Programación 1

UNIVERSIDAD TECNOLOGICA NACIONAL

Listas

Universidad Tecnológica Nacional - Cuch Sede Chivilcoy

Profesora Luciana Denicio

lucianadenicioapu@gmail.com


SIMPLE: aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.
TIPO DE DATO
COMPUESTO: pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.
SIMPLE
COMPUESTO
DEFINIDO POR EL LENGUAJE
DEFINIDO POR EL PROGRAMADOR
DEFINIDO POR EL LENGUAJE
DEFINIDO POR EL PROGRAMADOR
Integer
Real
Char
Boolean
Puntero
Subrango
String
Registros
Arreglos
Lista

Listas

Realizar un programa que lea números que representan edades y luego de realizar la lectura se quiere informar cuántos veces apareció la edad más grande. La lectura de edades finaliza cuando se lee la edad -1.

SOLUCION ALUMNO 1:

Leo una edad, sino es -1, la almaceno en un vector y voy contando cuantos elementos voy agregando (dimensión lógica).
Después de leer las edades recorro el vector y calculo el máximo.
Luego de calcular el máximo recorro el vector y cuento cuantas veces se leyó la edad máxima.

SOLUCION ALUMNO 2:

No sé como se resuelve pero la solución planteada por el alumno 1 está mal.

Realizar un programa que lea números que representan edades y luego de realizar la lectura se quiere informar cuántos veces apareció la edad más grande. La lectura de edades finaliza cuando se lee la edad -1.

Disponer de alguna **ESTRUCTURA** donde almacenar los números pero que no se necesite especificar un tamaño en el momento de la declaración.

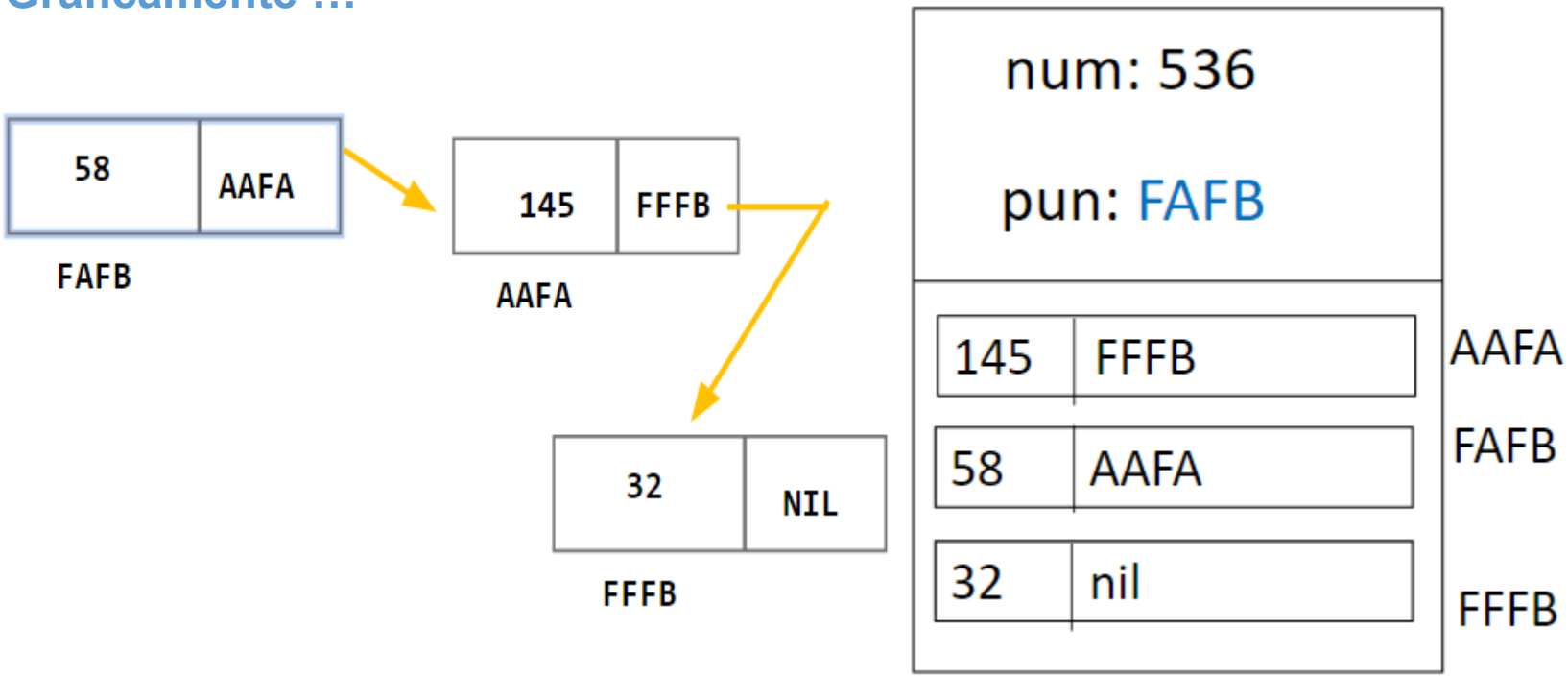
Listas

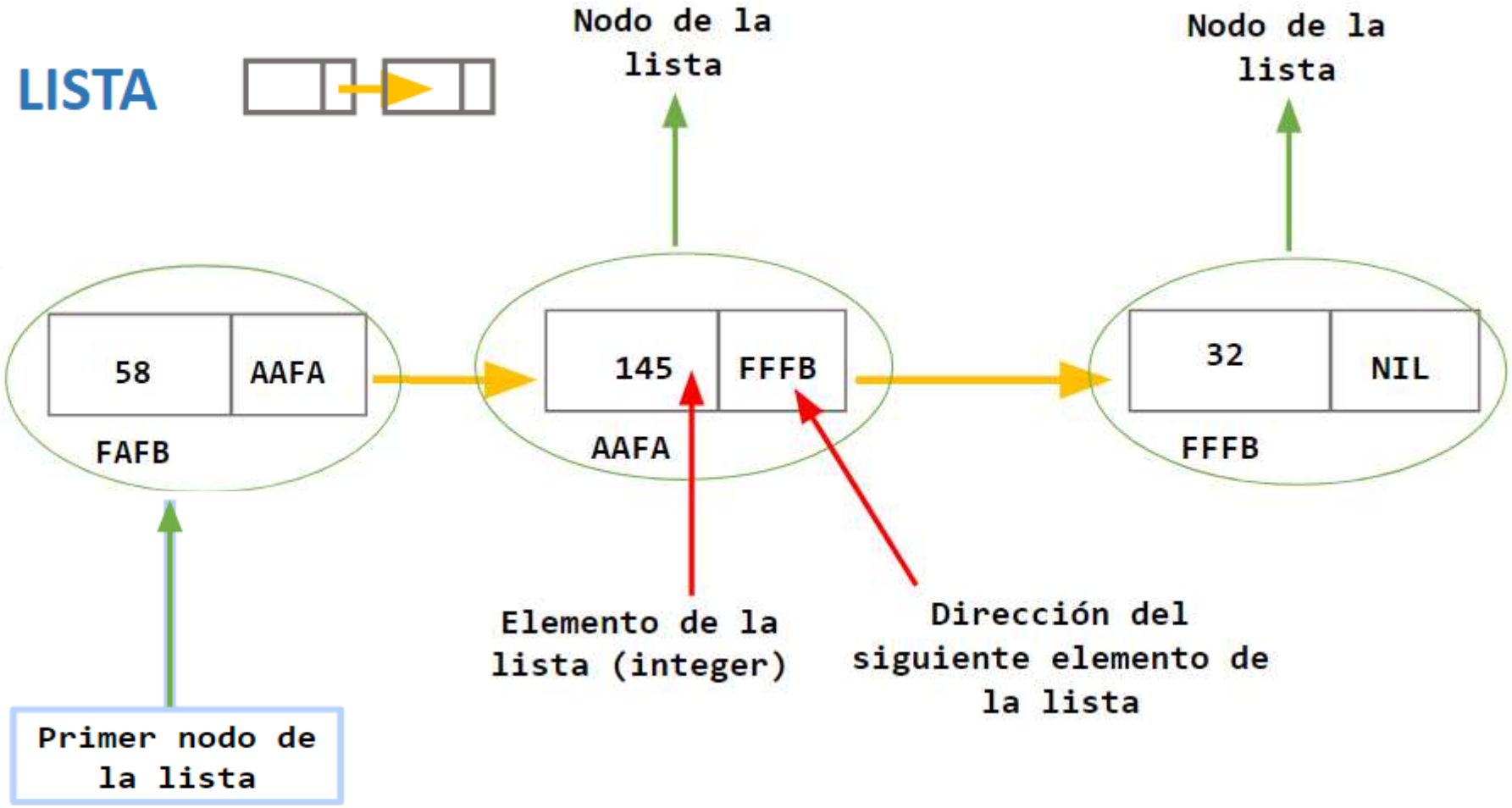
Colección de nodos, donde cada nodo contiene un elemento y en que dirección de memoria se encuentra el siguiente nodo.
Cada nodo de la lista se representa con un puntero, que apunta a un dato (elemento de la lista) y a una dirección (donde se ubica el siguiente elemento de la lista).
Toda lista tiene un nodo inicial.

Los **nodos** que la componen pueden no ocupar posiciones contiguas de memoria. Es decir pueden aparecer dispersos en la memoria, pero mantienen un orden lógico interno.

Listas

Gráficamente …




LISTA
Nodo de la lista
Nodo de la lista
58
AAFA
FAFB
145
FFFB
AAFA
32
NIL
FFFB
Primer nodo de la lista
Elemento de la lista (integer)
Dirección del siguiente elemento de la lista

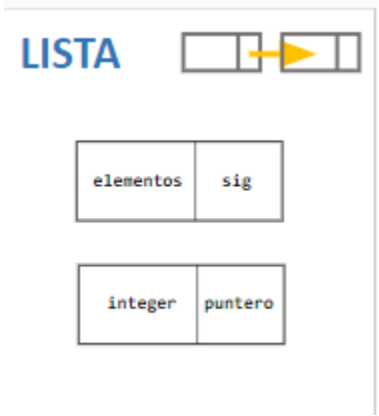
Listas

```
#include <iostream>

struct nombreElemento {
    tipoElemento elementos;   // Cualquiera de los tipos vistos



    nombreElemento * punteroSig; // Estructura recursiva
};

nombreElemento* Pri = nullptr; // Declaración de un puntero a nombreElemento

int main() {
    // Código
    return 0;
}
```

Listas

- **Lista de enteros**

```
#include <iostream>

  struct datosEnteros {
  int elementos;
   datosEnteros* sig;
};

datosEnteros* PriEnteros = nullptr;

int main() {
// Código C
return 0;
}
```



Listas

- Lista de Registro por ejemplo Alumnos

```
#include <iostream>
 #include <string>

struct alumno {
    string nombre;
     string numeroA;
};

struct datosAlumnos {
    alumno elementos;
    datosAlumnos* sig;
};

datosAlumnos* PriAlumnos = nullptr;

int main() {
    // Código C+
    return 0;
}
```

Listas

OPERACIONES

Creación de una lista.

Agregar nodos al comienzo de la lista.

Recorrido de una lista.

Agregar nodos al final de la lista.

Insertar nodos en una lista ordenada

Eliminar nodos de una lista

Listas

- Crear una lista

Implica marcar que la lista no tiene una

```
struct datosEnteros {
    int elem;
    datosEnteros* sig;
};

void crear(datosEnteros*& p) {
    p = nullptr;
}

int main() {

 datosEnteros* pri;
    crear(pri);
 // Código C
    return 0;
}
```

Listas

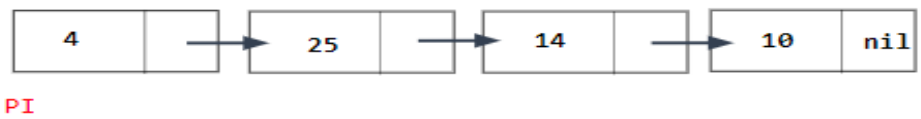
- **RECORRER UNA LISTA**

Implica posicionarse al comienzo de la lista y a partir de allí ir “pasando” por cada elemento de la misma hasta llegar al final.

- inicializo una variable **auxiliar** con la dirección del puntero inicial
- mientras (no sea el final de la lista)
- proceso el elemento (por ejemplo imprimo)

  avanzo al siguiente elemento de **auxiliar**

Listas




```
4
25
14
```

```
#include <stdio.h>
#include <stdlib.h>

struct datosEnteros {
    int elem;
    struct datosEnteros* sig;
};
void recorrerLista(struct datosEnteros* pI) {
    struct datosEnteros* aux = pI;

    while (aux != NULL) {
        printf("%d ", aux->elem);
        aux = aux->sig;
    }
}
```

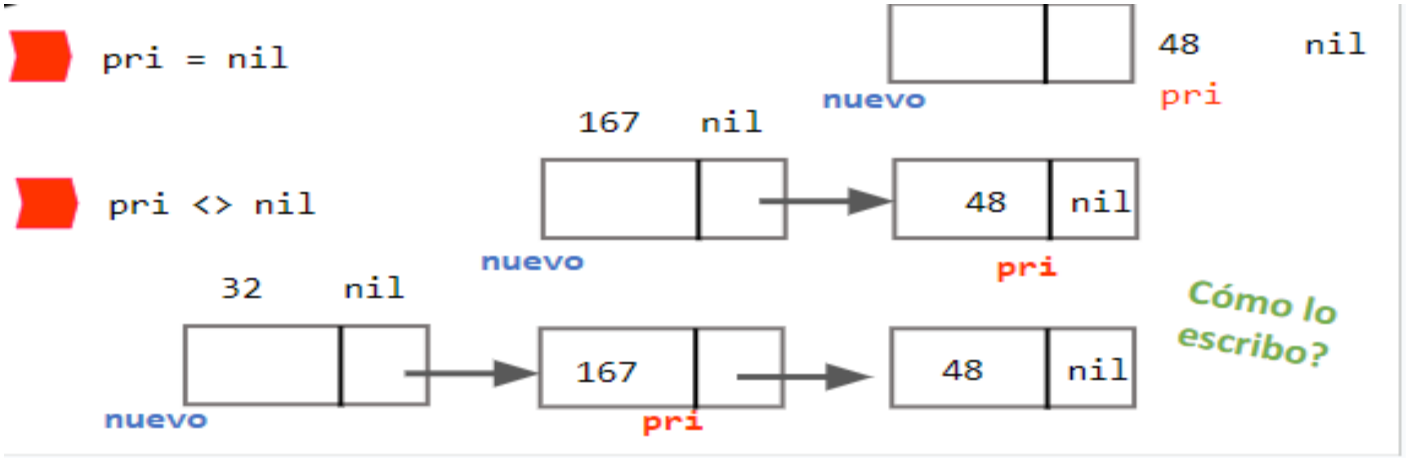
Listas

```
int main() {
datosEnteros* pri = nullptr; // Declaración de un puntero a datosEnteros

// Llama a la función crear para inicializar pri

recorrerLista(pri); // Llama a la función para recorrer e imprimir la lista
return 0; }
```

Listas

- **AGREGAR ADELANTE EN UNA LISTA**

Implica generar un nuevo nodo y agregarlo como primer elemento de la lista.



Listas

Reservo espacio en memoria **nuevo elemento**.

si (es el primer elemento a agregar)
  asigno al puntero inicial la dirección del **nuevo elemento**.

sino
  indico que el siguiente de **nuevo elemento** es el puntero inicial.
  actualizo el puntero inicial de la lista con la dirección del **nuevo elemento**

Listas

```
#include <stdio.h>
#include <stdlib.h>

struct datosEnteros {
    int elem;
    struct datosEnteros* sig;
};

void crear(struct datosEnteros**
p) {
    *p = NULL;
}
```

```
void agregarAdelante(struct datosEnteros** p, int
num) {
    struct datosEnteros* nuevoNodo = (struct
datosEnteros*)malloc(sizeof(struct datosEnteros));
    if (nuevoNodo == NULL) {
        printf("Error al asignar memoria.\n");
        return 0;
    }
    nuevoNodo->elem = num;
    nuevoNodo->sig = *p;
    *p = nuevoNodo;
}
```

Listas

```
int main() {
    struct datosEnteros* pri;
    int num;
    crear(&pri);
    printf("Ingrese un número: ");
    scanf("%d", &num);
    agregarAdelante(&pri, num);

    // Mostrar el valor ingresado
    printf("Valor en la lista: %d\n", pri->elem);

    // Liberar memoria
    free(pri);

    return 0;
}
```