Principios y Herramientas de Programación

Dra. Jessica Andrea Carballido

jac@cs.uns.edu.ar

Dpto. de Ciencias e Ingeniería de la Computación

UNIVERSIDAD NACIONAL DEL SUR

Arreglos

Tipo de dato ESTRUCTURADO

- Un arreglo es un tipo de dato estructurado **homogéneo** cuyos elementos son accedidos con un índice.

  EL PRIMER ELEMENTO ESTA EN LA POSICION (INDICE) 0.

- Por ejemplo, la definición:

  int A[10];

  define un arreglo A de 10 componentes llamadas A[0], A[1], …, A[9], donde cada una puede contener un entero.

Indexación de arreglos

La asignación y recuperación de los valores
se hace mediante **indexación** de sus componentes.

Partiendo de la declaración de la variable A anterior, las siguientes instrucciones modifican A como se muestra:

- A[2] = 37; A[7] = 12; A[9]=7;

		37					**12**		**7**
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

- A[1] = A[7]; A[3] = A[7]+A[9];

	12	37	**19**				12		7
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

Indexación de arreglos

- A[0] = A[A[9]];
- X = 5;
- A[X+1] = X;

12	12	37	19			**5**	12		7
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

- A[4] = A[7]++; [*primero asigna y después incrementa*]
- A[8] = ++A[9]; [*primero incrementa y después asigna*]

12	12	37	19	**12**		5	**13**	**8**	**8**
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

Declaración e Inicialización de Arreglos

int v[ ] = {10, 3, 5, 2};

v es un arreglo de 4 enteros inicializado con 10, 3, 5 y 2.

int ar1[7] = {12, 13, 15};

ar1 es un arreglo de 7 elementos ar1[0]==12, ar1[1]==13, ar1[2]==15, ar1[3]==0, …, ar1[6]==0.

float a[10];

a es un arreglo de 10 números reales (sin inicializar).

char arreglo1[4], arreglo2[10];

arreglo1 es un arreglo de 4 caracteres, arreglo2 de 10 caracteres (ambos sin inicializar).

Manipulación de Vectores

Suponga que se desea escribir un programa para:

- ✓ leer un vector de 10 componentes desde el teclado, y luego,
- ✓ mostrar por pantalla el vector ingresado.

Esto podemos hacerlo definiendo una función específica para cada operación de entrada y salida, y luego invocarlas desde el *main*.

VECTORES LLENOS

```
#include <stdio.h>
#define TamVector 10

...

int main()
{
    int V[TamVector];

            LeerVector(V);
            MostrarVector(V);
            return 0;
}
```

Funciona como dato de salida (sin usar &)

LO MANDO como dato de entrada

```
void LeerVector(int vector[])
{
  int i;
  printf("Ingrese las %i componentes del vector: \n", TamVector);
  for (i=0; i<TamVector; i++)
        scanf("%i", &vector[i]);
}
```

No se indica el tamaño

VECTORES LLENOS

```
void MostrarVector(int vector[])
{
    int i;
    printf("El vector posee las siguientes componentes: \n");
    for (i=0; i<TamVector; i++)
        printf("[%i] = %i \n", i, vector[i]);
}
```

Los arreglos como parámetro
siempre funcionan como datos
de **entrada-salida**
(sin usar & en la invocación!).
NO ES COPIA, se manejan internamente
con punteros
al ser pasados como parámetro!

Suma de Vectores

```
void SumaVectores(int v1[], int v2[], int suma[])
{
    int i;
    for (i=0; i<TamVector; i++)
        suma[i]=v1[i]+v2[i];
}

int main()
{
    int Vect1[TamVector], Vect2[TamVector], Sum[TamVector];
    LeerVector(Vect1); LeerVector(Vect2);
    SumaVectores(Vect1,Vect2,Sum);
    MostrarVector(Sum);
    return 0;
}
```

VECTORES LLENOS

dato de salida

TamVector-1

```c
#include <stdio.h>
#include <stdlib.h>
#define TamVector 5

void LeerVector(int vector[])
{ int i;
   printf("Ingrese las %i componentes del vector: \n", TamVector);
   for (i=0; i<TamVector; i++)
        scanf("%i", &vector[i]);
}
void MostrarVector(int vector[])
{ int i;
    printf("El vector posee las siguientes componentes: \n");
    for (i=0; i<TamVector; i++)
        printf("[%i] = %i \n", i, vector[i]);
}
void SumaVectores(int v1[], int v2[], int suma[])
{ int i;
    for (i=0; i<TamVector; i++)
        suma[i]=v1[i]+v2[i];
}
int main()
{ int Vect1[TamVector], Vect2[TamVector], Sum[TamVector];
    LeerVector(Vect1); LeerVector(Vect2);
    SumaVectores(Vect1,Vect2,Sum);
    MostrarVector(Sum);
    return 0;
}
```

Todos tienen mismo tamaño y tipo de componentes

```c
#include <stdio.h>
#include <stdlib.h>
#define TamVector 5

typedef int tvector[TamVector];

void LeerVector(tvector vector)
{
  int i;
  printf("Ingrese las %i componentes del vector: \n", TamVector);
  for (i=0; i<TamVector; i++)
        scanf("%i", &vector[i]);
}
void MostrarVector(tvector vector)
{
    int i;
    printf("El vector posee las siguientes componentes: \n");
    for (i=0; i<TamVector; i++)
        printf("[%i] = %i \n", i, vector[i]);
}

void SumaVectores(tvector v1, tvector v2, tvector suma)
{
    int i;
    for (i=0; i<TamVector; i++)
        suma[i]=v1[i]+v2[i];
}

int main()
{
    tvector Vect1, Vect2, Sum;
    LeerVector(Vect1); LeerVector(Vect2);
    SumaVectores(Vect1,Vect2,Sum);
    MostrarVector(Sum);
    return 0;
}
```

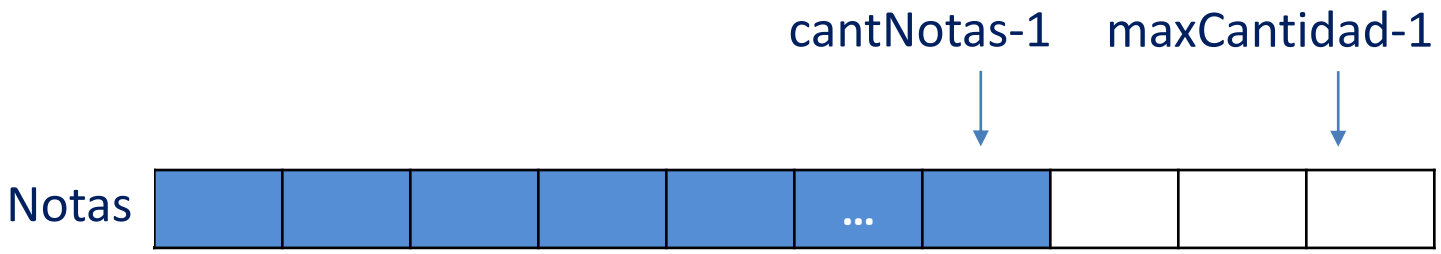
CREAMOS UN NUEVO TIPO tvector

VECTORES LLENOS

Definición de nuevos tipos

Vector de Notas

Cargar un vector con notas y luego mostrar la nota más alta.



VECTORES PARCIALMENTE CARGADOS

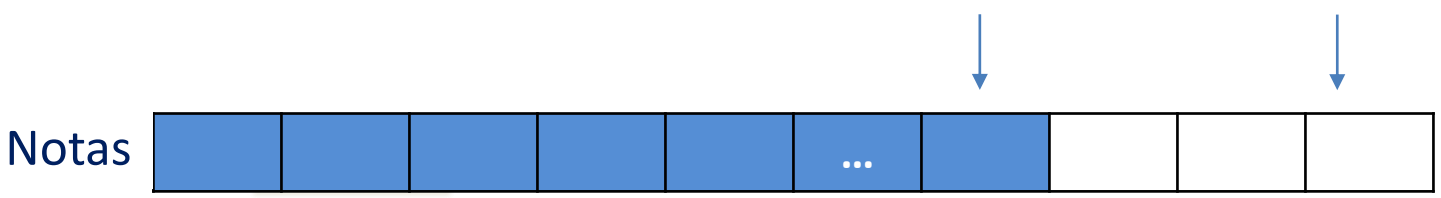
Vector de Notas

Cargar un vector con notas y luego mostrar la nota más alta.

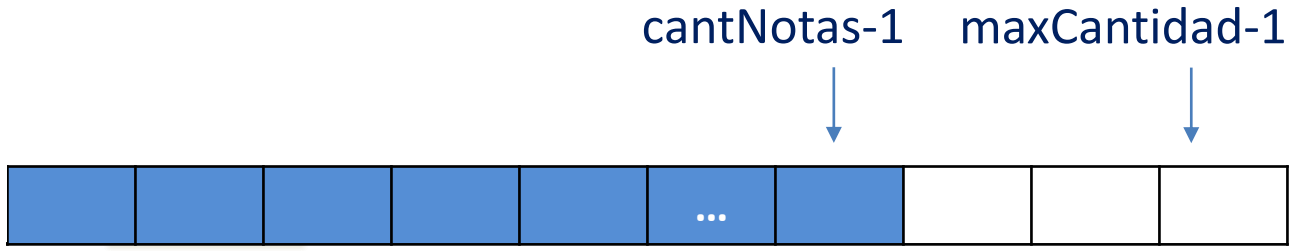
```
# include <stdio.h>
# define maxCantidad 100
typedef int tArrNotas[maxCantidad];
…        primitivas leerNotas y maximaNota
int main()
{
   int cantNotas;
   tArrNotas Notas;
            printf( "Ingrese la cantidad de notas a cargar: " );
            scanf( "%i", &cantNotas );
            leerNotas(Notas, cantNotas);
            printf("La mayor nota obtenida es: %i", maximaNota(Notas, cantNotas));
            return(0);
}
```

¿Cómo podríamos validar que la cantidad de notas no supere las 100?

VECTORES PARCIALMENTE CARGADOS



```
void leerNotas (tArrNotas Notas, int cant)
{
  int i;
  printf("Ingrese las %i notas: \n", cant);
  for (i=0; i<cant; i++)
        scanf("%i", &Notas[i]);
}
int maximaNota (tArrNotas Notas, int cant)
{
 int i, max;
 max=0;
 for (i=0; i<cant; i++)
     if (Notas[i]>max)
           max = Notas[i];
 return(max);
}
```



"a" es dato de entrada
a la primitiva "doble".

PARAMETROS: entre paréntesis
en el encabezado de las primitivas.

```c
#include <stdio.h>

int doble(int a)
{
    return(2*a);
}

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

    b=2; c=5; a=7;
    printf("Primera ejecucion: %i \n", doble(b));
    printf("Segunda ejecucion: %i \n", doble(c));
    printf("Tercera ejecucion: %i \n", doble(a));
    return 0;
}
```

Este dato "a" es un dato diferente
del que está declarado en "doble".
Tiene su propio lugar de almacenamiento.

"int" es el tipo del **ÚNICO**
dato de salida

TRAZA

Escribir un algoritmo para recibir un número positivo y calcular la suma y el producto de sus dígitos.

```
Algoritmo SumYprod
DE: N (entero)
DS: Suma, Prod (enteros)
Suma ← 0
Prod ← 1
Mientras (N>0) hacer
        Suma ← Suma + N mod 10
        Prod ← Prod * N mod 10
        N ← N div 10
```

DOS DATOS DE SALIDA!!

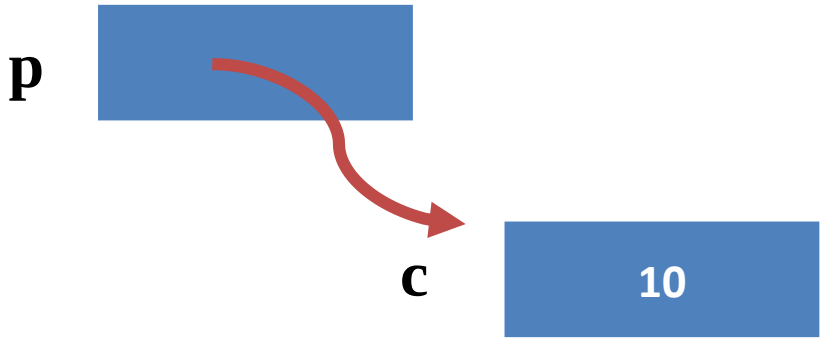
```c
#include <stdio.h>

void SumaYprod ( int N, int *suma, int *prod )
{
*suma = 0; *prod = 1;
 while(N>0)
   {
      (*suma) = (*suma) + (N % 10);
      (*prod) = (*prod) * (N % 10);
      N = N / 10;
   }
}

int main()
{
int p, s, numero;
printf("Ingrese un entero positivo: ");
scanf("%i", &numero);
SumaYprod(numero, &s, &p);
printf("la suma es %i y el producto es %i", s, p);
return 0;
}
```

Necesitamos usar PUNTEROS.

Punteros

Un puntero es una variable cuyo valor
es la **dirección** de otra variable.

- El operador unario & da la dirección de un objeto.
- El operador unario * es el operador de *indirección* o *desreferencia.*
  - p = &c (a “p” le asigno la dirección de “c”).
  - *p es 10 (lo apuntado por p es 10).



Punteros

De esta forma, aplicado a un puntero, el operador * nos da acceso al objeto apuntado por el puntero.

```
int x1 = 10, x2 = 20;
int * ipunt; // ipunt es una dirección; en este caso
                                un puntero a un entero.
    ipunt = &x1; // ipunt apunta a x1. Se le asigna una dirección
    x2 = *ipunt; // ahora x2 vale 10.
    *ipunt = 0; // x1 ahora es 0.
```

ipunt es una dirección.
*** ipunt es un entero.**

Punteros como alternativa para datos de salida de primitivas

- En C el pasaje de parámetros para variables con tipos de datos simples es siempre por “valor” (funcionan para datos de entrada).
- Sin embargo, en C podemos simular el mecanismo de pasaje de parámetros por “referencia” (datos de salida) de datos SIMPLES usando punteros.
- De hecho, lo hemos estado haciendo cuando usamos la operación de salida scanf:

```
int x;
...
scanf("%i", &x);
...
```

Ejemplo

Supongamos que deseamos definir una función, que llamaremos “swap”, para intercambiar el contenido de dos variables enteras:

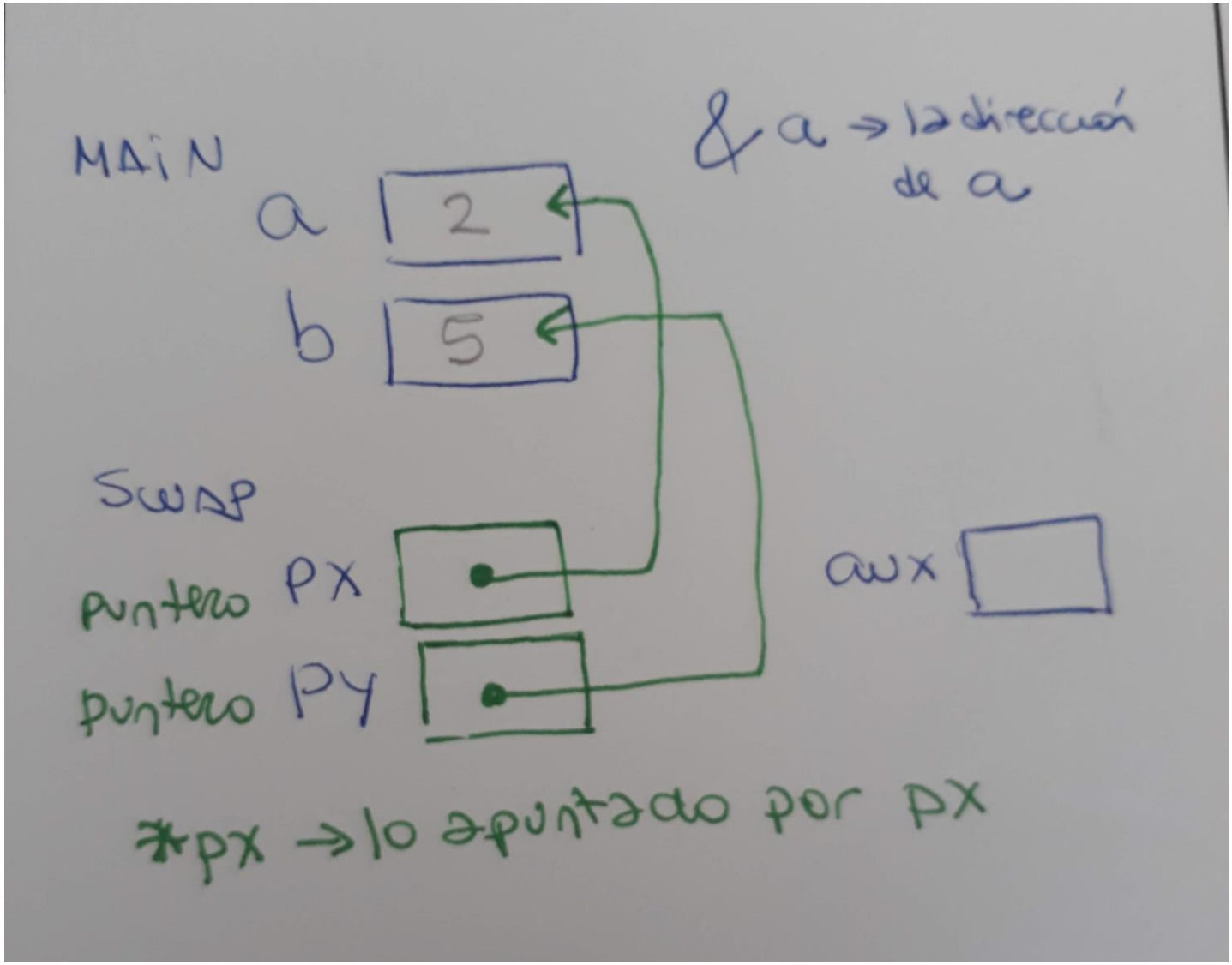
```
void swap( int *px, int *py )
{
int aux;
aux = *px;        /* asigna a aux el valor almacenado en la
                     variable a la que apunta px. */
*px = *py;        /* asigna el valor almacenado
                     en la variable a la que apunta py
                     a la variable a la que apunta px. */
*py = aux;        /* asigna el valor almacenado en aux
                     a la variable a la que apunta py. */
}
```

Puntero (referencia) a un dato de tipo entero

Cómo invocamos a la función *swap*?

```
void main( void )
{
    int a, b;
    printf("Ingrese a: \n"); scanf( "%i", &a);
    printf("Ingrese b: \n"); scanf( "%i", &b);
    swap( &a, &b );
    printf("%i %i \n", a, b);
}
```

Pasamos LA DIRECCION de a


MAIN
a
2
b
5
& a → la dirección de a
SWAP
puntero PX
puntero PY
aux
*px → lo apuntado por px

HAVE A
NICE DAY

Dra. Jessica Andrea Carballido
CONICET - DCIC (UNS)