

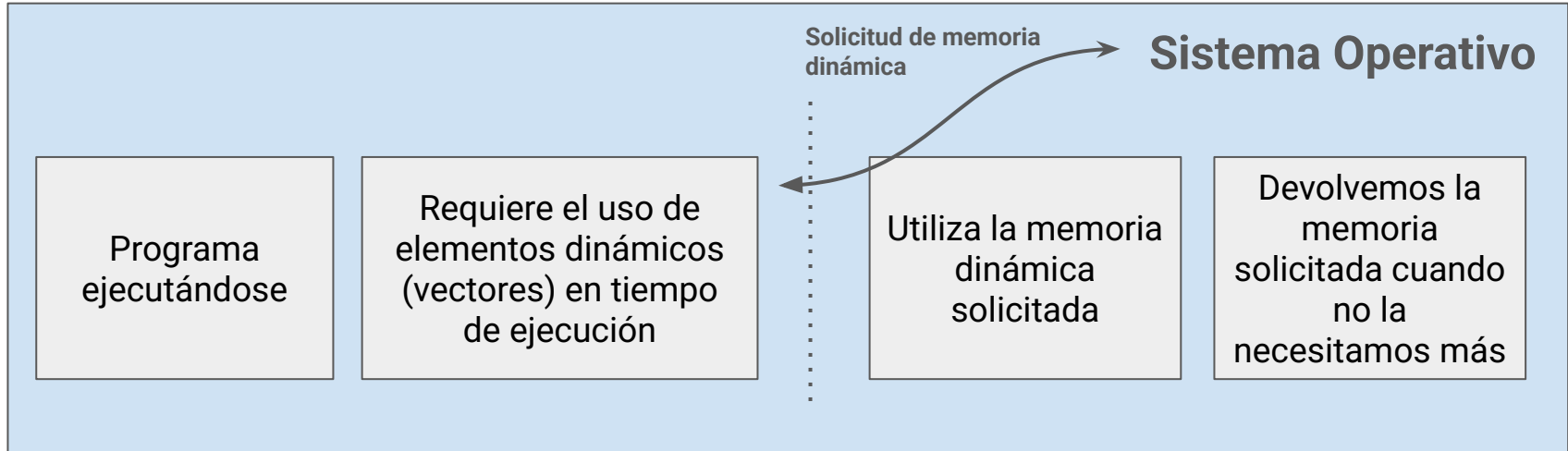
# Programación II

**Punteros**


**Asignación dinámica de memoria**

# Asignación dinámica de memoria

Proceso que permite solicitar memoria adicional al sistema operativo en **tiempo de ejecución**.

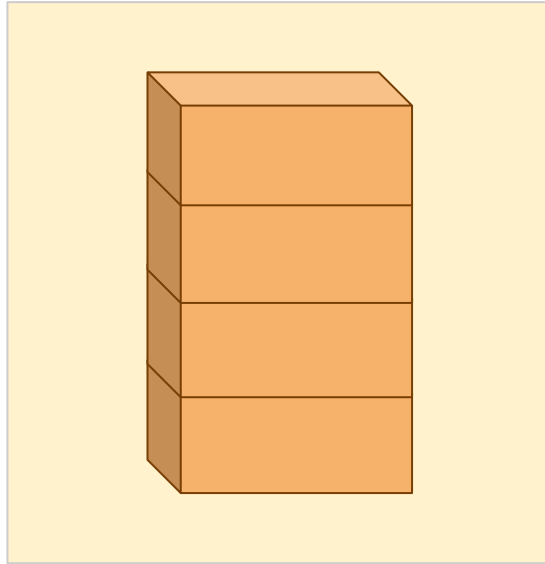


# Casos de uso de Asignación dinámica de memoria

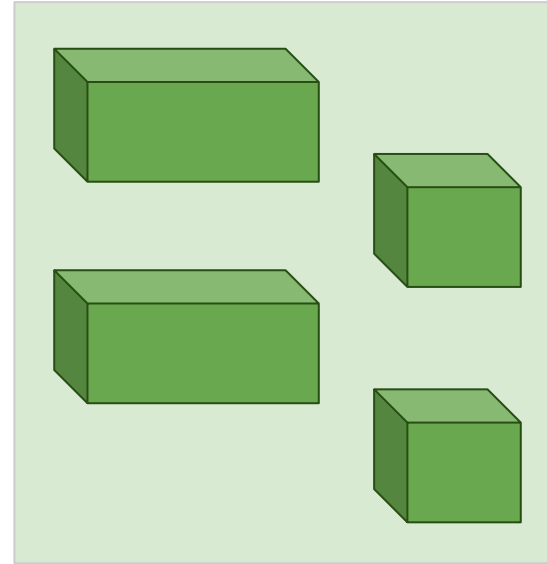
- Cuando necesitemos más memoria que la memoria **stack** admite ✓  

- Cuando sepamos la cantidad de memoria a necesitar durante el tiempo de ejecución del programa. ✓
- Cuando queremos que una variable pueda utilizarse más allá de su ámbito (sin declararla de manera global)

# Memoria

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



Memoria stack

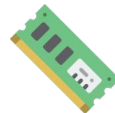


Memoria heap

# Memoria

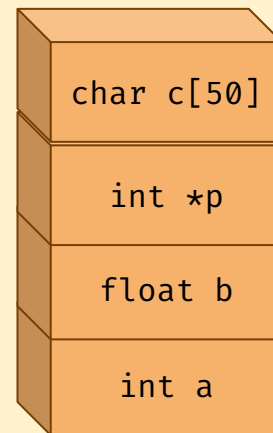
- Una variable puede figurar en la memoria `stack` o en la memoria `heap` dependiendo de cómo trabajemos con ella.
- Hasta el momento siempre utilizamos la memoria `stack`. La memoria dinámica permitirá ubicar nuestras variables en la memoria `heap`.
- La memoria `heap` es una memoria compartida por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

# Memoria stack

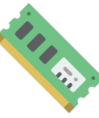


- Cada variable que declaremos en una función (incluso main) se ubica en la memoria stack.
- La memoria stack es limitada. De superar su límite genera una excepción (desbordamiento de pila o stack overflow).

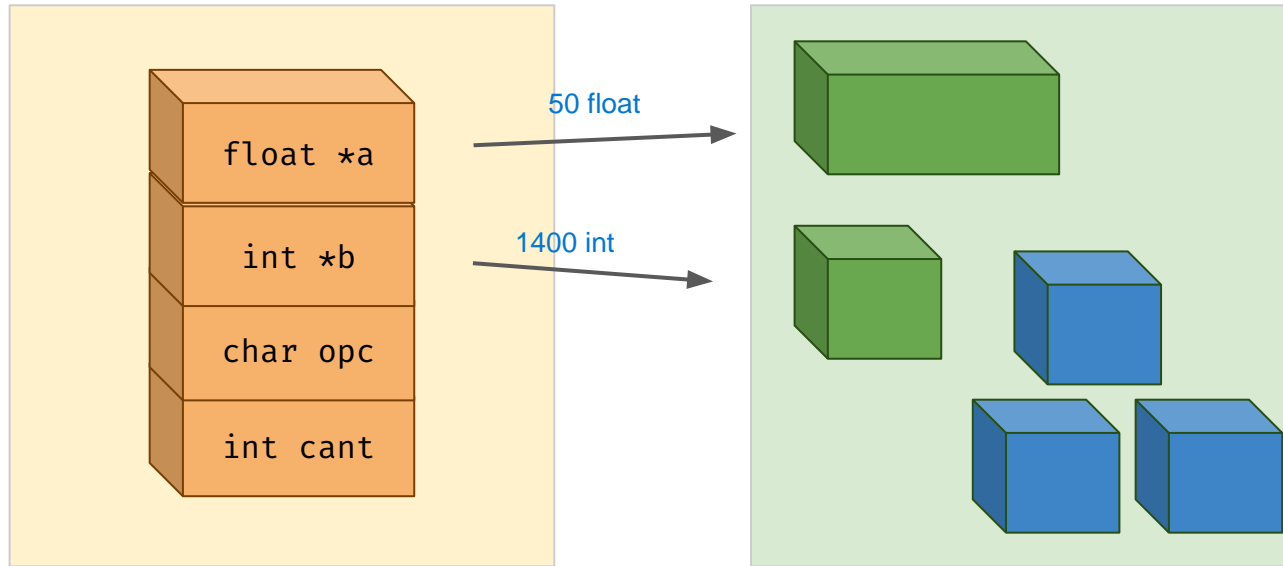
```
int main(){  
    int a, float b;  
    int *p;  
    char c[50];  
}
```



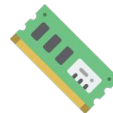
# Memoria heap



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

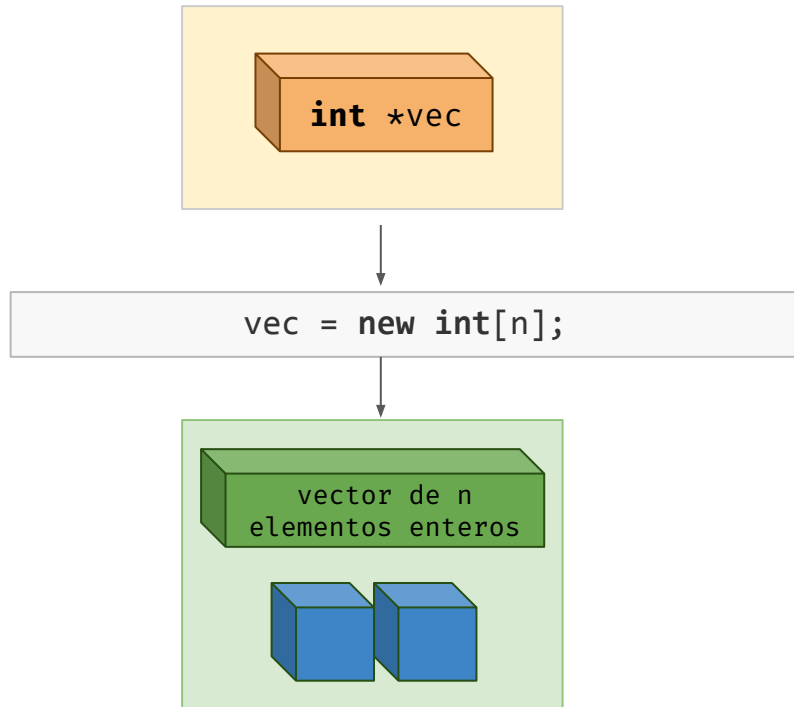


# Operador new



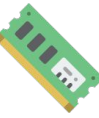
Operador utilizado para pedir memoria dinámica sobre un puntero previamente declarado.

```
#include <iostream>
using namespace std;
int main(){
    /* Memoria dinámica para
    un vector de N elementos */
    int *vec n;      int *vec = nullptr;
    cin >> n;
    vec = new int[n];
    if (vec == NULL) → nullptr
        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.

```
delete []vec; // Liberación de La memoria
```