Estructura de Datos y Algoritmos 1

Práctico #1:

Introducción al curso

Introducción al Lenguaje C/C++

Introducción al Framework de Ejercicios

Objetivos generales del curso

Presentar al estudiante y trabajar en:

- estructuras de datos y algoritmos esenciales para la resolución de problemas en computación, de manera eficiente;
- el rol de la abstracción en el desarrollo de sistemas de porte mediano.

Este curso profundiza conceptos de programación vistos en cursos previos (por ejemplo en: prog1, prog2, lógica y fundamentos) y sienta las bases para cursos posteriores.

Objetivos específicos del curso

- Algoritmos iterativos y recursivos.
- Análisis de algoritmos: la estimación y cálculo del tiempo de ejecución de los algoritmos y el análisis de la eficiencia en espacio de almacenamiento
- Estructuras de datos básicas de la programación
- Conceptos de abstracción de instrucciones y de datos (tipos abstractos de datos)

El curso hace especial énfasis en el rol de la abstracción de datos en el diseño y el desarrollo, y la aplicación de estructuras de datos.

Metodología

- Clases Teóricas
- Clases Prácticas y de Laboratorio
- Consultas semanales con ayudantes de cátedra y docentes del curso
- Evaluaciones
  - Exámenes Parciales.
  - Entregas obligatorias de ejercicios prácticos.

Ver y consultar periódicamente el sitio web oficial del curso en **Aulas**

Entregas, Evaluaciones

Evaluación	Fecha entrega/evaluación	Puntos
Ejercicios 1	Por AULAS hasta las 21:00 1. 24/Abril (Entrega parcial opcional) 2. 06/Mayo (Entrega oficial)	10
Parcial 1	08/Mayo (Presencial)	30
Ejercicios 2	Por AULAS hasta las 21:00 1. 19/Junio (Entrega oficial)	15
Parcial 2	01/Julio (Presencial)	45

Tópicos del curso (I)

- Introducción:
  - Nociones generales. Abstracción en programación: particiones - refinamientos. Abstracción procedural e introducción a la abstracción de datos
- Análisis de Algoritmos
  - Eficiencia en espacio de almacenamiento y tiempo de ejecución. Orden de tiempo de ejecución del peor caso y caso promedio. Cálculo de tiempo de ejecución para algoritmos iterativos y recursivos
- Inducción y recursión
  - Recursión en un sistema computacional. Stack de ejecuciones. Análisis de iteración vs recursión.
  - Definición de tipos de datos inductivos.
  - Programación recursiva: Precondiciones, tipos, aplicaciones.

Tópicos del curso (I)

- Estructuras Dinámicas:
  - Estructuras estáticas y estructuras dinámicas. Punteros y manejo de memoria dinámica. Definición de estructuras lineales y no lineales (árboles). Algoritmos sobre dichas estructuras.
- Introducción a Tipos Abstractos de Datos (TADs)
  - Abstracción procedural y abstracción de datos.
  - Especificación de TADs: uso de pre y post condiciones. Implementación y uso de TADs. Ventajas de la programación con TADs.
- TADs Fundamentales
  - Especificación e implementación eficiente de TADs fundamentales: Listas, Pilas, Colas, Conjuntos, Diccionarios y Tablas. Variantes. Aplicaciones

Cronograma del curso

Estructuras de Datos y Algoritmos 1

- Introducción al análisis de algoritmos.
- Recurrencia - Análisis de algoritmos recursivos.
- Punteros y estructuras dinámicas: lineales y arborescentes.

TADs:
especificaciones,
implementaciones eficientes,
usos y
aplicaciones

Parcial 1

Parcial 2

Obligatorios / Ejercicios

Materiales bibliográficos

- Materiales en AULAS
  - Guías de clases teóricas, Lecturas, Videos, Ejercicios, Foros

- Libros básicos
  - **Cómo Programar en C/C++**, H.M Deitel & P.J. Deitel
  - **Data Structures and Algorithm Analysis in C and C++**, Mark Allen Weiss
  - **Estructuras de Datos y Algoritmos**, A. Aho, J.E. Hopcroft & J.D. Ullman

Lenguaje de programación

- Se usará C++, sin incluir la parte del lenguaje referida al paradigma orientado a objetos propiamente
- Se usarán esencialmente las construcciones de C pero incorporando algunos elementos de C++, que se irán introduciendo a medida que se desarrollen los temas del curso.
- El objetivo principal es trabajar con abstracciones y estructuras de datos y algoritmos, y no enseñar un lenguaje.
- Entorno de desarrollo (IDE): Microsoft Visual Studio

Introducción a C/C++

1. Estructura de un programa C++
2. Tipos de datos primitivos. Variables
3. Operadores
4. Instrucciones de Entrada/Salida
5. Estructuras de Control Alternativas
6. Estructuras de Control Repetitivas
7. Vectores (Arrays unidimensionales)

Estructura de un programa C++

```
//inclusiones de módulos necesarios
//declaraciones globales
int main() /* Función principal, donde comienza la
              ejecución. Debe existir solo una por
              proyecto */
{
 //declaraciones de variables y constantes
 //instrucciones
  return 0; /* Respuesta del programa al Sistema
               Operativo. Por defecto es cero */
}
```

Tipos de Datos. Variables

Los ***tipos*** determinan el conjunto de valores de los datos, así como las operaciones que se pueden hacer con ellos.

Tipo	Valores que representa	Tamaño en memoria
`char`	caracteres (‘a’, ‘b’, …)	1 byte
`short`	Números enteros	2 bytes
`int`	Números enteros	4 bytes
`long`	Números enteros	8 bytes
`float`	Números reales	4 bytes
`double`	Números reales	8 bytes
`bool`	true, false	1 byte

Declaración de variables

Introduce nuevas variables al programa. Se especifica el nombre de la variable y el tipo de dato correspondiente

```
int main()
{
   char c;
   int a, b;

   …
   return 0;
}
```

“Once a programmer has understood the use of variables, he has understood the essence of programming”

Edsger Dijkstra
(Holanda 1930 - 2002)

Operadores

Aritméticos

Operador	Significado	Tipos
+	Suma	int, char, float, double
-	Resta	int, char, float, double
*	Multiplicación	int, char, float, double
/	División	int, char, float, double
%	Resto de la división	int, char
++	Incremento	int, char, float, double
--	Decremento	int, char, float, double

Operadores ++ y --

Permiten incrementar y decrementar variables. Es un operador unario cuya notación puede ser postfija o prefija

```
int main()
{
  int a, b;
  b = a++; //Toma el valor de a, lo asigna y lo
           //incrementa en 1
  b = ++a; //Toma el valor de a, lo incrementa
           //en 1 y lo asigna
  return 0;
}
```

Operadores de asignación

Permite almacenar un valor en una variable

```
<operando> = <expresión>
```

Puede utilizarse como parte de una declaración de variables:

```
int a = 5;
int b = a;
```

Se pueden combinar con operadores aritméticos:

```
+=, -=, *=, /=, %=
```

`c += 3` es equivalente a `c = c + 3;`

Operadores

Relacionales o de comparación

Operador	Ejemplo	Significado
`==`	`x == y`	`x` es igual a `y`
`!=`	`x != y`	`x` es distinto de `y`
`<`	`x < y`	`x` es menor que `y`
`>`	`x > y`	`x` es mayor que `y`
`<=`	`x <= y`	`x` es menor o igual que `y`
`>=`	`x >= y`	`x` es mayor o igual que `y`

Operadores

Lógicos (aplica a tipos bool)

Operador	Ejemplo	Significado
`!`	`!A`	No se cumple **A**
`&&`	`A && B`	Se cumplen **A** y **B**
`||`	`A || B`	Se cumple **A** o **B**

Instrucciones de Entrada/Salida

Se utilizan dos objetos: **cin** y **cout**. Ambos se refieren a flujos de entrada y salida.
La secuencia de símbolos **>>** se utiliza como operador de entrada al objeto **cin**.

```
int main()
{
   int i;
   cin >> i;      //Lee un valor entero y lo
                  //asigna a la variable i
   double d;
   cin >> d;      //Lee un valor real y lo
                  //asigna a la variable d
}
```

Instrucciones de Entrada/Salida

La secuencia de símbolos **<<** es utilizada como operador de salida al objeto **cout**

```
int main()
{
   int a = 10;
   cout << a;    //Imprime en pantalla el valor
                 //de a
}
```

Entrada/Salida. Ejemplo

```
int main()
{
  int ladoa, ladob, ladoc, perimetro;
  // Entrada de datos
  cout << "Valor de lado a: ";
  cin >> ladoa;
  cout << "Valor de lado b: ";
  cin >> ladob;
  cout << "Valor de lado c: ";
  cin >> ladoc;
  // Calculo del perimetro
  perimetro = ladoa + ladob + ladoc;
  // Muestra de los resultados
  cout << "Perimetro del triangulo es: " <<
          perimetro << endl;
  return 0;
}
```

Estructuras de control alternativas

```
if (<condición>)
{
    <Lista_de_instrucciones>
}
else
{
    <Lista_de_instrucciones>
}
```

Estructuras de control alternativas

```
switch (<Expresión>)
{
      case <constante> : <instrucciones>
      case <constante> : <instrucciones>
      ...
      case <constante> : <instrucciones>
      [default: <instrucciones>]
}
```

Se recomienda el uso de la instrucción **`break`** al final de cada lista de instrucciones.

Estructuras de control repetitivas

Precondicional

```
while (<condición>) {
      <instrucciones>
}
```

Postcondicional

```
do {
      <instrucciones>
}
while (<condición>)
```

Estructuras de control repetitivas

Ciclo for

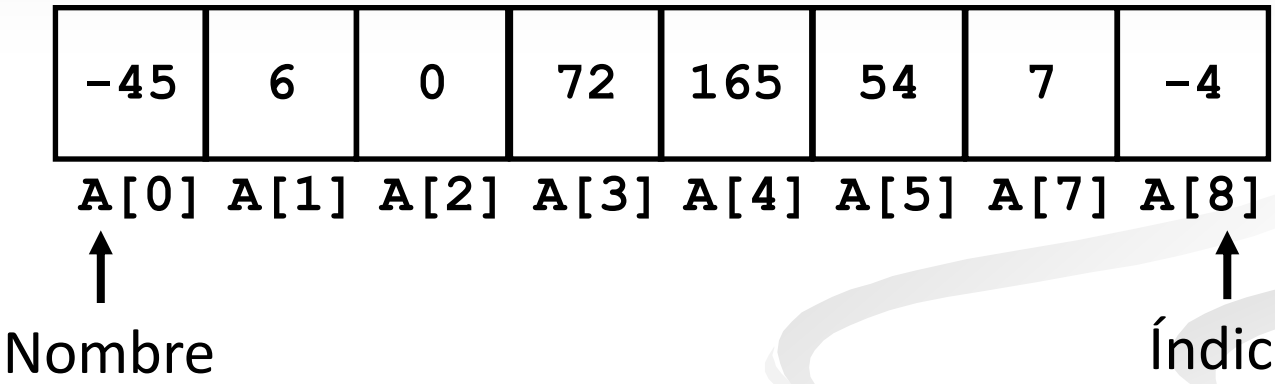
```
for (<enunciado1>; <condición>; <enunciado2>)
{
      <instrucciones>
}
```

Ejemplo: Cálculo de Promedio

```
int main()
{
  int N;
  cout << "Cantidad de valores: ";
  cin >> N; cout << endl;
  int valor, suma = 0;
  for (int i = 1; i <= N; i++){
      cout << "Entre el valor #" << i;
      cin >> valor; cout << endl;
      suma += valor;
  }
  float promedio = suma/N;
  cout << "Promedio: " << promedio << endl;
  return 0;
}
```

Vectores (arrays unidimensionales)

Es un grupo de posiciones en memoria que almacena bajo el mismo nombre (identificador) a una colección de datos del mismo tipo.



Vectores (arrays unidimensionales)

Declaración y creación

```
<tipo>* <nombre> = new <tipo>[<longitud>];
```

Ejemplo

```
int* puntuaciones = new int[30];
```

Importante: En C++ los vectores creados mediante **new** deben ser destruidos cuando ya no se usarán más.

```
delete [] puntuaciones;
```

Vectores (arrays unidimensionales)

Acceso a un elemento

```
<nombre>[<indice>]
```

El índice comienza en 0 y termina en N-1 siendo N la longitud del vector.
El índice puede ser cualquier expresión entera.

Ejemplos

```
int a = A[0];
cout << A[i+1];
A[i] = A[i-1] + 2;
```

Ejercicio

Hacer un programa que lea un entero largo y calcule la cantidad de ocurrencias de cada dígito (0..9)

Idea para la solución: Construir un vector de ocurrencias de cada dígito. El vector tendría longitud 10.

Ocurrencias de cifras

```
int main() {
  int* ocurrencias = new int[10];
  unsigned long nro;
  cout << "Ingrese el número: ";
  cin >> nro; cout << endl;
  while(nro != 0){
    int dig = nro % 10;
    nro = nro / 10;
    ocurrencias[dig]++;
  }
  cout << "Cantidad de ocurrencias" << endl;
  for (int i = 0; i < 10; i++)
      cout << "Dígito " << i << ": "
           << ocurrencias[i] << endl
  delete [] ocurrencias;
  return 0;
}
```