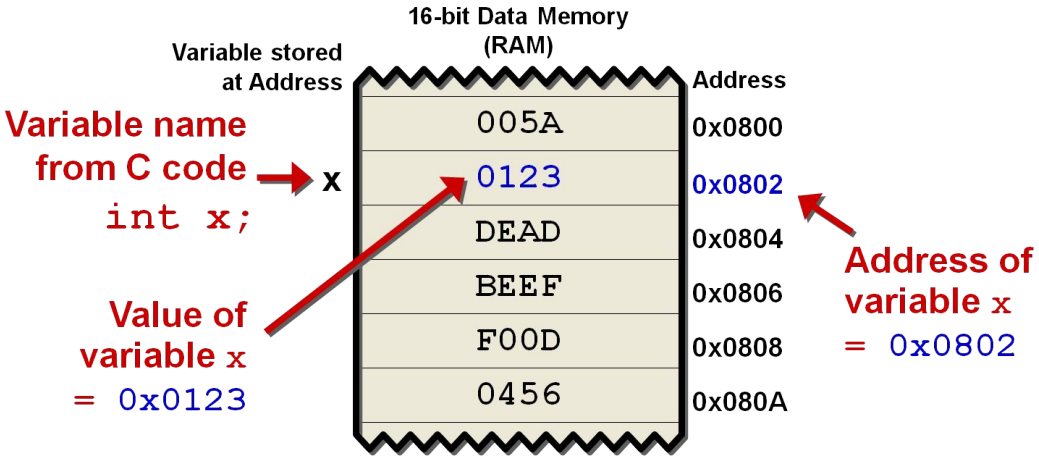
Analysis of Algorithms: Space Complexity

Memoria

Aparte del tiempo, también es muy importante estimar la **memoria** utilizada en un programa

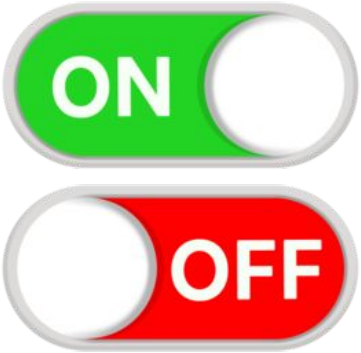
- La memoria (RAM) es como un arreglo muy grande de bytes.
- Al índice de cada posición lo llamamos **dirección de memoria.** Por convención se usan números hexadecimales para representarlas
- Las variables ocupan uno o más bytes (posiciones).



Fuente : Microchip developer help

Bit (b)

- Unidad mínima de información.
- Es un dígito en el sistema binario, puede tomar dos valores: 0 (apagado) y 1 (prendido)
- La computadora solo “entiende” a nivel de bits.



Byte (B)

- Unidad de memoria que consiste en 8 bits
- Esta unidad es usada por conveniencia para medir la cantidad de memoria utilizada por un programa
- También hay unidades derivadas como múltiples de bytes como **kilobytes (KB), megabytes (MB)**

Multiple-byte units V·T·E

Decimal			Binary			
Value	Metric		Value	IEC		Memory
1000	kB	kilobyte	1024	KiB	kibibyte	KB kilobyte
1000^2	MB	megabyte	1024^2	MiB	mebibyte	MB megabyte
1000^3	GB	gigabyte	1024^3	GiB	gibibyte	GB gigabyte
1000^4	TB	terabyte	1024^4	TiB	tebibyte	TB terabyte
1000^5	PB	petabyte	1024^5	PiB	pebibyte	–
1000^6	EB	exabyte	1024^6	EiB	exbibyte	–
1000^7	ZB	zettabyte	1024^7	ZiB	zebibyte	–
1000^8	YB	yottabyte	1024^8	YiB	yobibyte	–
1000^9	RB	ronnabyte			–	
1000^{10}	QB	quettabyte			–	

Orders of magnitude of data

Fuente: Wikipedia

Uso de memoria

Nombre	# bits	# bytes	Rango
bool	8	1	Verdadero (*true*) y falso (*false*)
short	16	2	$[-2^{15}, 2^{15}-1]$
int	32	4	$[-2^{31}, 2^{31}-1]$
long long	64	8	$[-2^{63}, 2^{63}-1]$
float	32	4	[1.175e-38, 3.402e+38]
double	64	8	[2.225e-308, 1.797e+308]
long double	80-96	12	
	128	16	[3.362e-4932, 1.189e+4932]
char	8	1	256 caracteres ASCII.

Uso de memoria

Basado en los datos primitivos, también podemos calcular la cantidad de memoria utilizada de un arreglo.

```
const int MX = 1000;
int A[MX];
```

$$memory = 1000 \times 4B = 4000\ B = 4\ KB$$

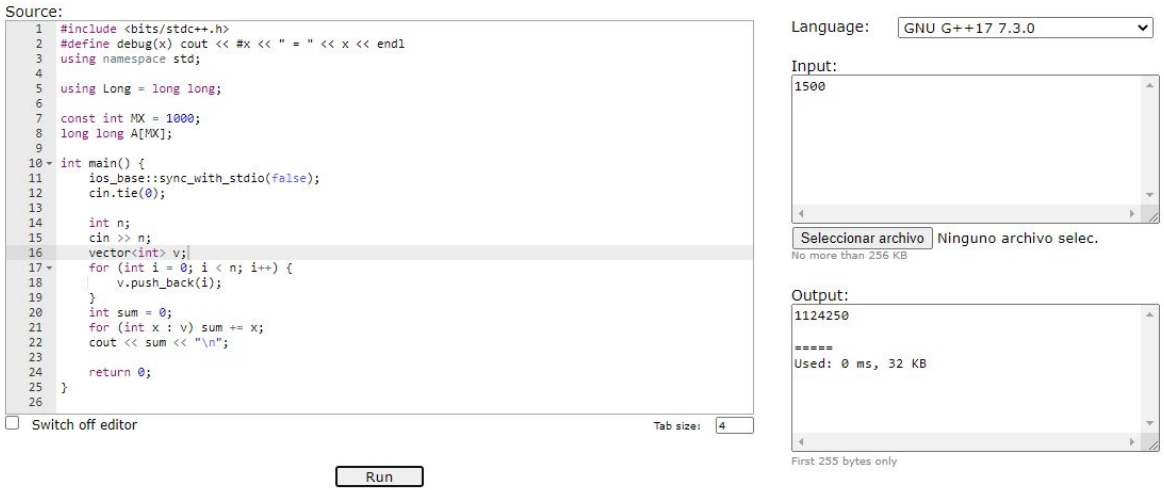
```
const int MX = 1e6;
long long A[MX];
```

$$memory = 10^6 \times 8B = 8 \times 10^6\ B = 8\ MB$$

Calcular la memoria que utiliza un programa

A diferencia del tiempo, calcular la memoria total que usa un programa es más complicado. Normalmente no es necesario tener un cálculo exacto; sin embargo, en caso se necesite puedes usar la herramienta ***Custom Test*** (previamente llamada ***Custom Invocation***) de Codeforces.

Link



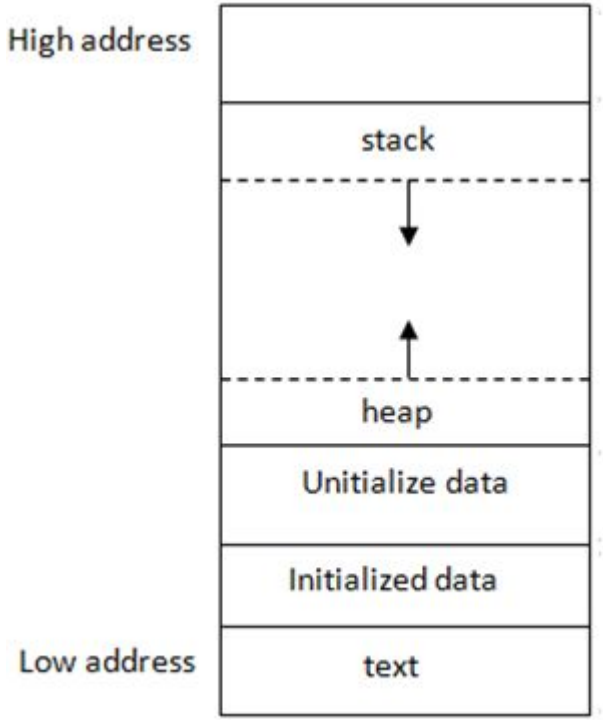
Modelo de memoria en C++

La memoria de un programa en C++ se divide en 4 partes importantes

- **Stack:** Guarda variables locales.
- **Heap:** Memoria dinámica para que el programador pueda alocar.
- **Data:** Guarda variables globales, está separado en data inicializada y no inicializada.
- **Text:** Guarda el código en ser ejecutado propiamente, las instrucciones.

Ejemplos:

- Un **arreglo** estático dentro de una función estará en **stack**
- Un **arreglo** estático global estará dentro de **data**
- Los elementos de un **vector** estarán en dentro de **heap** (debido a la implementación interna de un **vector**), independientemente si es local o global.



Fuente: [1] CS 225 - Univeristy of Illinois

Modelo de memoria en C++

La memoria **stack** también se usa para las variables locales de las funciones. Cuando una función es llamada, se insertan las variables al stack hasta que la función termina su llamada y las elimina del stack.

La memoria del **stack** generalmente está limitada a 1MB o 8MB dependiendo del sistema operativo, y si sobrepasas esto, obtendrás un **stack overflow**. Por otro lado, el **heap** o el segmento de **data** no suelen tener una limitación definida, por lo que pueden utilizar toda la RAM.

Es por esta razón que no se recomienda declarar un **arreglo estático localmente** ya que tendrá mayores limitaciones.

Sin embargo, hay una forma de aumentar la memoria stack al momento de la compilación. La imagen de abajo aumenta la memoria stack hasta 256MB. Muchos jueces como codeforces realizan este aumento del stack, pero hay jueces que no lo hacen.

```
$ g++ -Wl,--stack=268435456 -O2 -std=c++17 sample.cpp
```

Space Complexity

Es la cantidad de memoria utilizada en función de las variables del input. Esto incluye la memoria del propio **input y la memoria auxiliar** para realizar el programa. Sin embargo, de manera práctica, se suele **excluir la memoria del input** del análisis.

La memoria puede cambiar dinámicamente a lo larga de la ejecución de un programa, por lo que nos importará el **pico** donde se use mayor cantidad de memoria.

Calcular el número exacto de bytes sería muy engorroso de todas formas por lo que, para simplificar los cálculos, se suele utilizar notación Big O, de la misma forma que lo utilizamos en *time complexity*.

Ejemplo: Si un programa usa $4n^2 + 2n + 8$ bytes, entonces usa $O(n^2)$ de espacio.

Space Complexity

Normalmente en los problemas, el límite es 256MB lo cual alcanza aproximadamente para $6x10^7$ enteros de tipo *int* y $3x10^7$ enteros de tipo *long long*. Por lo que más o menos deberíamos tratar que la complejidad obtenida no supere el valor de 10^7.

Cuando está muy cerca de ese valor, es más seguro también tomar en cuenta la constante escondida.

Ejemplos

```cpp
int f(int n) {
    vector<int> v(n);
    for (int i = 0; i < n; i++) {
        v[i] = i + 5;
    }
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += v[i];
    }
    return sum;
}
```

$Space\ O(n)$

```cpp
int h(vector<int> &v) {
    int sum = 0;
    for (int x : v) {
        sum += x;
    }
    return sum;
}
```

$Total\ Space\ O(n)$

$Auxiliary\ Space\ O(1)$

```cpp
int g(int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += i + 5;
    }
    return sum;
}
```

$Space\ O(1)$

Ejemplos

```
vector<vector<int>> p(int n) {
    vector<vector<int>> answer;
    int size = 1;
    int value = 1;
    while (n > 0) {
        vector<int> row;
        for (int i = 0; i < size; i++) {
            row.push_back(value);
            value++;
        }
        answer.push_back(row);

        n -= size;
        size++;
    }
    return answer;
}
```

$$Space\ O(2n) = O(n)$$

¡Gracias por su atención!

Referencias

- [1] Jenny Chen, Ruohao Guo (Univeristy of Illinois). Stack and Heap Memory. Recuperado de https://courses.engr.illinois.edu/cs225/fa2022/resources/stack-heap/