Programación I

http://proguno.unsl.edu.ar

proguno@unsl.edu.ar

Asignación Dinámica de la Memoria

- Hasta ahora, hemos visto que en C existen las *permanencias o tiempos de vida*:
  - Estática
  - Automática

- Los punteros tienen una *permanencia Asignada o dinámica*: se asigna o desasigna memoria en tiempo de ejecución

¿Cómo pido Memoria?

```
void *malloc(size_t tamaño);
```

- Reserva la cantidad de memoria especificada por parámetro.
- Retorna la dirección de la porción de memoria que reservó. Es decir que **retorna un puntero**.
- En el caso de que no hubiera suficiente memoria para asignar retorna NULL.
- Debo agregar una librería

```
#include<malloc.h>
```

Ejemplo 1

```
int *puntero = malloc(4);

printf("Direccion del Puntero: %i\n",
puntero);

printf("Contenido de la memoria: %i",
*puntero);
```

Resultado

Direccion del Puntero: 0x109c010
Contenido de la memoria: 0

Ejemplo 2

```
int *puntero = malloc(4);
 if (puntero == NULL) {
        printf( "La aplicación no
    pudo reservar memoria y se va a
    cerrar!\n");
    exit(1)
    }
```

Asignación Dinámica de la Memoria

– ¿Cuál es la cantidad de bytes que necesito?
– Para determinar ese valor se utiliza la función

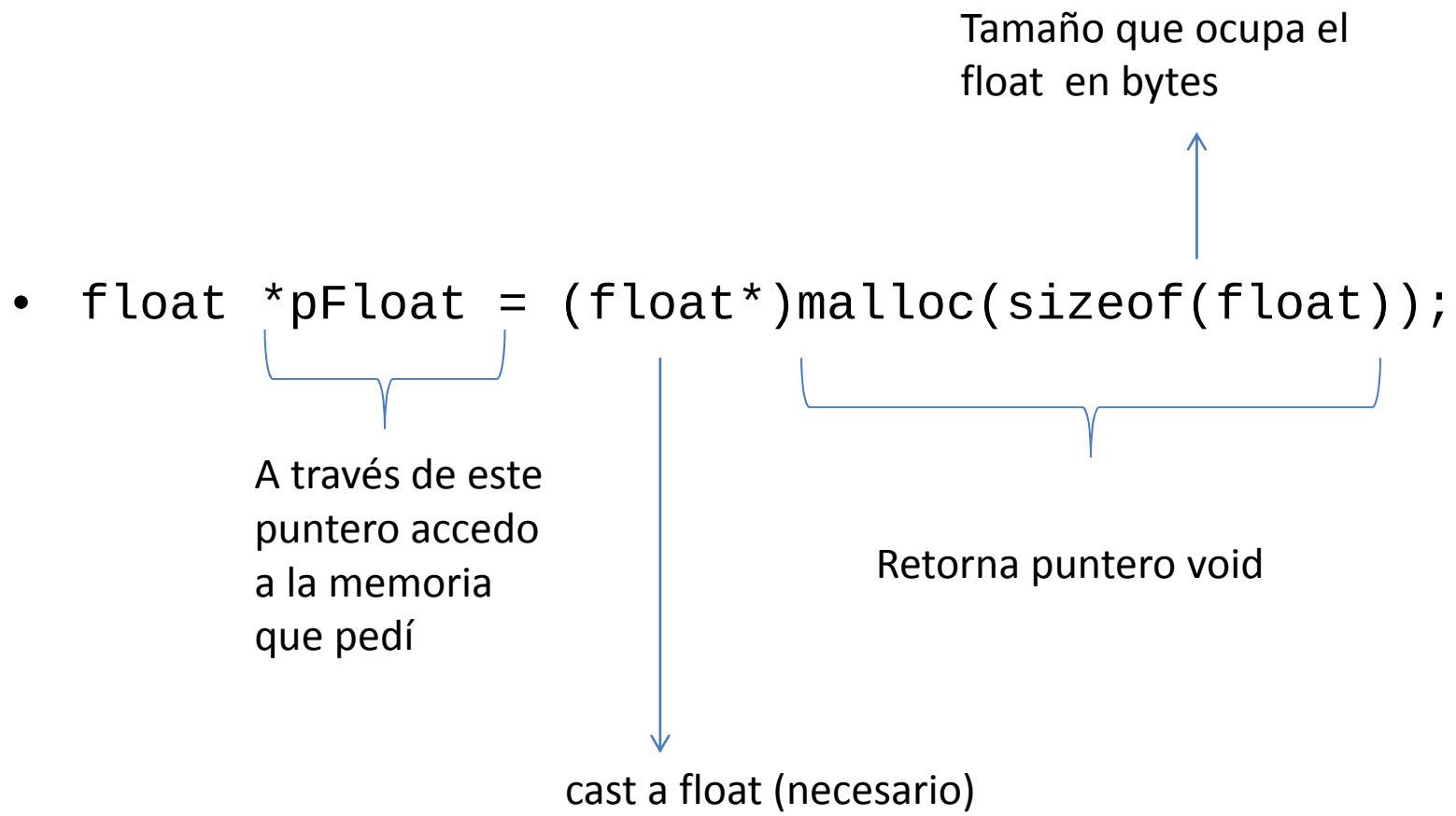
Int SIZEOF (tipo)

– Recibe como operando un tipo de dato y retorna el tamaño en bytes que ocupa en memoria un objeto del tipo de dato al cual se le aplicó el operador.

sizeof(int) ---> 4

sizeof(char)--> 1

Pedido de memoria de un float



Pedido de memoria de ***n*** float

```
float *pnfloat = (float*)malloc(sizeof(float)*n);
```

- Acceso a los elementos

  Los lugares de memoria asignados se encuentran enlazados y consecutivos.

  pnfloat[i]=0.0 //lo manejo como arreglo!2

Acceso a los valores

```
*pnfloat= 0.0;     //coloco 0.0 en el 1° lugar
pnfloat=pnfloat + sizeof(float)
                   //muevo el  puntero al siguiente
*pnfloat= 1.0;
```

- Si bien hay similitud entre el arreglo y el acceso a memoria asignada consecutivamente puede haber inconvenientes por el tamaño

Ejemplo completo

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

 main(){
float  f=1.0;

float *pnfloat = (float*)malloc(sizeof(float)*5);
for(int i=0;i<5;i++)
{
   *pnfloat= f;   //coloco valor de f en el 1° lugar
     printf("Valor:  %f\n", *pnfloat);
     f+=(float)i*2;
     pnfloat=pnfloat+sizeof(float); //muevo el puntero al elemento siguiente

}
getchar();
```

Memoria dinámica para estructuras

```
struct alumno{
//nombre, registro,carrera
        }
struct alumno *alu
alu=(struct alumno *)
        malloc(sizeof(struct alumno))
```

Asignación Dinámica de la Memoria

- `void free (void *puntero)`
- `free`
  - Desasigna la memoria apuntada por el puntero al que se le aplica, la cual es retornada al sistema y en consecuencia la misma podrá ser reasignada en el futuro.

```
free((void*)pFloat); /*libera memoria */
```

Memoria dinámica y strings

```
0    char *a;
1    a= "hola" //correcto?
2    b[]="estas?";  //correcto?
3    a=(char *)malloc(sizeof(b));//correcto?
4    a=(char *)malloc(strlen(b)+1);
                          //correcto?
5    a= "chau" //correcto?
6    (*a)= "chau"   //correcto?
7    strcpy(*a,b); //correcto?
8    strcpy(a,b); //correcto?
```

Pedido de Memoria de n registros

```
typedef struct{
    char nombre[40];
    long int registro;
    int carrera;// codifico carrera
        }alumno; //def tipo alumno
alumno *alu;
alu=(alumno *)
    malloc((sizeof(alumno))*n);
//simulo un arreglo de registro en memoria dinámica
```

Uso de **n** registros en memoria

Cargar valores:

```
alu[0].registro=123499;
strcpy(alu[0].nombre,"Juan Lopez");
(*alu).carrera= 1;//carrera codificada
```

Mostrar valores:

```
for(i=0; i<n; i++)
printf("El registro es:%d\n",alu[i].registro);
```

Es equivalente a:

```
for(i=0; i<n; i++){
printf("El registro es:%d\n",(*alu).registro);
alu=alu+sizeof(alumno);
}
```

Pasaje de parámetros

- ¿Cómo se ve afectado el pasaje de parámetros en las funciones con asignación de memoria?

```
char *a
alumno *alu
```

- Independientemente de que pida o no memoria se utilizan como punteros. Es decir paso el contenido (lo que apunta) o la dirección dependiendo de la operación a realizar.