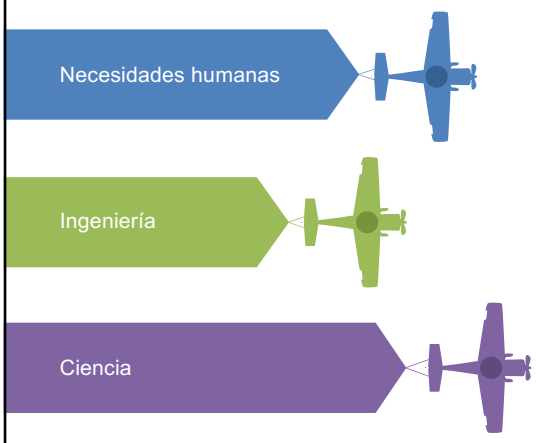
Introducción al curso

INF2240 – Estructura de datos
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

7

Ciencia, tecnología e ingeniería



Lo primero que se debe plantear en un curso ligado a tecnologías e ingeniería es: ¿Porqué son importantes las tecnologías de información para la Ingeniería?

8

8

Tecnologías de información y comunicación

- Permiten procesar y transmitir información.
- Estudio, diseño, desarrollo, soporte y gestión de sistemas de información:
  - software (programas) y hardware (equipos).
- Manipulación de datos para la generación de información, mediante:
  - conversión, almacenamiento, protección, transmisión y recuperación en forma segura.

9

9

Sistema

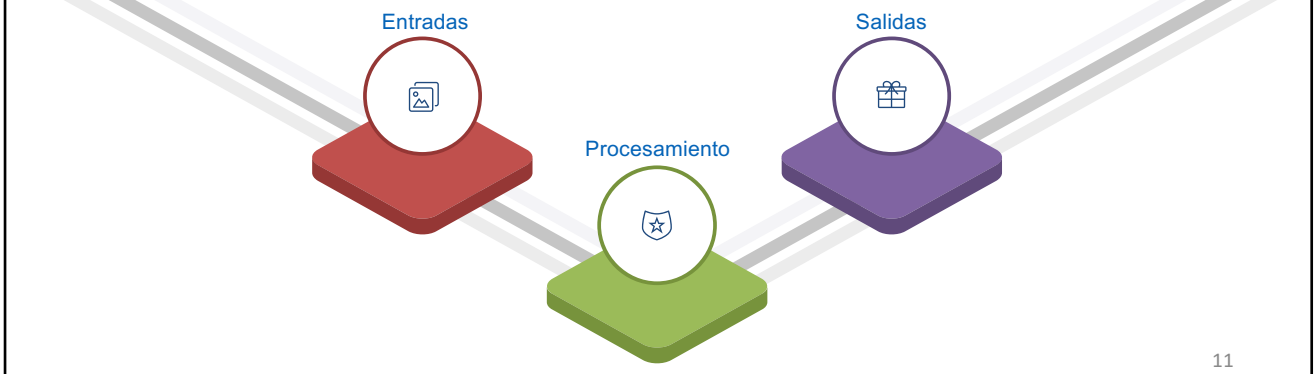
- Objeto con cierto grado de complejidad, sus componentes se relacionan con al menos algún otro componente.
- Un sistema puede ser material, conceptual, natural o artificial.
- Todos los sistemas deben considerar su composición, estructura y entorno.

10

10

Especificación de problemas

- Los problemas siempre se piensan enfocados en entregar soluciones, y actualmente asociadas fuertemente al manejo de información.



11

11

Técnicas algorítmicas para resolver problemas

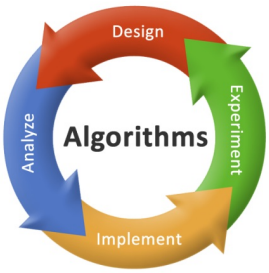
- **Algoritmos de fuerza bruta**: toman la ruta más evidente o corta para resolver el problema, independiente de que sea buena o no.
- **Dividir y conquistar:** la solución global es la unión de todas las soluciones parciales.
- **Programación dinámica:** es similar al anterior, pero busca reutilizar los resultados obtenidos de los sub-problemas.
- **Otros:** algoritmos voraces, algoritmos probabilistas.

12

12

Análisis de algoritmos

- Busca establecer la calidad de un programa y compararlo con otros que resuelvan el mismo problema, sin necesidad de desarrollarlos.
- Permite evaluar el diseño de las estructuras de datos de un programa, midiendo su eficiencia.
- Se basa en: Las características estructurales del algoritmo que respalda al programa.



13

13

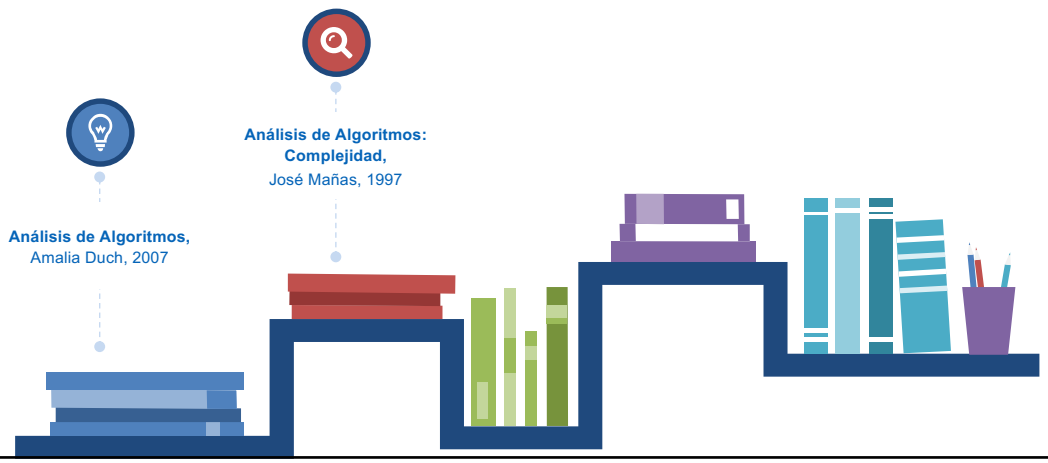
Eficiencia

- Espacio de memoria que el algoritmo requiere:
  - Número y tamaño de las variables y estructuras de datos.
- Tiempo de ejecución del algoritmo (tiempo de cómputo) considerando:
  - La estructura del algoritmo, número de operaciones elementales que deben ser realizadas durante la ejecución del algoritmo.
  - La velocidad de operación del computador en que se ejecuta.
  - El compilador utilizado.
  - Tamaño de los datos de entrada con los que el programa trabaja.

14

14

Lecturas del módulo



15

15

Tipo abstracto de dato

INF2240 – Estructura de datos
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

16

Datos, información y conocimiento

- Antes de definir un TDA es necesario definir una serie de conceptos elementales:
  - Información.
  - Datos.
  - Conocimiento.

17

17

Datos, información y conocimiento

- Datos: Son los hechos que describen sucesos y entidades.
  - Basados en símbolos: letras del alfabeto, números, movimientos de labios, puntos y rayas, señales con la mano, dibujos, etc.
  - No contienen información.
  - No tienen la capacidad de comunicar un significado
  - No pueden afectar el comportamiento de quien los recibe (por lo anterior).

18

18

Datos, información y conocimiento

- Información: Es una colección de hechos significativos y pertinentes para el organismo que los percibe.
  - Datos Significativos: Símbolos reconocibles, estar completos y expresar una idea no ambigua.
  - Datos Pertinentes: Pueden ser utilizados para responder a preguntas propuestas.

19

19

Datos, información y conocimiento

– Conocimiento: Es la toma de conciencia y comprensión de un conjunto de hechos significativos y pertinentes para el organismo que los percibe y cómo estos pueden ser útiles, basándose en la experiencia y juicio de dicho organismo.

20

20

Datos, información y conocimiento

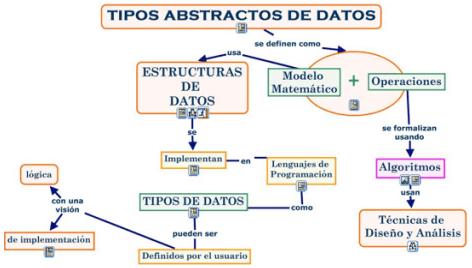
Categorías de información: ¿A quién va dirigida?, ¿Para quién es útil?

21

21

Tipo abstracto de dato

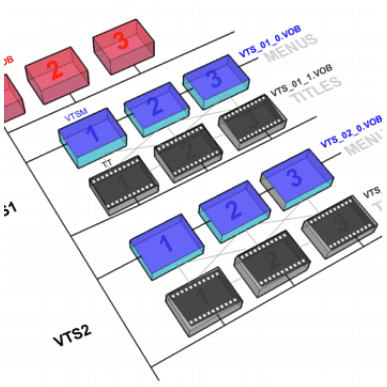
"Un TDA es un ente cerrado y autosuficiente, que no requiere de un contexto específico para que pueda ser utilizado en un programa, lo que garantiza portabilidad y reutilización del software y minimiza los efectos que puedan producir un cambio al interior del TDA".



22

22

Clasificación de las estructuras de dato

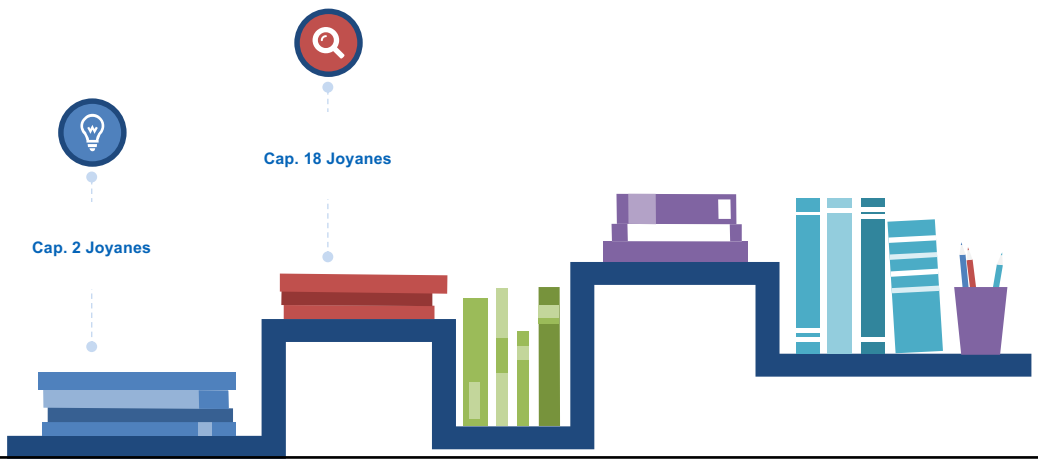


- Estructura de datos **estática** : cantidad fija de memoria para esa estructura antes de la ejecución del programa.
- Estructura de datos **dinámica** cuando se le asigna memoria a medida que es necesitada, durante la ejecución del programa.

23

23

Lecturas del módulo



24

24

Arreglos y matrices

INF2240 – Estructura de datos
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

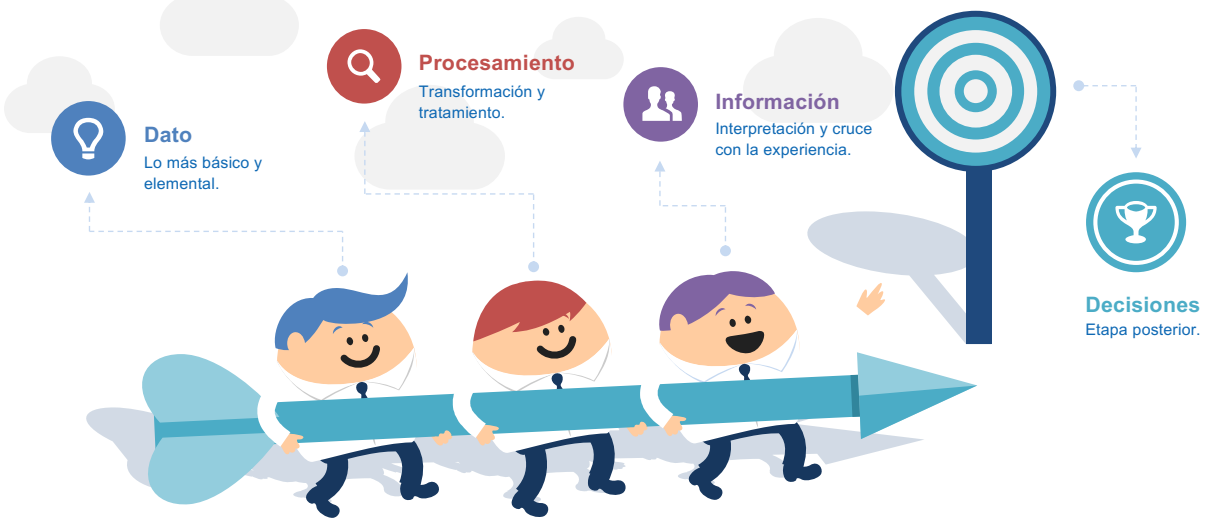
25

Estructuras de dato

- ¿Qué se entiende por estructura de dato?
  - Generalmente se de entiende como estructura de dato a la forma de organizar los datos que se utilizarán.

26

Información



27

27

Estructuras de dato

- ¿Qué es una estructura de dato?
- Una colección de datos, donde tiene algunas características:
  - Una determinada organización
  - Un número de operaciones asociadas
- Estas colecciones de datos siempre están relacionados entre si (existe homogeneidad), como por ejemplo un listado de puntajes PSU, donde todos estos puntajes tienen la característica principal de ser tipo float.

28

28

Estructuras estáticas

- Siempre se debe recordar que en una estructura de datos estática:
  - El tamaño ocupado en memoria se define antes de que el programa se ejecute
  - Este tamaño no puede modificarse durante la ejecución del programa, ¿porqué?, ¿existe alguna forma de cambiar el tamaño a las estructuras definidas?
- Entre las estructuras de datos estáticas se encuentran:
  - Arrays (vectores y matrices), Registros, Archivos, Cadenas.

29

29

Arreglos unidimensionales

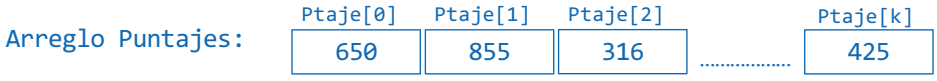
- Así, se puede definir a un array como un conjunto finito y ordenado de elementos homogéneos, donde:
  - **Finito**: tiene un tamaño determinado.
  - **Ordenado**: cada elemento puede ser identificado.
  - **Homogéneo**: son del mismo tipo de datos.
- ¿Qué operaciones se pueden realizar con arrays?

30

30

Arreglos unidimensionales

- Un arreglo unidimensional o vector es el tipo de arreglo más simple.

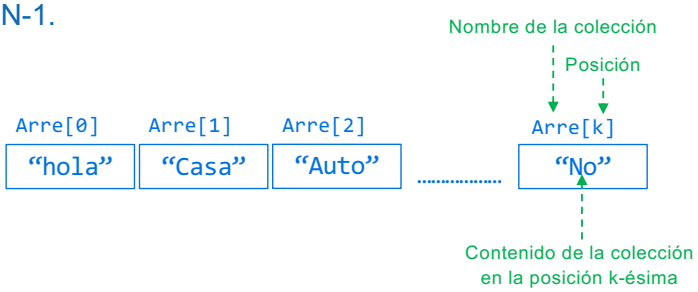


31

31

Arreglos unidimensionales

- Recordar que:
  - Cada arreglo se reconoce por un identificador y cada dato se almacena en una posición indexada.
  - Un arreglo de largo N, tiene posiciones indexadas mediante enteros desde 0 hasta N-1.



32

32

Arreglos unidimensionales

```c
#include <stdio.h>
#define limite 10

main()
{
        /*Declaracion*/
        int arregloDeEnteros[limite];
        int i;

        /*Llenado de la coleccion*/
        for(i=0; i<limite; i++)
        {
                printf("Ingrese un valor para llenar el arreglo:");
                scanf("%d", &arregloDeEnteros[i]);
        }

        /*Despliegue de la coleccion*/
        for(i=0; i<limite; i++)
        {
                printf("[%d]", arregloDeEnteros[i]);
        }
}
```

Código en Github
https://goo.gl/ClO8DX

33

33

Eliminación en un arreglo

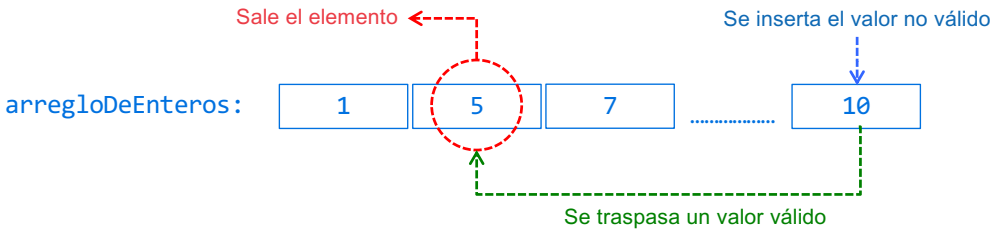
1. **Definición de un valor no válido,** depende del problema.
2. **Especificación del arreglo,** ultimo valor real y váñido
3. **Identificar posición,** se debe modificar.
4. **Ejecutar borrado,** realmente se borra?

34

34

Arreglos unidimensionales

– Entonces, ¿Cómo se eliminaría un elemento de la posición *k-ésima* en el arreglo del ejemplo anterior?



35

35

Arreglos bidimensionales

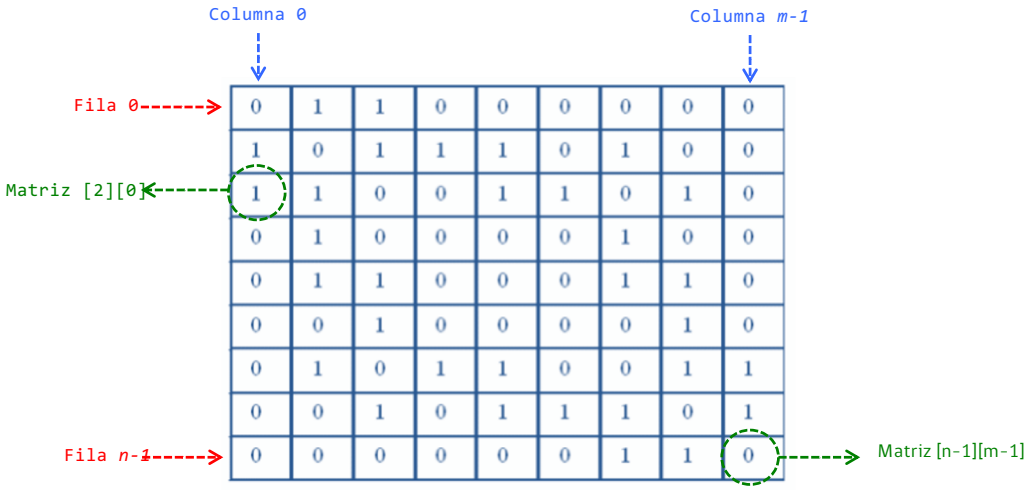
– ¿Qué es un arreglo bidimensional?
  – Conjunto de elementos homogéneos y ordenados.
  – Se necesita explicitar dos subíndices para poder identificar cada elemento del arreglo.
  – Se conoce a los arreglos bidimensionales con el nombre de matrices.
  – Alguna de las operaciones que se pueden realizar con matrices son: asignación, lectura, escritura, recorrido, actualización (añadir, borrar, insertar), ordenación y búsqueda.

36

36

Arreglos bidimensionales

Sea Matriz[n][m]



37

37

Arreglos bidimensionales

```c
#include <stdio.h>
#define cantFilas 10
#define cantColumnas 15

main()
{
	/*Declaracion*/
	int matrizDeEnteros[cantFilas][cantColumnas];
	int i, j;

	/*Llenado de la coleccion*/
	for(i=0; i<cantFilas; i++)
	{
		for(j=0; j<cantColumnas; j++)
		{
			printf("Ingrese un valor para llenar la matriz:");
			scanf("%d", &matrizDeEnteros[i][j]);
		}
	}

	/*Despliegue de la coleccion*/
	for(i=0; i<cantFilas; i++)
	{
		for(j=0; j<cantColumnas; j++)
		{
			printf("[%d]", matrizDeEnteros[i][j]);
		}
	}
}
```

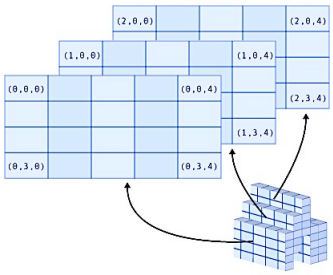
Código en Github
https://goo.gl/JBoF1B

38

38

Arreglos multidimensionales

- ¿Cuántas dimensiones se pueden definir para un arreglo?
  - Si se declara un arreglo k-dimensional se debe explicitar los k subíndices que lograrán las operaciones básicas sobre este arreglo multidimensional.
- Las operaciones que se pueden realizar con matrices multidimensionales son las mismas que con los arreglos.



39

39

Actividad personal

Una empresa de predicción meteorológica debe manejar los nombres de cada región del país.

Por cada una de las regiones (r) debe contener una cantidad de estaciones (e) las cuales tienen una cantidad igual de mediciones por cada estación (m).

Se solicita que guarde las mediciones de cada estación, muestre su promedio y el nombre de la estación con mayor cantidad de mediciones de precipitaciones.

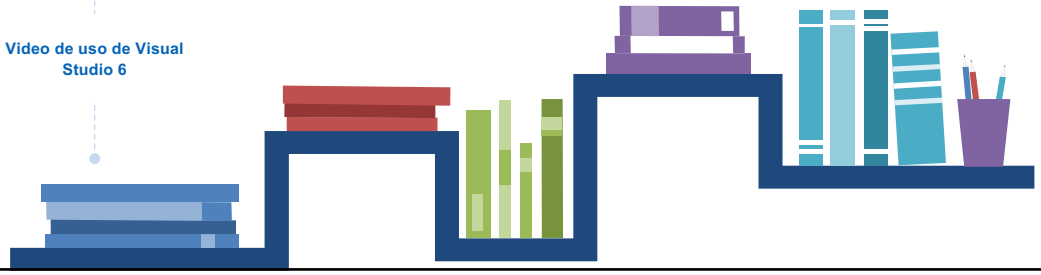
Resuelva el problema usando arreglos y matrices.

40

40

Lecturas del módulo

Video de uso de Visual Studio 6



41

Especificación sobre arreglos

INF2240 – Estructura de datos
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

42

Arreglos compactos

- Un arreglo compacto es en aquel que lógicamente se debe asegurar que los datos siempre estén juntos.
- Dependiendo del problema la compactación puede ser manejada con variables hacia los índices o `null`.
- Se debe considerar los casos de agregar o eliminar para siempre mantener el arreglo compacto.

43

43

Eliminación en un arreglo compacto

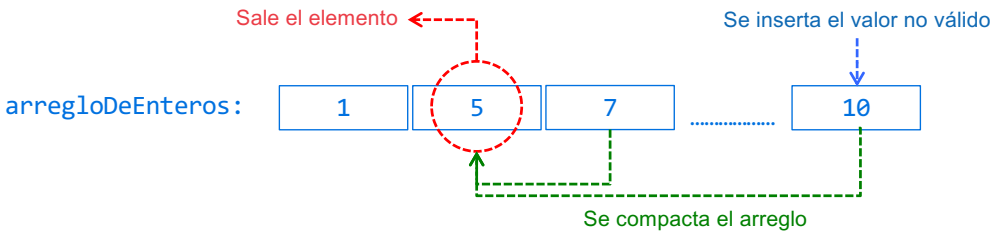
1. **Definición de un valor null,** depende del problema.
2. **Especificación del arreglo compacto,** con plibre o ult (se verá posteriormente)
3. **Identificar posición,** se debe modificar.
4. **Ejecutar borrado,** realmente se borra?

44

44

Eliminación en un arreglo compacto

– Entonces, ¿Cómo se eliminaría un elemento de la posición *k-ésima* en el arreglo del ejemplo anterior?



45

45

Lógica de control

- Puede existir variadas formas lógicas de controlar el funcionamiento y reglas de comportamiento de una colección, en este caso mencionaremos dos atingentes a los arreglos y matrices:
  - plibre
  - ultPosicion

pLibre

- Corresponde a una variable de tipo `int` y que representa la primera posición libre que se encuentra dentro del arreglo.
- Es utilizada bajo la lógica de los arreglos compactos.
- Algunas características son:
  - Siempre cuando la colección está vacía parte en cero.
  - Para su manejo es importante compactar los arreglos.

pLibre

```c
#include <stdio.h>
#define tam 100

main()
{
        int pLibre=0, i, arreglo[tam];

        /*Se agregaran 3 elementos*/
        for(i=0; i<3; i++)
        {
                printf("Ingrese un valor:");
                scanf ("%d", &arreglo[i]);
                pLibre++;
        }

        /*Se muestran los valores usando pLibre*/
        for(i=0; i<pLibre; i++)
        {
                printf("%d", arreglo[i]);
        }
}
```

Código en Github
https://goo.gl/CPNm6p

48

48

ultPosicion

- Corresponde a una variable de tipo `int` y que representa la última posición ocupada que se encuentra dentro del arreglo.
- Es utilizada bajo la lógica de los arreglos compactos.
- Los arreglos que manejan `ultPosicion` se conocen como listas contiguas.
- Algunas características son:
  - Siempre cuando la colección está vacía parte en -1.
  - Para su manejo es importante compactar el arreglo.

49

49

ultPosicion

```c
#include <stdio.h>
#define tam 100

main()
{
        int ultValor=0, i, arregloDeValores[tam];

        /*Se agregaran 3 elementos*/
        for(i=0; i<3; i++)
        {
                printf("Ingrese un valor:");
                scanf ("%d", &arregloDeValores[i]);
                ultValor++;
        }

        /*Se muestran los valores usando ultValor*/
        for(i=0; i<=ultValor; i++)
        {
                printf("%d", arregloDeValores[i]);
        }
}
```

Código en Github

https://goo.gl/oD6tTW

50

50

Punteros

INF2240 – Estructura de datos
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

51

Punteros

- Un puntero es una variable que representa la posición (en vez del valor) de otro dato, tal como una variable o un elemento de un arreglo.
- Los punteros son usados generalmente para traspasar información entre una función y sus puntos de llamada.
- Proporcionan una forma de devolver varios valores desde una función a través de los argumentos de la función.
- Permiten que referencias a otras funciones puedan ser especificadas como argumentos de una función.

52

52

Punteros

Relación entre `pv` y `v`, donde `pv=&v` y `v=*pv`

53

Operadores

- Se distinguen dos tipos de operadores:
  - & : Operador de dirección o referencia. Permite obtener la dirección de memoria en que se halla ubicada. Lo hemos visto en la función scanf().
  - * : Operador de indirección. Cuando se aplica a un puntero es posible acceder al contenido de lo apuntado por dicho puntero.

Relación entre `pv` y `v`, donde `pv=&v` y `v=*pv`

54

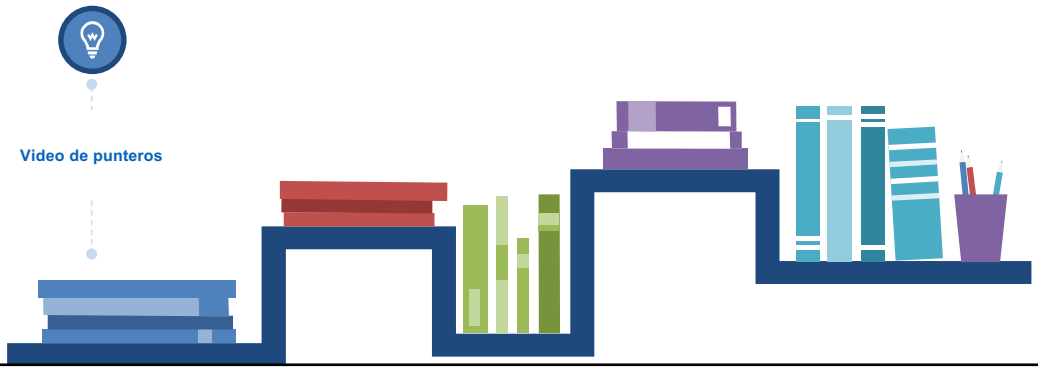
Ejemplos

Caso	Resultado	Estado
`int i, j, *p;`	p es un puntero a int	Correcto
`p = &i;`	p contiene la dirección de i	Correcto
`*p = 10;`	i toma el valor 10	Correcto
`p = &j;`	p contiene ahora la dirección de j	Correcto
`*p = -2;`	j toma el valor -2	Correcto
`p = &34;`	las constantes no tienen dirección	Incorrecto
`p = &(i+1);`	las expresiones no tienen dirección	Incorrecto
`&i = p;`	las direcciones no se pueden cambiar	Incorrecto

55

55

Lecturas del módulo



56

56

Punteros en las colecciones

INF2240 – Estructura de datos
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

57

Definición de un struct

- Antes de declarar variables del nuevo tipo estructura es necesario definir los campos (datos) que contendrá la estructura y,
- asignarle un nombre para identificarla como un nuevo tipo de dato.
- Para ocupar el nuevo tipo de dato definido es necesario declarar variables de ese tipo.

58

58

Definición de un struct

- La declaración de variables puede hacerse de dos formas:

```c
#include <stdio.h>

struct Persona
{
	char nombre[30];
	char rut[11];
	char telefono[10];
	int fechaNacimiento;
	int estatura;
	int peso;
}cliente 1, cliente2;
```

```c
#include <stdio.h>

struct Persona
{
	char nombre[30];
	char rut[11];
	char telefono[10];
	int fechaNacimiento;
	int estatura;
	int peso;
};

struct Persona cliente3, cliente4;
```

59

59

Operadores sobre struct

- Acceso a campos
  - Es importante destacar que los campos de un struct deben ser definidos y pensados de forma funcional:

```
variableTipoStruct.campoDelStruct
```

- Declaración de colecciones – arreglos –

```
struct nombreStruct declaraciónVariableAtomica;
struct nombreStruct declaraciónColeccion[tamaño];
```

- Acceso en colecciones

```
Nombrecolección[indice].campo
```

60

60

Punteros a struct

- Se pueden definir también punteros a estructuras:

```
struct alumno *pt;
pt = &nuevo_alumno;
```

- Ahora, el puntero pt apunta a la estructura nuevo_alumno y esto permite una nueva forma de acceder a sus miembros utilizando el operador flecha:

```
pt->telefono;
(*pt).telefono;
```

- Con ello, el paréntesis es necesario por la mayor prioridad del operador (.) respecto a (*).

61

61

Uso de char*

- Cuando se manejan datos en la realidad (en las empresas, en el gobierno, en el mundo real), se desconoce el tamaño de las palabras (nombres, direcciones, etc), de esta forma, se debe trabajar con memoria dinámica.
- La memoria dinámica en strings se implementa utilizado char *.
- Para ello véase el siguiente ejemplo, donde se tiene la defición de un struct, y se debe proceder al llenado de un vector de struct tipo person.

62

62

Uso de char*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define cantidad 5

struct Persona
{
	char *nombre;
	char *carnetIdentidad;
	int edad;
};

struct Persona personas[cantidad];

main()
{
	→
```

Código en Github
https://goo.gl/jIDE9n

63

Uso de char*

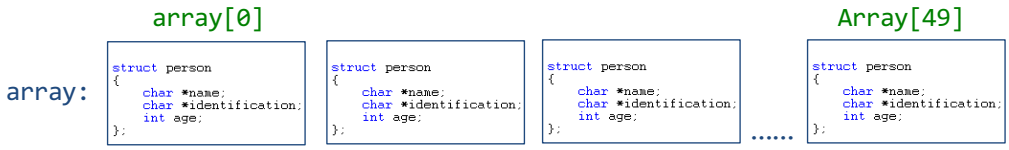
```
main()
{
	int i, size;
	char buffer[50];

	for(i=0; i<cantidad; i++)
	{
		printf("Ingrese nombre: ");
		scanf(" %[^\n]", buffer);
		size=strlen(buffer);
		personas[i].nombre=(char *) malloc (size*sizeof(char));
		strcpy(personas[i].nombre, buffer);

		printf("Ingrese rut: ");
		scanf(" %[^\n]", buffer);
		size=strlen(buffer);
		personas[i].carnetIdentidad=(char *) malloc (size*sizeof(char));
		strcpy(personas[i].carnetIdentidad, buffer);

		printf("Ingrese edad: ");

		scanf("%d", &personas[i].edad);
	}
}
```

Código en Github
https://goo.gl/jIDE9n

64

Arreglo de struct

– Cuando se manejan datos en la realidad (en las empresas, en el gobierno, en el mundo real), se Ahora, se obtiene como resultado un arreglo llamado array que contiene en cada posición un struct tipo person.

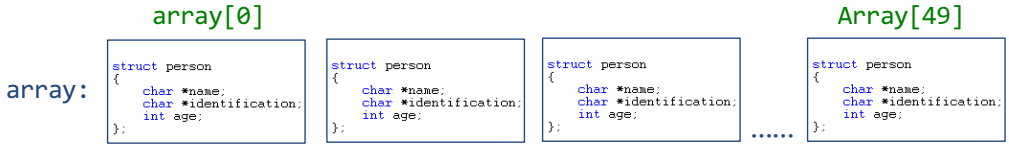


65

65

Arreglo de struct

– Entonces, si quisiéramos hacer algún cambio, como eliminación, modificación o mover, deberíamos hacerlo campo por campo… y ¿Sí estos campos (n) son muchos, se necesitarían n variables?



66

66

Ejercicio

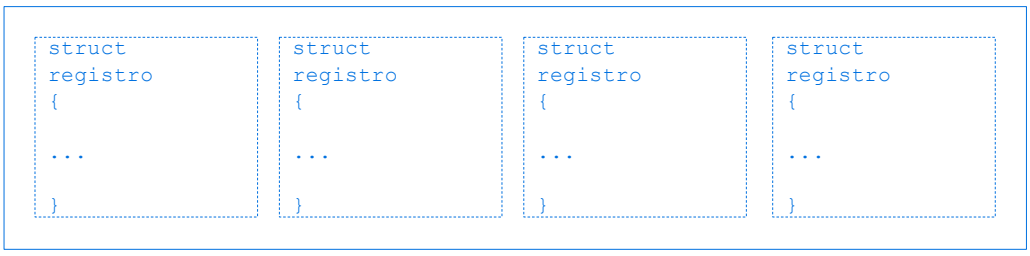
- Utilizando el struct antes definido, genere un programa que llene los datos y luego ordene el arreglo de forma ascendente por el campo edad.
- Considere 3000 personas.

```
struct Persona
{
	char *nombre;
	char *carnetIdentidad;
	int edad;
};

struct Persona personas[3000];
```

67

Arreglos de punteros a struct

- Si se tiene `struct registro arreglo[4];`

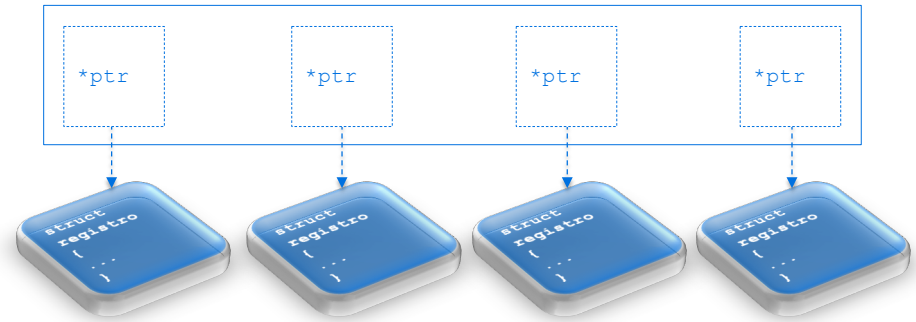


- Entonces, si quisiéramos hacer algún cambio, como eliminación, modificación o mover, deberíamos hacerlo campo por campo… y ¿Sí estos campos (n) son muchos, se necesitarían n variables?

68

Arreglos de punteros a struct

- Si se tiene `struct *registro arreglo[4];`



- Entonces, ¿Cómo quedaría el problema si deseamos hacer lo mismo que se hizo anteriormente?

69

69

Ejemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define cantidad 5

struct Persona
{
        char *nombre;
        char *carnetIdentidad;
        int edad;
};

struct Persona *personas[cantidad];

main()
{
        →
```

Código en Github
https://goo.gl/sFJd2Z

70

70

Ejemplo

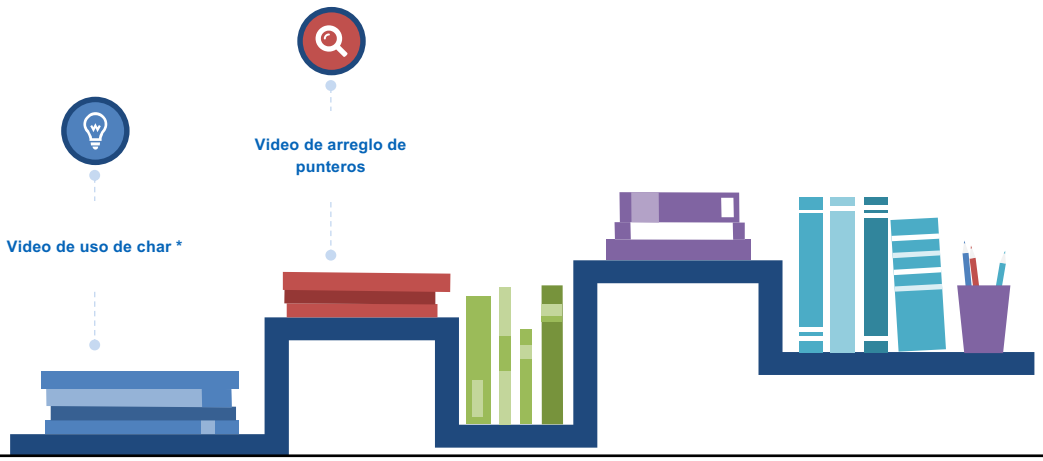
```
main()
{
        int i, size;
        char buffer[50];

        for(i=0; i<cantidad; i++)
        {
                struct Persona *nueva;
                nueva=((struct Persona*)malloc(sizeof(struct Persona)));


                printf("Ingrese nombre: ");
                scanf(" %[^\n]", buffer);
                size=strlen(buffer);
                nueva->nombre=((char *)malloc(size*sizeof(char)));
                strcpy(nueva->nombre, buffer);

                printf("Ingrese rut: ");
                scanf(" %[^\n]", buffer);
                size=strlen(buffer);
                nueva->carnetIdentidad=((char *)malloc(size*sizeof(char)));
                strcpy(nueva->carnetIdentidad, buffer);

                printf("Ingrese edad: ");
                scanf("%d", &nueva->edad);

                personas[i]=nueva;
        }
}
```

Código en Github
https://goo.gl/sFJd2Z

71

71

Lecturas del módulo



72

72