Laboratorio de Computación II

Punteros
Asignación dinámica de memoria

Punteros

Un puntero es un tipo de variable que puede almacenar la dirección de memoria de otra variable. Para hacerlo hay que garantizar las siguientes reglas:

- **Un puntero debe declararse de un tipo de dato en concreto**
- **Un puntero debe apuntar a una variable de su mismo tipo (salvo punteros void)**

Operadores de dirección e indirección

Para trabajar con punteros es necesario hacer uso de dos operadores especiales.

Operador de dirección: &
Permite obtener la dirección de memoria de una variable.

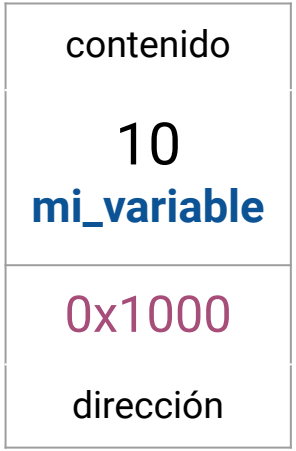
Operador de indirección: *
Permite acceder al contenido de una dirección de memoria.

Operador de dirección

Permite obtener la dirección de memoria a partir de una variable

Ejemplo

```
int mi_variable = 10;
cout << mi_variable;
cout << &mi_variable;
```



Operador de indirección

Permite acceder al contenido a partir de una dirección de memoria

Ejemplo

```
int mi_variable = 10;
int *mi_puntero;
mi_puntero = &mi_variable;
cout << *mi_puntero;
```

contenido	contenido
10 **mi_variable**	0x1000 **mi_puntero**
0x1000	0x2000
dirección	dirección

Ventajas

- Enviando la dirección de una variable a una función, permite que la misma modifique su contenido de forma permanente.
- Necesarios para hacer uso de Asignación dinámica de memoria.
- Al usarlos bien hacen felices a los profesores de Laboratorio II.

Punteros y vectores

Un puntero puede apuntar a la dirección de inicio de un vector de su mismo tipo y ser de utilidad para acceder a todas sus posiciones.

```
int mi_vector[10]={};
int *mi_puntero;
mi_puntero = mi_vector;
mi_puntero[4] = 100;
```

Recordar que el nombre de un vector es un **puntero constante a la dirección donde inicia el vector**.

¿Y el operador de indirección? El uso de corchetes indirecciona un puntero en la posición indicada. Es equivalente a: ***(mi_puntero+4) = 100**

Aritmética de punteros

La notación *(mi_puntero+4) corresponde a la aritmética de punteros. Es decir, aplicar operaciones matemáticas a punteros.

```
vec   → &vec[0] → 0x100
vec+0 → &vec[0] → 0x100
vec+1 → &vec[1] → 0x104
```

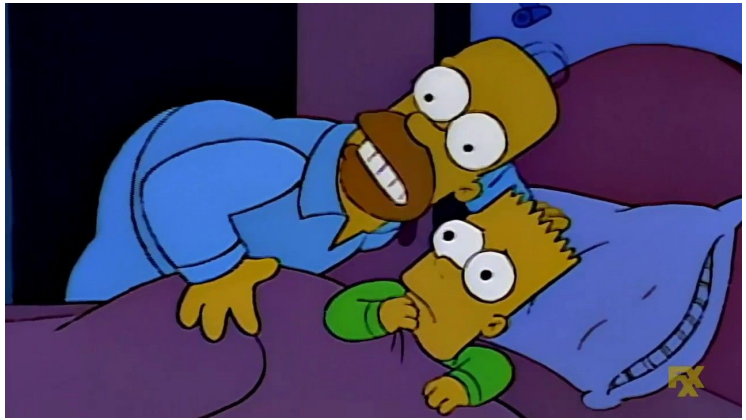
vec+N → &vec[N]

```
*(vec)   → *(&vec[0]) → 100
*(vec+0) → *(&vec[0]) → 100
*(vec+1) → *(&vec[1]) → 200
```

***(vec+N) → *(&vec[N]) → vec[N]**

Elemento	Contenido	Dirección
vec[0]	100	0x100
vec[1]	200	0x104
vec[2]	300	0x10C
vec[3]	400	0x110
vec[4]	500	0x114

Aritmética de punteros



No quiero asustarlos pero **también aplica a matrices** de N dimensiones.

```
*(*(vec+0)+0) → vec[0][0]

*(*(vec+3)+5) → vec[3][5]

*(*(vec+F)+C) → vec[F][C]
```

Actividad

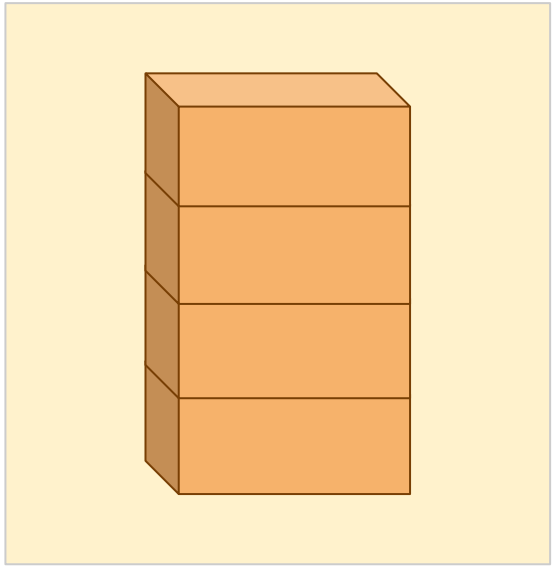
Hacer una función que permita cargar un vector de 10 elementos y otra que permita obtener el máximo valor de un vector de 10 elementos. Usar aritmética de punteros.

Asignación dinámica de memoria

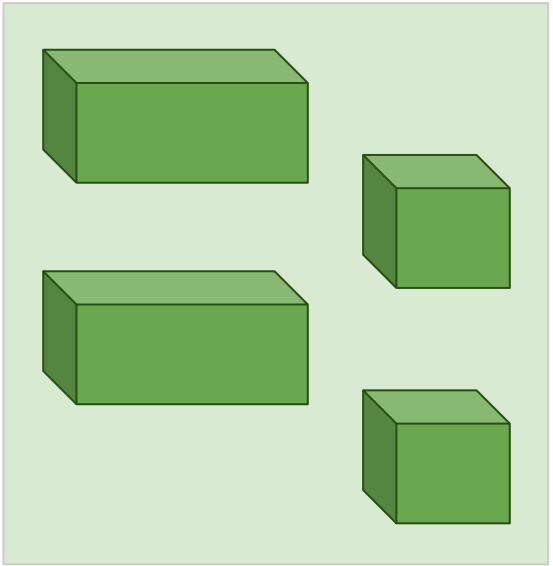
- Proceso que permite solicitar memoria adicional al sistema operativo en **tiempo de ejecución**.
- Nos permite utilizar la memoria exacta que necesitamos para trabajar y, una vez utilizada, debemos liberarla.
- Nos permite utilizar una mayor cantidad de memoria que de la manera convencional.

Memoria

La memoria se puede clasificar en stack o heap según su ubicación.



Memoria stack

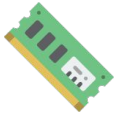


Memoria heap

Memoria

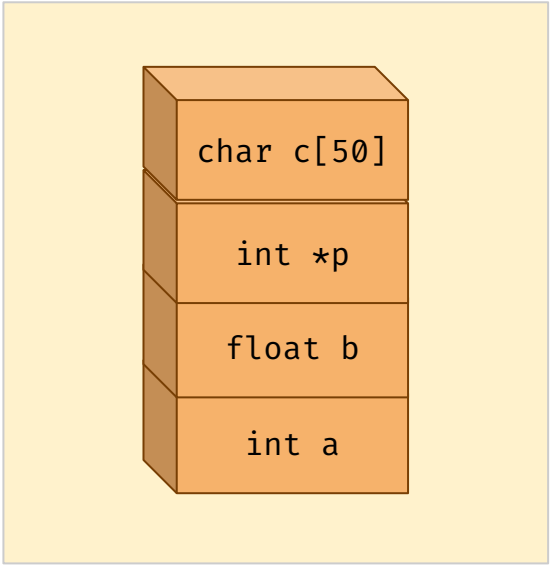
- Una variable puede figurar en la memoria stack o en la memoria heap dependiendo de cómo la declaremos.
- Hasta el momento siempre utilizamos la memoria stack. La memoria dinámica permitirá ubicar nuestras variables en la memoria heap.
- La memoria heap es una memoria compartida por varios programas ejecutándose en el sistema operativo. No hay garantía de poder obtener la necesaria para nuestro programa.

Memoria stack



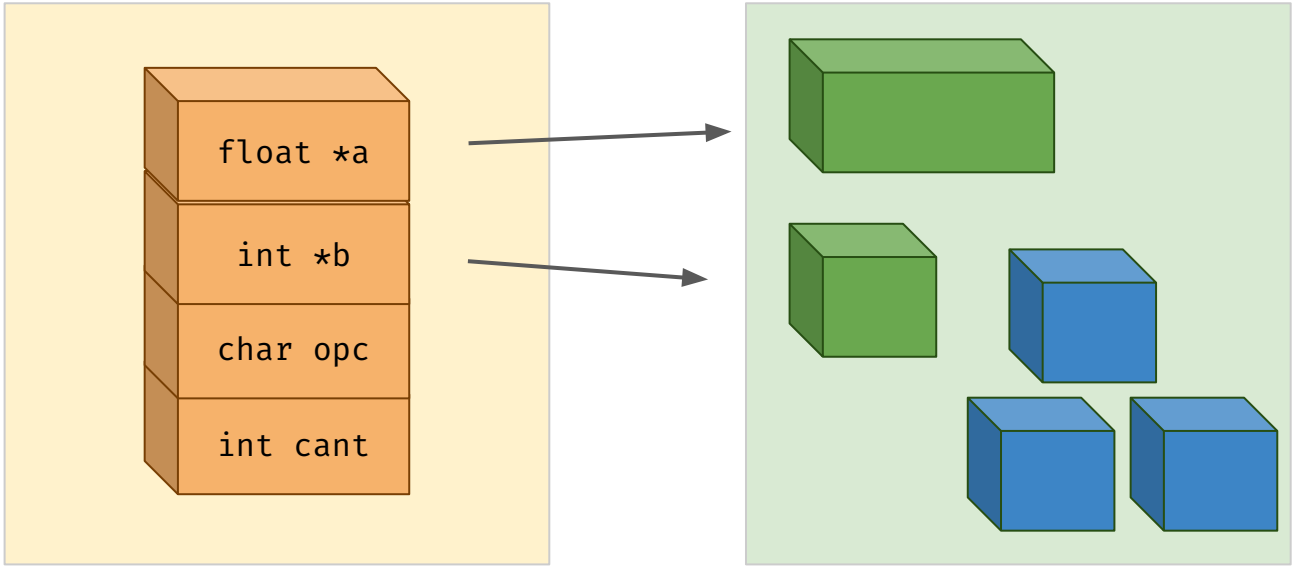
- Cada variable que declaremos en una función (incluso main) se ubica en la memoria stack.
- La memoria stack es limitada. De superar su límite genera una excepción (desbordamiento de pila o stack overflow).

```
int main(){
  int a, float b;
  int *p;
  char c[50];
}
```



Memoria heap

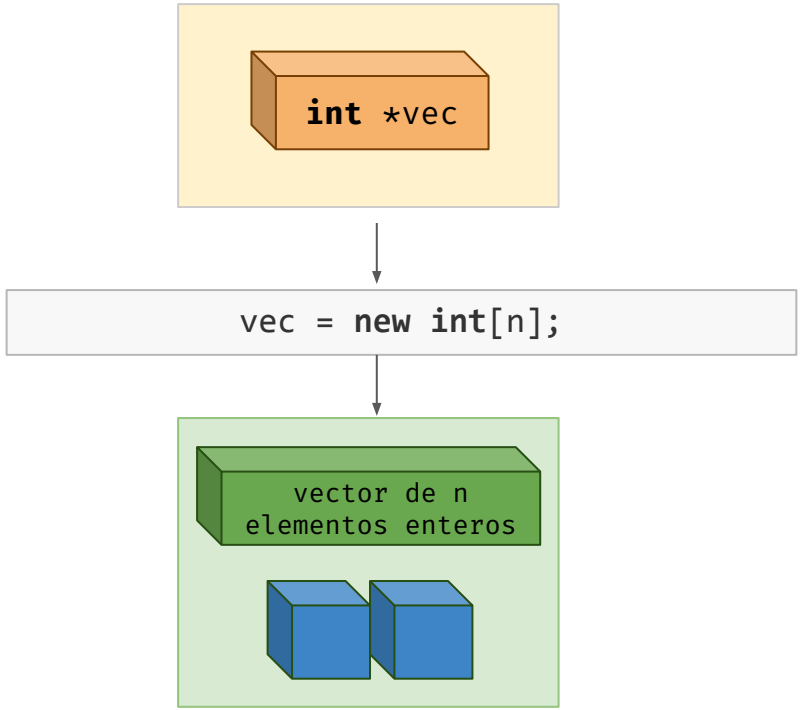
Se crea un puntero en la memoria stack que, luego de pedir memoria, apunta al comienzo del espacio de memoria solicitado.



Operador new

Operador utilizado para pedir memoria dinámica sobre un puntero previamente declarado.

```
#include <iostream>
using namespace std;
int main(){
  /* Memoria dinámica para
  un vector de N elementos */
  int *vec, n;
  cin >> n;
  vec = new int[n];
  if (vec == NULL)
    return 1; // No hay memoria
  // Resto del programa
  delete vec; // Liberación de la memoria
  return 0;
}
```



Operador delete

Operador utilizado para liberar memoria dinámica sobre un puntero previamente utilizado.

"Parece un operador tonto pero así como lo ves, si no lo utilizás, se clava cinco programas en un día"