

## MODULO 2

### OBJETIVOS

---

Con el desarrollo de este módulo se pretende que el estudiante:

- Adquiera la capacidad de realizar programas usando los tipos de datos agregados nativos, implementar funciones de búsqueda e intercambio en arreglos de estructuras.
- Aprenda a utilizar los tipos de datos agregados iterados y logre programarlos.

### CONTENIDO

---

- 1 Especificación de un tipo de dato
    - 1.1 Especificación de un tipo de dato escalar
    - 1.2 Especificación de un tipo de dato agregado
  - 2 Tipos de datos agregados iterados
  - 3 Programación con tipos de datos agregados nativos en C++
    - 3.1 Arreglos de estructuras
      - 3.1.1 Recorridos, búsquedas e intercambios en arreglos de estructuras
  - 4 Programación con tipos de datos agregados iterados en C++
- A Referencias
- B Taller
- C Laboratorio guiado

## 1 ESPECIFICACIÓN DE UN TIPO DE DATO

---

Para utilizar un tipo abstracto en un programa debemos tener un cierto conocimiento del tipo. A este conocimiento lo denominaremos la “especificación” del tipo.

### 1.1 ESPECIFICACIÓN DE UN TIPO DE DATO ESCALAR

Para especificar un tipo cuyos valores son escalares se debe conocer los siguientes elementos:

- **Identificación:** Para declarar variables asociadas al tipo debemos contar con símbolos específicos del lenguaje para referirnos al tipo (un nombre).
- **Dominio:** Es el conjunto de valores que hacen parte del tipo. El dominio se especifica de forma explícita, enumerando los valores que pertenecen al tipo, o de forma implícita definiendo un conjunto de condiciones que definen cuando un valor pertenece al tipo.
- **Operaciones:** Que operaciones (funciones) son aplicables a los valores del tipo y que efecto tiene su aplicación. Lo que involucra:

*Plantilla o firma:* símbolo asociado a la operación; operandos de la operación (cuantos y de que tipo); y sintaxis de los términos que la usan (como expresarla en el lenguaje dado el símbolo y los operandos)

*Significado (o efecto de la operación):* Que puede ser descrito de varias formas:

-Intuitiva: Por medio de descripciones verbales del lenguaje o con base en la experiencia.

-Axiomática: Definiendo las propiedades de la operación por medio de axiomas.

-Pre y Post condiciones: Describen el estado del sistema antes y después de la aplicación de la operación. Las precondiciones son aserciones que deben ser verdaderas para que la operación se ejecute correctamente y las postcondiciones describen tanto el resultado de las acciones llevadas a cabo por la operación como el valor retornado por esta (si lo hay)

- **Constantes Literales:** Sintaxis de las construcciones del lenguaje que se usan para referirse a *elementos* específicos del tipo. También se les conoce como *constantes*.

---

#### *Ejemplo 1.1: Constantes literales*

5.43 Es un literal que representa un valor de tipo flotante.

“Juan Perez” Es un literal que representa un valor de tipo string.

---

---

*Ejemplo 1.2: Especificación mediante pre y post condiciones de un TD escalar*  
(Basado en el ejemplo de Stubbs & Webre, Data Structures pg 12)

Se desea especificar un tipo de dato llamado color, cuyo dominio son los colores primarios (rojo, amarillo y azul) y los colores secundarios (verde, naranja, violeta), que se pueden obtener combinando colores primarios.

La operación asignar es necesaria, porque siempre debe haber secuencias de operaciones que permitan crear variables del tipo especificado (*constructor*).

La operación mezclar combina dos colores primarios para crear uno secundario.

La operación primario averigua si un color es primario o no.

La operación formado\_por devuelve los colores primarios que forman el color secundario dado como argumento.

La operación asignar da un valor, dado como argumento, a una variable de tipo color dada como argumento.

#### *Especificación tipo de dato color*

**Dominio:** Los posibles valores iniciales son [rojo, amarillo, azul, verde, naranja, violeta]

**Operaciones:** Se definen cuatro operaciones:

Color Mezcla(Color c1, Color c2)

**Pre:** c1 y c2 son colores primarios.

**Post:** La mezcla es un color formado por la combinación de los colores c1 y c2 en cantidades iguales.

bool Primario(Color c)

**Pre:** ninguna

**Post:** Si c es un color primario, entonces primario devuelve verdadero, de lo contrario devuelve falso.

Void Formado\_por(Color c, \*Color c1, \*Color c2)

**Pre:** c es un color secundario.

**Post:** c1 y c2 son dos colores primarios que forman el color c.

Void Asignar(\*Color c1, Color c2)

**Pre:** Ninguna

**Post:** c1 tiene el valor de c2.

---

## 1.2 ESPECIFICACIÓN DE UN TIPO DE DATO AGREGADO

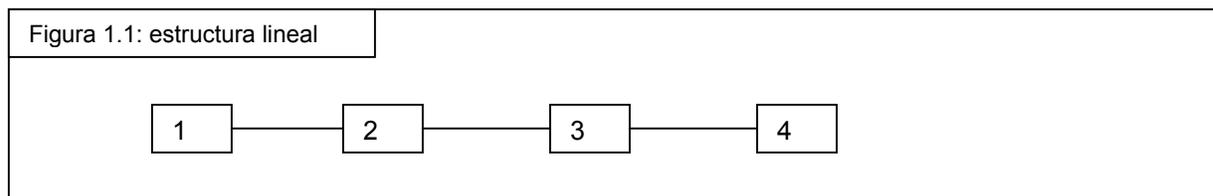
Para llevar a cabo la especificación de un tipo de dato agregado se debe conocer, en adición a los elementos anteriores, los siguientes elementos:

- **Componentes:** El número y tipo de los componentes. El número de componentes puede ser fijo o variable. El tipo de los componentes puede ser, también, fijo o variable. Solo consideraremos estructuras con componentes de número fijo y tipo variable, y estructuras de componentes con número variable y tipo fijo. A los primeros los denominaremos “tipos estructurados” y a los segundos “tipos iterados”.

- **Estructura de composición:** La forma como están unidos los componentes de la estructura. Esta forma será caracterizada describiendo los “enlaces” entre los componentes. Los enlaces entre los componentes serán conexiones rotuladas y dirigidas que pueden ser usadas para acceder a un elemento desde otro (“recorriendo” el enlace). Con respecto a estos enlaces, nos concentraremos en su “aridad” y “ciclicidad”. La aridad de un enlace es el número de componentes que conecta (o enlaza) un solo enlace. La ciclicidad de una cadena de enlaces es la propiedad de conectar un elemento a si mismo recorriendo los enlaces en una sola dirección.

*Ejemplo 1.1: Especificación con pre y post condiciones de un TD agregado*  
(Basado en el ejemplo de Stubbs & Webre, Data Structures pg. 14)

A continuación se especificará un tipo de dato llamado `cadena_letras` cuyos valores son secuencias de caracteres ASCII. El tamaño máximo de la cadena es 5. Ya que la cadena es un arreglo, su estructura de composición es lineal, es decir, los enlaces entre los elementos son de uno a uno (cada elemento del es adyacente al anterior y el siguiente) como se muestra en la figura:



La operación `letra_izq` devuelve la primera letra de la cadena.  
 La operación `agregar` añade una letra al final de la cadena.  
 La operación `invertir` invierte la cadena.

*Especificación: tipo de dato `cadena_letras`*

**Elementos componentes:** Los elementos que lo componen son caracteres ASCII.

**Estructura de composición:** Hay una relación lineal entre los caracteres de la cadena.

**Dominio:** Todas las cadenas de caracteres ASCII de tamaño 5.

**Operaciones:**

```
char letras_izq(*Cadena_letras c)
```

**pre:** el tamaño de la cadena es mayor que 0.

**post:** el valor devuelto es la primera letra de la cadena a la que apunta c.

```
void pegar(char l, *Cadena_letras c)
```

**pre:** el número de elementos de la cadena apuntada por c es menor que 5.

**post:** la cadena c tiene una longitud mayor en una unidad a la longitud que tenía antes de efectuarse la operación y la letra l es su último carácter.

```
void invertir(*Cadena_letras c)
    pre: Ninguna
    post: La secuencia de caracteres en la cadena c esta invertida.
```

---

## 2 TIPOS DE DATOS AGREGADOS ITERADOS

Los tipos de datos agregados iterados son aquellos con un número variable de componentes de tipo fijo. Estos tipos de datos son útiles en tareas como la búsqueda de palabras en un texto, en las que el tamaño de la estructura usada para guardar las palabras cambia durante el proceso.

---

### *Ejemplo 2.1: Creación de un tipo de dato agregado iterado*

En este ejemplo se usa un arreglo de 100 caracteres para *emular* un tipo de dato agregado iterado con un tamaño máximo de 100 caracteres. Se llamará cadena al tipo de dato iterado, que se usará para guardar una cadena de caracteres ingresada por el usuario de la cual no se conoce su tamaño (debido a esto decimos que el tipo de dato tiene un número *variable* de componentes):

Se utiliza el carácter '\0' para indicar el fin de la cadena, C++ lo inserta automáticamente luego de que el usuario termina de introducir la cadena.

```
/******Probado en Borland C++ 5.0******/
typedef char cadena[100];    /*Define el tipo de dato cadena*/
cadena c1;                  /*Se define un elemento de tipo cadena llamado c1*/
scanf("%s", c1);
```

---

## 3 PROGRAMACIÓN CON TIPOS DE DATOS AGREGADOS NATIVOS EN C++

### 3.1 ARREGLOS DE ESTRUCTURAS

Estas estructuras son muy comunes y de mucha utilidad. Consisten en un arreglo unidimensional cuyos miembros son estructuras del mismo tipo, previamente definidas. Puede accederse a un campo de alguna de las estructuras que forman el arreglo por medio de la sintaxis:

```
Nombre_arreglo[indice].nombre_campo
```

#### 3.1.1 Recorridos, búsquedas e intercambios en arreglos de estructuras

---

##### *Ejemplo 3.1: Recorrido en un arreglo de clases*

Suponiendo que ya está definida la clase (tipo de dato) estudiante, puede crearse un arreglo de estudiantes llamado grupo. A continuación se crea la clase estudiante, nótese que ha debido crearse un constructor por defecto requerido para la creación del arreglo. La función key() se

encarga de retornar la *clave* (número de carné) del estudiante, que se compone de un número de identificación antecedido por el año de ingreso:

```
/******Probado en Borland C++ 5.0******/

class estudiante
{
    private:
        int aCarne;
        char *aNombre;
        float aNota;
        int aAnno_ing;

    public:
        estudiante(int eCarne, float eNota, char *eNombre, int
                    eAnno_ing);
        estudiante();          /*Constructor por defecto*/
        float ver_nota();
        int key();
        char *ver_nombre();
        int ver_anno();
        void asignar_nota(float eNota);
        void asignar_carne(int eCarne);
        void asignar_anno(int eAnno_ing);
        void asignar_nombre(char *eNombre);
};

estudiante::estudiante(int eCarne, float eNota, char *eNombre, int
eAnno_ing)
{
    aCarne=eCarne;
    aNota=eNota;
    aNombre=eNombre;
    aAnno_ing=eAnno_ing;
}

estudiante::estudiante()
{
    aCarne=0;
    aNota=0;
    aNombre='\0';
    aAnno_ing=0;
}

float estudiante::ver_nota()
{
    return aNota;
}

int estudiante::key()
{
    int aKey;
```

```

        aKey=((aAnno_ing)*10000+aCarne);
        return aKey;
    }

char *estudiante::ver_nombre()
{
    return aNombre;
}

int estudiante::ver_anno()
{
    return aAnno_ing;
}

void estudiante::asignar_carne(int eCarne)
{
    aCarne=eCarne;
}

void estudiante::asignar_nota(float eNota)
{
    aNota=eNota;
}

void estudiante::asignar_nombre(char *eNombre)
{
    aNombre=eNombre;
}

void estudiante::asignar_anno(int eAnno_ing)
{
    aAnno_ing=eAnno_ing;
}

```

Ahora puede definirse un arreglo de estudiantes, aquí se crea uno llamado grupo1:

```
estudiante grupo1[10];
```

Una vez definido el arreglo, pueden invocarse las funciones de la clase estudiante usando la sintaxis “punto” de los arreglos y además pueden hacerse recorridos usando ciclos for o while. La siguiente función asigna una nota dada a un estudiante de cierto grupo:

```

void nota_est_n(estudiante *grupo, float nota, int clave)
{
    for(int i=0; i<10; i++)
    {
        if(grupo[i].key() == clave)
        {
            grupo[i].asignar_nota(nota);
        }
    }
}

```

Para asignar una nota dada a todos los estudiantes del grupo se crea la función:

```

void nota_grupo(estudiante *grupo, float nota)
{
    for( int i=0; i<10; i++ )
    {
        grupo[i].asignar_nota(nota);
    }
}

```

---

### *Ejemplo 3.2: Búsqueda en arreglos de clases*

Usando el arreglo de estudiantes del ejemplo anterior, se implementa a continuación una función que busca el estudiante con la mayor nota y devuelve su número de identificación y la nota. Como la función debe retornar dos resultados, se crea un arreglo donde los pondrá:

```

/*****Probado en Borland C++ 5.0*****/

void buscar_mejor(estudiante *grupo, float *result)
{
    float temp_nota=0.0;
    int temp_id;

    for(int i=0; i<10; i++)
    {
        if((grupo[i].ver_nota())>temp_nota)
        {
            temp_nota=grupo[i].ver_nota();
            temp_id=grupo[i].key();
        }
    }

    result[0]=temp_nota;
    result[1]=temp_id;
}

```

---

### *Ejemplo 3.3: Intercambio*

La siguiente función invierte el arreglo de estudiantes grupo, funciona para arreglos de cualquier tamaño. el argumento dim es el tamaño del arreglo:

```

/*****Probado en Borland C++ 5.0*****/

void invertir(estudiante *grupo, int dim)
{
    int i=0;
    estudiante temp;

    while( (i!=((dim-1)-i)) && (i!=(dim/2)))
    {
        temp=grupo[i];

```

```

        grupo[i]=grupo[ ((dim-1)-i) ];
        grupo[ ((dim-1)-i) ]=temp;
        i++;
    }
}

```

---

#### 4 PROGRAMACIÓN CON TIPOS DE DATOS AGREGADOS ITERADOS EN C++

---

Con el uso de un arreglo de tamaño fijo y una *bandera* (elemento usado para marcar el final de la secuencia de elementos) puede simularse un tipo de dato agregado iterado. Los iterados son tipos de datos muy útiles en aplicaciones donde se desea usar la memoria de manera óptima y no se tiene certeza del tamaño de los elementos, como por ejemplo, en la edición de textos.

---

##### *Ejemplo 4.1: Número de componentes en una cadena*

Usando el tipo de dato cadena, implementado en el ejemplo 2.1 de este módulo, se presenta a continuación la función tamaño, que retorna el número de caracteres contenidos en una cadena dada como parámetro.

```

/*****Probado en Borland C++ 5.0*****/

int tamaño(char *cad)
{
    int i=0;
    int num_el=0;

    while(cad[i]!='\0')
    {
        i++;
        num_el++;
    }

    return num_el;
}

```

---

Ejemplo 4.2:

*La función agregar, añade una cadena al final de otra:*

```

/*****Probado en Borland C++ 5.0*****/

void agregar(char *cad1, char *cad2)
{
    int i=0,fin_cad1=0;

    while(cad1[i]!='\0')
    {
        i++;
        fin_cad1++;
    }
}

```

```
    }  
    i=0;  
    while (cad2[i]!='\0')  
    {  
        cad1[fin_cad1]=cad2[i];  
        i++;  
        fin_cad1++;  
    }  
    cad1[fin_cad1]='\0';  
}
```

---

## ANEXOS

### A REFERENCIAS

[DS1984] Stubbs. Daniel F. & Webre, Neil W., "Data Structures: with Abstract Data Types and Pascal", Brooks/Cole Publishing Company, 1984.  
1: Secciones 1.5 hasta 2.3

### B TALLER

1. Especifique el tipo de dato creado en el laboratorio del módulo anterior
2. Basándose en el arreglo *grupo* de estructuras de tipo estudiante, escriba un programa que busque el estudiante con la mayor nota e imprima su nombre en pantalla.

Ejercicios página 42 y 59 libro de Stubbs y Webre