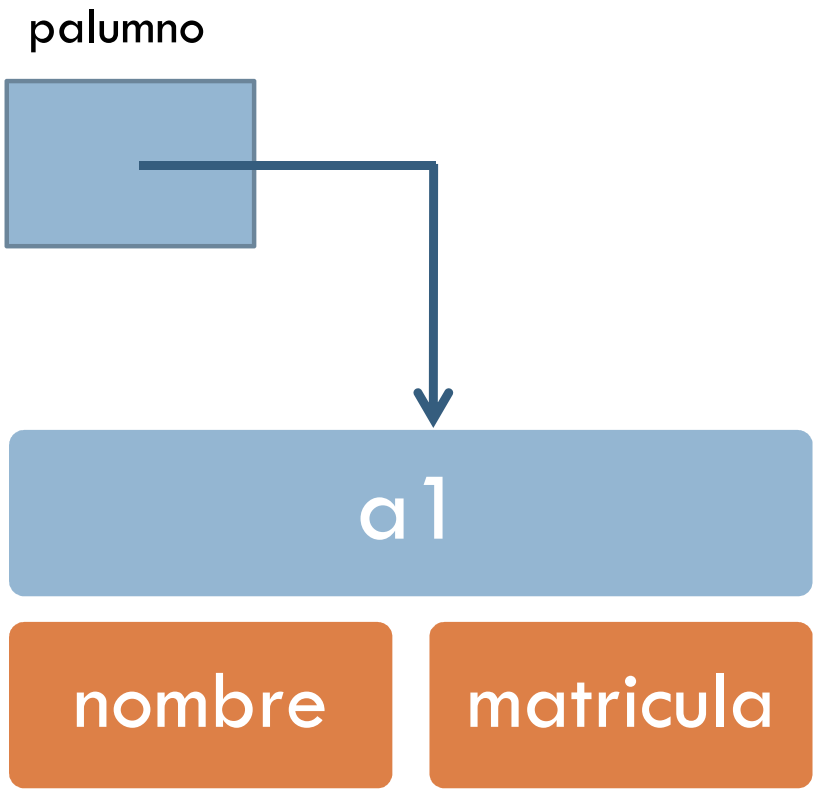
Apuntadores a Estructuras

- Un apuntador puede apuntar hacia cualquier tipo de datos incluyendo estructuras.
- Por Ejemplo

```
typedef struct{
    char nombre[30];
    int  matricula;
    } alumno;

alumno *palumno, a1;

palumno = &a1;
```



Apuntadores a Estructuras

- Cuando se usan apuntadores a estructuras, no se puede utilizar el operador punto para acceder un campo.
  - `palumno.nombre` es una forma incorrecta de acceder al nombre del alumno

- Los apuntadores requieren del operador flecha para acceder a los campos de la estructura
  - `palumno->nombre`

Apuntadores a Estructuras

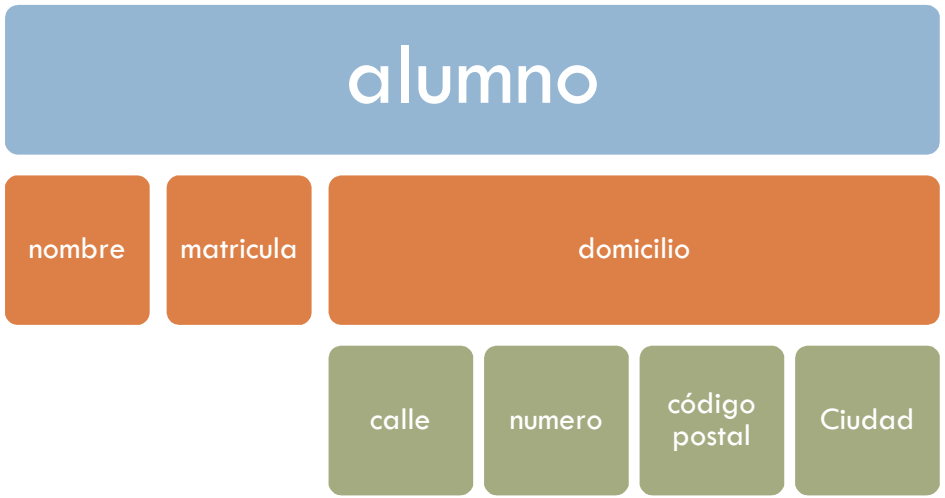
- Considere la siguiente estructura y diga como puedo acceder a cada campo

```
typedef struct{
    char calle[30];
    int  numero;
    int  CP;
    char ciudad[30];
    } dir;

typedef struct{
    char nombre[30];
    int  matricula;
    dir  domicilio;
    } alumno;

alumno *palumno, a1;

palumno = &a1;
```

```
palumno->nombre;
palumno->matricula;
palumno->domicilio.calle;
palumno->domicilio.numero;
palumno->domicilio.CP;
palumno->domicilio.ciudad;


palumno->nombre[0];
palumno->domicilio.ciudad[0];
```

Apuntadores a Estructuras

- Considere la siguiente estructura y diga como puedo acceder a cada campo




```
palumno->nombre;
palumno->matricula;
palumno->domicilio.calle;
palumno->domicilio.numero;
palumno->domicilio.CP;
palumno->domicilio.ciudad;
```

```
palumno->nombre[0];
palumno->domicilio.ciudad[0];
```

Reserva de Memoria Dinámica

- Los apuntadores deben apuntar hacia algún lugar de memoria que contenga una estructura, por ejemplo
  - `palumno = &a1;`
- Cuando requerimos de un conjunto de estructuras entonces debemos reservar la memoria por medio de las funciones malloc, calloc o realloc.
- El operador sizeof nos proporcionara el tamaño adecuado de nuestra estructura.

Reserva de Memoria Dinámica

- Ejemplo

```
typedef struct{
    char nombre[30];
    int  matricula;
    dir  domicilio;
    } alumno;

alumno *palumno;

printf("Proporcione el numero de estudiantes: ");
scanf("%d",&n);
palumno = (alumno *)malloc(n*sizeof(alumno));

palumno[i].matricula;
(palumno+i)->matricula;
```

Ejemplo 1

- Considere la siguiente estructura:
  - Alumno
    - Matricula
    - Nombre
    - Parciales (lista de 5 calificaciones parciales)

- Lea una lista de n alumnos con estas características y despliegue el promedio por alumno.

Solución

```
typedef struct{
    char nombre[30];
    int  matricula;
    float  parcial[5];
    } alumno;

main()
{
   alumno *palumno;
   printf(“Proporcione el numero de estudiantes: ”);
   scanf(“%d”,&n);
   palumno = (alumno *)malloc(n*sizeof(alumno));
   lea(palumno, n);
   imprime(palumno,n);
   system(pause);
}
```

Función lea

```
void lea(alumno *lista, int n)
{
    int i,j;
    for(i = 0; i <n; i++)
      {
        printf(“alumno %d\n”,i)
        printf(“Nombre: ”);gets(lista[i].nombre);
        printf(“Matricula:”);
        scanf(“%d”,&lista[i].matricula);
        for(j = 0; j<5; j++){
           printf(“Parcial %d: ”,j);
           scanf(“%d”,&lista[i].parcial[j]);
           }
       }
}
```

Función promedio

```
void promedio(alumno *lista, int n)
{
  int i,j,suma;
  for(i = 0; i <n; i++)
  {
    printf(“Nombre: %s\t”,lista[i].nombre);
    printf(“Matricula:%d\t”,lista[i].matricula);
    suma = 0;
    for(j = 0; j<5; j++)
        suma += lista[i].parcial[j];
    printf(“Promedio %d \n”,suma/5);
   }
}
```

Función promedio

```
void promedio(alumno *lista, int n)
{
  int i,j,suma;
  for(i = 0; i <n; i++,lista++)
  {
   printf(“Nombre: %s\t”,lista->nombre);
   printf(“Matricula:%d\t”,lista->matricula);
   suma = 0;
   for(j = 0; j<5; j++)
       suma += lista->parcial[j];
   printf(“Promedio %d \n”,suma/5);
   }
}
```

Ejercicio 1

- Escriba un programa que lea un grupo de n alumnos. La información de cada alumno debe contener los siguientes datos:
  - Nombre
  - Matricula
  - Promedio

- Utilice una función para leer los n alumnos. Utilice apuntadores para crear dinámicamente la lista de alumnos.

Uniones

- Las uniones son similares a las estructuras.
- La diferencia fundamental entre estructuras y uniones es que los miembros de las uniones comparten el espacio de almacenamiento.
- No sirven para ahorrar memoria, sin embargo se debe considerar que solo un miembro de la unión a la vez puede tener un valor asignado
- El espacio de memoria reservado para la unión es el del campo mas amplio.

Declaración de Uniones

- Ejemplo:

```
union contacto
{
  char celular[15];
  char correo[20];
}
```


```
union contacto a,b;
strcpy(a.celular,"444158329");
printf("%s\n",a.celular);
printf("%s\n",a.correo);
```


```
strcpy(b.correo,"rma1294@gmail.
com");
printf("%s\n",b.celular);
printf("%s\n",b.correo);
```

typedef

- Podemos utilizar typedef para crear un nombre mas corto para las uniones

- Ejemplo

```
typedef union
{
   char celular[15];
   char correo[30];
} datos;
```

- Las uniones pueden ser usadas dentro de estructuras.

- Ejemplo:

```
typedef struct{
    char  nombre[30];
    int   matricula;
    datos contacto;
    } alumno;
```

Enumeraciones

- Las enumeraciones es una alternativa para definir diferentes valores constantes
- La sintaxis de la enumeración es la siguiente
  - `enum nombre{valores};`
- Por ejemplo

```
enum boolean{NO,YES};
NO = 0;
YES = 1;
enum key{BELL = '\a',
BACKSPACE = '\b',
TAB='\t',ENTER='\n'};
enum mes{ene=1, feb,
mar, abr, may, jun,
jul,ago,sep,oct,nov,d
ic};
```

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

typedef struct{
        char nombre[30];
        int matricula;
        } alumno;

int main()
{
    enum boolean{NO,YES};
    alumno *p;
    int n=1,i=0;
    char resp[5];
    boolean decision;

    p = (alumno *)malloc(n*sizeof(alumno));
    printf("Proporcione los siguientes datos\n");
    do{
        printf("Nombre: ");gets(p[n-1].nombre);
        printf("Matricula: ");scanf("%d",&p[n-1].matricula);
        fflush(stdin);
```

```
            if(!strcmp(strupr(resp),"SI"))
              {
                decision = YES;
                n++;
                p = (alumno *)realloc(p,n*sizeof(alumno));
               }
            else
                decision = NO;
     }while(decision != NO);

    free(p);
    system("pause");
    return 0;

}
```