MODULO 1

OBJETIVOS

Al finalizar este módulo el estudiante deberá:

- Tener un conocimiento básico del proceso computacional y comprender los conceptos de programa e instrucción.

- Comprender que son los tipos de datos y sus conceptos asociados (valores, funciones y operaciones), establecer las relaciones adecuadas entre ellos y tener claras sus diferencias.

- Entender los conceptos de tipos de datos nativos y definidos, tener una visión general de los TD nativos más comunes y de cómo y para qué el programador define sus propios TD.

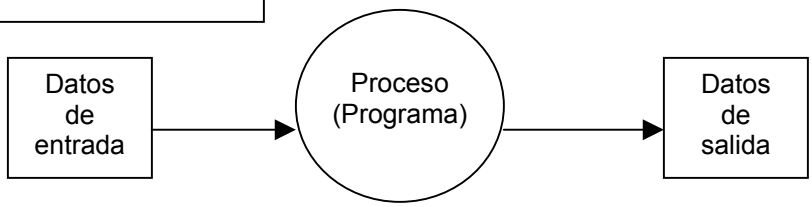
- Diferenciar los tipos escalares y agregados, conocer sus categorizaciones (agregados iterados, agregados estructurados y escalares nativos) Estar en capacidad de concebir un tipo de dato agregado estructurado propio de acuerdo a una aplicación personal y programarlo.

CONTENIDO

1 EL PROCESO COMPUTACIONAL

El computador es una máquina cuya función básica es llevar a cabo operaciones de cómputo (transformaciones) sobre elementos de datos. Como todas las máquinas, el computador recibe una "materia prima" (*datos de entrada)*, la transforma a través de un *proceso* y devuelve un "producto" (*datos de salida)* Pero el computador posee una característica primordial que lo diferencia: el *proceso* que define la transformación efectuada sobre los *datos de entrada* puede ser especificado por el usuario; es por ello que se dice que el computador es *programable.*

Figura 1.1 Proceso computacional



$R = t(D)$

El proceso computacional es análogo a una función matemática, con R el rango (datos de salida), D el dominio (datos de entrada) y t() la transformación.

1.1 PROGRAMAS

Un programa es una transformación específica, definida y expresada de forma que pueda ser utilizada por el computador. Como una primera aproximación, puede definirse *programa* como un conjunto de pasos, especificados por el usuario en un lenguaje comprensible para la máquina (*lenguaje de programación*), que *definen el proceso* que se efectúa sobre los datos; de esto se concluye, que al ejecutarse el programa, se lleva a cabo un *proceso computacional.*

1.2 INSTRUCCIONES

Cada proceso está compuesto de subprocesos y estos últimos a su vez, se componen de otros subprocesos más elementales. Es así como los programas están formados por un conjunto de funciones, estas por subfunciones y así sucesivamente hasta llegar a las *operaciones elementales del procesador* u *operaciones de máquina*.
Las instrucciones son *construcciones de lenguaje máquina* (cadenas finitas de dígitos) que permiten invocar a las operaciones elementales. Existe un mapeo de uno a uno entre las operaciones disponibles y las instrucciones. (Es decir, por cada operación hay una y sólo una instrucción)

1.2.1 Instrucciones nativas:

Las instrucciones nativas son aquellas propias de la arquitectura que se está considerando. Cada procesador trae por defecto un conjunto de operaciones definidas por el fabricante llamado *instruction set* (conjunto de instrucciones) y para cada una de ellas se ha definido una instrucción nativa.

Ejemplo 1.1: Instrucción mov

Una de las operaciones básicas más importantes de la arquitectura 80x86 (procesadores Intel) es aquella que permite copiar un dato de un lugar a otro, la instrucción que invoca a esta operación tiene esta sintaxis:

```
Mov orig, dest
```

1.3 VARIABLES

Las variables son un espacio de memoria (espacio de almacenamiento de datos, La memoria es donde son ejecutados los programas y procesos de una computadora) en el que se almacena un dato de un tipo dado. La característica más importante de las variables es que, como su nombre lo indica, es posible cambiar el valor que almacenan en el transcurso del programa.
Cada variable tiene asignado un identificador, formado por uno o más caracteres que permiten referirse a su contenido dentro de su dominio.

1.4 TÉRMINOS

Los términos son construcciones del lenguaje que permiten representar un valor en el programa. Es factible que en algunos lenguajes existan construcciones que además de representar un valor, indiquen la aplicación de una función y la reasignación de una variable.

1.5 ASIGNACIONES

Las variables son útiles debido a que el programador puede definir su valor por medio de construcciones del lenguaje en el transcurso del programa. A este procedimiento se le conoce como asignación. Se debe tener en cuenta que a una variable sólo se le pueden asignar valores que correspondan a su tipo de dato, así, por ejemplo, no se le puede asignar un número flotante a una variable entera.

Ejemplo 1.2: declaración de variables, términos y asignación en C++

Definidas las variables a y b, estas pueden usarse en la construcción de términos y se les pueden asignar los valores representados por estos:

```
int a;            /*Se declara la variable entera a*/
int b;            /*Se declara la variable entera b*/

a=5;              /*Se asigna a a el valor 5*/
b=7+2*(4/2);      /*Se asigna a b el valor del término 7+2*(4/2)*/
a=(a+b)*3;        /*Se asigna a a el valor del término (a+b)*3*/
```

1.6 E/S: ENTRADA / SALIDA

Los dispositivos de entrada / salida permiten que el usuario interactúe con la máquina. Por medio de los dispositivos de entrada el usuario ingresa los datos a procesar en el sistema y los dispositivos de salida presentan los resultados en un formato legible.

Las instrucciones de E/S dan acceso al programador a las funciones básicas de estos dispositivos, permitiéndole capturar datos de los dispositivos de entrada y asignarlos a variables para operar con ellos y mostrar resultados del proceso en los dispositivos de salida.

Ejemplo 1.2.2: Entrada / Salida en C++

Los dispositivos de E/S más comunes son el teclado y el monitor. Las siguientes funciones de C++ permiten ingresar un dato de un tipo dado por teclado e imprimir en pantalla.

```
int a;
scanf("%i",&a);   /*pide al usuario ingresa un valor por teclado*/
printf("El valor de a es %i",a);    /*imprime el valor de a en
                                    pantalla*/
```

1.7 CONTROL DE PROCESO

Para desarrollar programas funcionales se hacen necesarias construcciones del lenguaje que permitan desviar el flujo de ejecución por diferentes caminos de acuerdo a parámetros desconocidos al momento de la codificación y cambiantes en cada ejecución. A estas construcciones se les conoce como sentencias de control de proceso o de control de flujo.
Las más comunes se ilustran en el siguiente ejemplo.

Ejemplo 1.3: Sentencias de control de proceso en C++

If-else: Permite bifurcar el flujo de ejecución según el argumento del if. Esta sentencia ejecuta el código correspondiente al if si se cumple la condición de su argumento, de no ser así ejecuta el código correspondiente al else. El else es opcional.

```
if(condición)
{
      /*Código del if*/
}
else
{
      /*Código del else*/
}
```

For: La sentencia for ejecuta un ciclo controlado por un contador. Al entrar al ciclo el contador se inicializa al valor indicado en el argumento, se chequea la condición de salida, y luego de ejecutar el código se aumenta el contador en la cantidad indicada en el argumento, se chequea la condición y se repite el ciclo mientras que esta sea VERDADERA.

```
For(int i; condición; i+incremento)
{
      /*Código del ciclo*/
}
```

While: Esta sentencia permite ejecutar un ciclo controlado por una condición. El código del ciclo se ejecuta repetidamente hasta que la condición sea FALSA

```
While(condición)
```

```
{
      /*Código del ciclo*/
}
```

1.8 INSTRUCCIONES DECLARATIVAS

Las instrucciones declarativas son aquellas en las que se describen las características de la transformación o las características del resultado que con ellas se obtiene. A diferencia de las instrucciones procedimentales, que especifican el "como" se efectúa la operación, las instrucciones declarativas especifican el "que" hace la operación.

Ejemplo 1.4: Instrucciones declarativas en Prolog

En Prolog, un lenguaje declarativo, primero se declaran *Hechos,* que son proposiciones siempre verdaderas en el contexto del programa, en este ejemplo se declaran dos hechos: juan es el padre de maría y pedro es el padre de josé:

```
padre(juan, maría).
padre(pedro, josé).
```

Ahora, para averiguar quien es el padre de maría se usa una instrucción de consulta, nótese que el programador sólo indica QUE información desea consultar, y en ningún momento expresa el CÓMO se accede a esta.

```
?- padre( X, maría).
```

El intérprete de Prolog da la respuesta:

```
X=juan;
```

Nótese como este programa se asemeja a una base de datos. En efecto, los lenguajes de bases de datos como el SQL son lenguajes declarativos.

2 TIPOS DE DATOS

Un tipo de datos es una estructura matemática que agrupa un conjunto de elementos con características comunes, sobre los cuales se definen determinadas operaciones.

En esencia un tipo de dato reúne las cualidades comunes a un grupo de elementos u objetos que los distinguen en una clase o especie. Y consta de dos partes: el conjunto de valores y un conjunto de operaciones sobre dichos valores. Al conjunto de valores que incluye un tipo de dato se le conoce como *dominio.*

2.1 TIPOS DE DATOS NATIVOS

Los tipos de datos nativos son aquellos ofrecidos por un lenguaje de programación. Ya están implementados, con las operaciones básicas que se pueden aplicar sobre ellos.

Ejemplo 2.1: Algunos TD nativos del C++

Tipo	Nombre en C++	Algunas operaciones
Entero	int	Suma, resta, potencia, producto
Real	float, double	Suma, resta, truncar, raíz
Carácter	char	Mayúscula, minúscula

2.2 TIPOS DE DATOS DEFINIDOS

En ocasiones se utilizan datos propios de la aplicación en la que se está trabajando. Para agrupar estos datos de acuerdo a las funciones aplicables a cada uno de ellos y para abstraerse de sus detalles durante el proceso de desarrollo, el programador puede definir sus propios tipos de datos, distintos a los nativos, de acuerdo a sus necesidades. A estos tipos definidos por el programador se les llama tipos de datos definidos.

Ejemplo 2.3: Definición de un tipo de dato en C++.

Para definir un tipo de dato en C++ utilizando los tipos de datos nativos del lenguaje, se utiliza la palabra clave typedef con la sintaxis:

```
typedef <definición del tipo> <identificador>
```

El <identificador> es el nombre que se le quiere asignar al nuevo TD correspondiente a la <definición del tipo>. Para definir un nuevo tipo de dato llamado punt_int de manera que corresponda a los punteros a enteros se escribe el código:

```
Typedef * int punt_int;
```

2.2.1 Elemento estándar:

Un elemento estándar (*stdelement)* es un tipo de dato que tiene como característica distintiva un campo llamado *clave* que lo diferencia de todos los demás elementos de su tipo y lo hace único. Estos elementos son los más usados al definir tipos de datos para crear estructuras de datos con ellos.

2.3 VALORES

Se denomina valores al conjunto de datos manipulados en un proceso. Los valores son los elementos individuales que al agruparse de acuerdo a sus características y operaciones, forman un tipo de dato, por tanto, cada valor tiene un tipo correspondiente, y de acuerdo a ese tipo se determina su representación en memoria, su rango y precisión.

Ejemplo 2.4: Algunos valores de los tipos del C++

A continuación se listan algunos valores con su correspondiente tipo y representación en memoria.

Valor	Tipo	Rango	Tamaño en memoria
30	Int	-32768 a 32767	16 bits

150.324	Float	$3.4*10^{-38}$ a $3.4*10^{38}$	32 bits
A	Char	Caracteres ASCII	8 bits

2.4 FUNCIONES

Las funciones de los lenguajes de programación son agrupaciones de instrucciones que llevan a cabo una tarea específica. Algunas corresponden a la definición matemática de función: son relaciones que asignan a cada elemento del dominio (*datos de entrada)* un y sólo un elemento del rango (*resultados*) Toda operación matemática que se lleve a cabo en el computador (la suma, por ejemplo) se hace con una función de este tipo.

Sin embargo, en C++ pueden definirse funciones que no corresponden a la definición matemática: es posible, por ejemplo, definir funciones que no retornen ningún resultado y/o no tengan ningún argumento (*dato de entrada*) y sólo cambien el contenido de la memoria y/o invoquen algún dispositivo de hardware (es decir, provoquen un cambio de *estado*). Aquí la distinción entre las funciones matemáticas y las que sólo efectúan un cambio de estado, llamando a estas últimas eventos.

Ejemplo 2.5: Funciones en C++

Para declarar una función en C++ se utiliza la sintaxis:

```
Tipo_de_variable_ret nombre_funcion ( tipo_arg_1 nombre_arg_1, ... ,
tipo_arg_n nombre_arg_n)
{
            /*Código de la función*/
            return nombre_variable_retorno;
}
```

La siguiente función, no tiene argumentos ni devuelve ningún resultado, simplemente borra la pantalla.

```
void borrar_pantalla ()
{
      clrscr();
      printf("La pantalla está limpia");
}
```

La siguiente función equivale a una función matemática: toma dos valores enteros y los compara. Para cada par de argumentos devuelve UN solo valor según el primero sea mayor que el segundo o no:

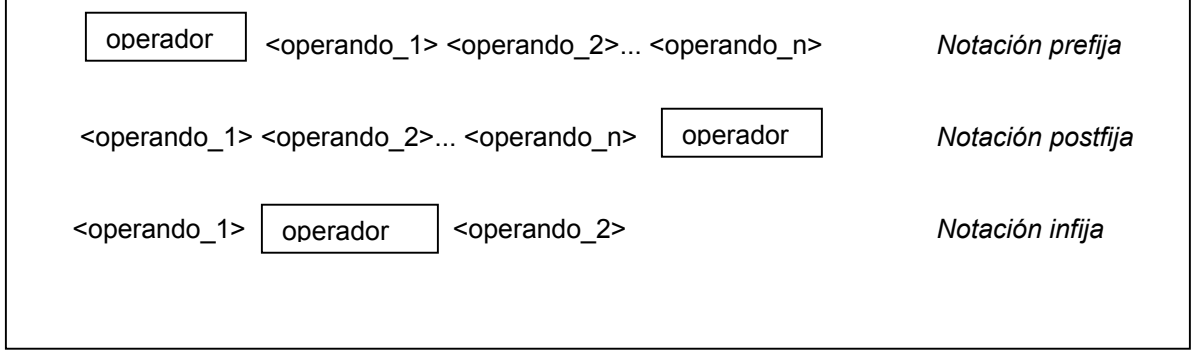
```
int compara( int a, int b)
{
      if(a>b)
      return 1;

      if(a<=b)
      return 0;
}
```

2.5 OPERADORES

Permiten aplicar una función predefinida por el lenguaje sobre uno o más valores de algún tipo nativo llamados operandos. Los operadores se representan por uno o más caracteres que se introducen antes, en medio o después de sus operandos como se ve en la figura 2.1.

Figura 2.1 *Posición de operadores*



Ejemplo 2.6: Algunos operadores en C++

Algunos operadores aritméticos:

Operador	**Aridad**	**Descripción**
-	Diádico	Resta
+	Diádico	Suma
*	Diádico	Multiplicación
/	Diádico	División

Operadores lógicos:

Operador	**Aridad**	**Descripción**
!	Monádico	Negación (NOT)
||	Diádico	O no exclusivo (OR)
&&	Diádico	Y (AND)

En el siguiente fragmento de código se utiliza el operador + para aplicar la función de suma a los operandos a y b de tipo entero. El resultado, 5, se guarda en la variable entera c:

```
int a=2;
int b=3;
int c;

c= a + b;
```

2.6 TIPOS DE DATOS ESCALARES

Son aquellos tipos de datos cuyos miembros están compuestos por un solo ítem (dato). Los tipos de datos escalares nativos son aquellos tipos de datos escalares que ya están implementados en el lenguaje junto a sus respectivas operaciones.

Ejemplo: 2.7: Tipos de datos escalares nativos de C++

Los siguientes tipos de datos de C++ son escalares:

Tipo de dato	Nombre en C++
Entero	Int
Real	Float o double
Carácter	Char
Booleano	Bool
Puntero	* <tipo_de_dato>

2.7 TIPOS DE DATOS AGREGADOS

Se denominan agregados a los tipos de datos cuyos valores están compuestos por más de un elemento de información (*dato*), a cada uno de estos elementos se les conoce como miembros o componentes.

2.7.1 Tipos de datos agregados nativos

Los lenguajes de programación por lo general incluyen varias construcciones para hacer composiciones de tipos de datos. A estas construcciones se les conoce como tipos de datos agregados nativos, de los cuales los más comunes son:

- **Estructuras:** Las estructuras son tipos de datos compuestos formados por varios datos llamados campos o miembros, cada uno de los cuales posee su propio espacio en memoria. Para declarar una estructura en C++ se utiliza la siguiente sintaxis:

```
struct nombre_de_estructura
{
      Campos_de_estructura;
};
```

Ejemplo 2.8: Estructuras

En C++, primero debe declararse la estructura para poder usarla:

```
struct estudiante
{
      char[20] nombre;
      char[20] apellido;
      float nota;
      int id;
} pedro;        /*pueden declararse datos de tipo estructura
```

```
                    escribiendo los nombres después de la llave y
                    antes del punto y coma final*/
```

Para declarar un dato de tipo estructura estudiante:

```
Struct estudiante juan;
```

Para acceder a cada uno de los campos del dato juan, se utiliza la sintaxis punto, por ejemplo, para asignarle una nota de 4.0 a juan:

```
juan.nota=4.0;
```

- **Arreglos:** Los arreglos son un conjunto de posiciones adyacentes de memoria para almacenar datos del mismo tipo que tienen el mismo nombre y se diferencian en el índice.

  Para declarar un arreglo en C++ se utiliza la sintaxis:

  Tipo_de_dato nombre_arreglo [*dimension*]

Ejemplo 2.9: Arreglos

Una vez declarado el arreglo, puede accederse a cada uno de sus componentes por medio del índice como se muestra a continuación (los índices comienzan en 0):

```
Int arreglo[10];

Arreglo[3]=5;     /*Asigna el valor 5 al 4to elemento del arreglo*/

Printf("el cuarto elemento del arreglo tiene el valor:
%i",arreglo[3]);
```

- **Uniones:** Las uniones tienen un aspecto similar a las estructuras en cuanto a cómo se definen y a su manejo, pero tienen una diferencia fundamental con respecto a estas: los miembros comparten el mismo trozo de memoria. El espacio que ocupa una unión en memoria es el espacio que ocupa el campo más grande. Para declarar una unión en C++ se utiliza la sintaxis:

```
union nombre_de_union
{
      Campos_de_union;
};
```

Ejemplo 2.10: Uniones

Se declara la unión persona y una variable llamada p y se le asigna el nombre juan:

```
union persona
{
      char nombre[20];
      char inicial;
```

```
    } p;

p.nombre="juan";
```

No se ha asignado ningún valor a p.inicial pero como los campos de la unión comparten el mismo espacio en memoria, el valor actual de p.inicial es j (el primer carácter de la variable p.nombre).

- **Clases:** Para los propósitos de este curso, las clases pueden definirse como *estructuras* en las cuales, además de los campos (que en el caso de las clases se denominan atributos), se declaran las funciones (que se conocen como métodos) que pueden llevarse a cabo sobre el tipo de dato representado por la clase.

  Para definir una clase en C++ se utiliza la siguiente sintaxis:

```
class nombre_de_clase
{
     private:
          Campos_de_clase;
          ...
     public:
          Constructor;
          Declaración_de_funciones;
          ...
};
```

  La palabra private significa que los campos no podrán accederse directamente desde el exterior de la clase, para hacer esto, deben usarse funciones.
  La palabra public significa que las funciones de la clase podrán accederse directamente desde otras clases. El *Constructor* es simplemente una función usada para inicializar los campos de la clase, esta función no retorna ningún valor y siempre tiene el mismo nombre que la clase.

  Dentro de la definición de la clase se *declaran* las funciones, esto significa que se escribe sólo el prototipo de la función y no su código. Las funciones se *definen* (se les escribe su código) fuera de la clase. Todos estos aspectos se ilustran en el siguiente ejemplo:

Ejemplo 2.11: Clases

A continuación se definirá la clase (tipo de dato) punto, que tiene un color (representado por un entero) y una coordenada en X y Y:

```
class punto
{
  public:
      int color;
      int coordenada_x;
      int coordenada_Y;

  public:
      punto(int color, int X, int Y);
      void cambiar_color(int nuevo_color);
```

```
        int consultar_color();
    };
```

Ahora se definen las funciones de la clase punto:

```
punto::punto(int color1, int X, int Y)
{
        color=color1;
        coordenada_X=X;
        coordenada_Y=Y;
}

void punto::cambiar_color(int nuevo_color)
{
        color=nuevo_color;
}

color punto::consultar_color()
{
        return color;
}
```

Ahora, se crea un punto llamado alpha, de color 23 y posición (100, 80), luego se le cambia el color a 50, nótese que para invocar alguna función de la clase, se usa la notación "." de manera similar a como se usa para acceder a los campos de una estructura:

```
punto alpha( 23, 100, 80 );
alpha.cambiar_color( 50 );
```

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 1.1 hasta 1.4

[FA1997] Arango I. Fernando, "Elementos Fundamentales de los Lenguajes del Computador", Trabajo de Promoción a Profesor Asociado, Universidad Nacional de Colombia Sede Medellín, 1997.
1: Capítulos 1,2,5

[JO1999] Joyanes Aguilar. Luis, Zahonero Martínez. Ignacio, "Estructura de Datos: Algoritmos, Abstracción y Objetos", McGrawHill, 1999.
1: Capitulo 3: Abstracción de Datos: Tipos Abstractos y Objetos.

[VL1988] Villalobos S. Jorge, Quintero G. Alejandro, Otero D. Mario, "Estructuras de Datos: Un Enfoque desde Tipos Abstractos", Ediciones Uniandes, Agosto 1988.
1: Capitulo 1: Tipos Abstractos de Datos.

B TALLER

Ejercicios página 11, 21 y 27 libro de Stubbs y Webre

C LABORATORIO

Implementar en C++ un tipo de dato agregado propio de cada estudiante, definiendo sus campos y funciones miembro en una clase.