreglo(int a [][nro_columnas],int n)
{
      int i,j;

      for(i=0;i < n;i++)
      {
         printf("Fila %d\n", i);
         for (j=0;j < nro_columnas;j++)
             printf(" %d ", a[i][j]);
         printf("\n");
      }
}
```

Ejemplo

Consideremos un arreglo donde cada renglon representa un alumno y cada columna representa una calificación en uno de los cuatro examenes que los alumnos pasaron durante el semestre.

Utilizaremos las siguientes funciones: **minimo** determina la calificación más baja de entre las calificaciones de todos los estudiantes. **maximo** determina la calificación más alta de entre las calificaciones de todos los estudiantes. **promedio** determina el promedio para el semestre de un alumno en particular. **imprimo_arreglo** es la función que ya vimos.

Las funciones minimo y maximo reciben tres argumentos: el arreglo llamado **notas**, el número de alumnos (renglones) y el número de examenes (columnas). Las funciones iteran a través del arreglo notas utilizando estructuras for anidadas.

Las funciones se presentan a continuación:

```
/* funcion minimo */
int minimo(int notas[][nro_examenes],int n,int m)
{
    int i,j,menorgrado=notas[0][0];

    for (i=0; i < n;i++)
        for (j=0; j < m; j++)
            if (notas[i][j] < menorgrado) menorgrado = notas[i][j];
    return menorgrado;
}


/* funcion maximo */
```

```
int maximo(int notas[][nro_examenes],int n,int m)
{
    int i,j,mayorgrado=notas[0][0];

    for (i=0; i < n;i++)
        for (j=0; j < m; j++)
            if (notas[i][j] > mayorgrado) mayorgrado = notas[i][j];
    return mayorgrado;
}


/* funcion promedio */
int promedio(int notas_de_alumno [],int n)
{
    int i,total=0;

    for (i=0; i < n;i++)
        total += notas_de_alumno[i];
    return (float) total/n;
}
```

la función promedio se invoca con una fila del arreglo notas del siguiente modo:

```
promedio(notas[alumno],cant_examenes);
```