

Utilizzo della memoria *heap* in C++

Liceo G.B. Brocchi

Classi terze Scientifico - opzione scienze applicate

Bassano del Grappa, Gennaio 2022

Prof. Giovanni Mazzocchin

La scomodità di malloc, calloc, realloc e free

- Le funzioni della libreria standard C malloc, calloc, realloc e free permettono di gestire la memoria heap
- Avete sicuramente notato che la loro interfaccia non è particolarmente comoda per il programmatore
- Il C++ mette a disposizione 2 operatori integrati nel linguaggio che permettono di fare sostanzialmente la stessa cosa, ma con un interfaccia più semplice:
 - **new**
 - **delete**
- New e delete sono operatori integrati nel linguaggio C++, quindi non serve includere niente di particolare

L'operatore new

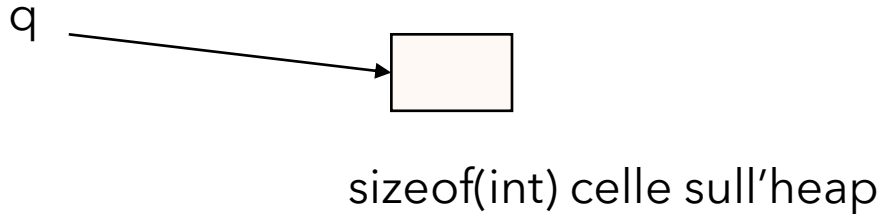
```
int *q = new int; //allocates 1 int on the free store
```

```
double *alloc_double_array(unsigned int n, int val) {  
    double *array = new double[n];  
    //allocated an array of n doubles on the free store  
    for (int i = 0; i < n; i++) {  
        array[i] = val;  
    }  
    return array;  
}
```

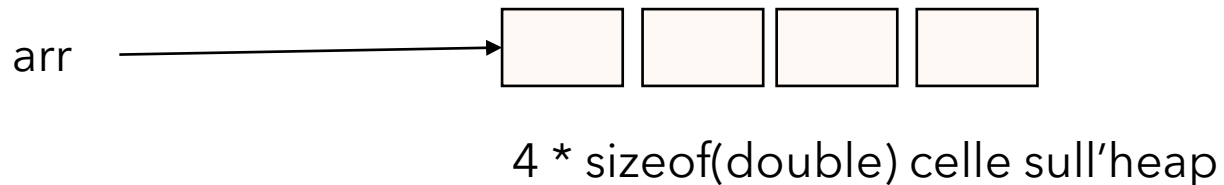
Ovviamente, sotto il tappeto, molto probabilmente, ci sono malloc e calloc. Ma l'interfaccia è più chiara e tipizzata. All'operatore new potete chiedere direttamente di allocare un certo numero di elementi del tipo che volete.

L'operatore new

```
int *q = new int; //memory could be uninitialized
```



```
double *arr = new double[4];
```



Attenti ai tipi

```
#include <iostream>
#include <cstdio>
using namespace std;
```

```
int main() {
    //correct typing
    double *p = new double(3.1415);
    printf("%6.4f\n", *p);
```

```
    //wrong typing: the compiler doesn't allow that
    char *p1 = new double(3.1415);
}
```

```
type_errs.cpp: In function 'int main()':
type_errs.cpp:11:31: error: cannot convert
'double*' to 'char*' in initialization
    11 |     char *p1 = new double(3.1415);
        |
```

Ma se proprio volessi fare conversioni strane...

```
#include <iostream>
#include <cstdio>
using namespace std;
```

```
int main() {
    //correct typing
    double *p = new double(3.1415);
    printf("%6.4f\n", *p);
```

```
    //wrong typing: the compiler doesn't allow that
    char *p1 = new double(3.1415);
}
```

```
type_errs.cpp: In function 'int main()':
type_errs.cpp:11:31: error: cannot convert
'double*' to 'char*' in initialization
    11 |     char *p1 = new double(3.1415);
        |
```

Ma se proprio volessi fare conversioni strane...

Con un **typedef** sto dicendo al compilatore: voglio assolutamente fare questa conversione, quindi non disturbarmi

//right typing

```
float *f_ptr = new float(3.1415);
```

//forcing the conversion with a typedef, written in C style

```
void *v_ptr1 = (void*) f_ptr;
```

//forcing the conversion with a typedef, written in C++ style

```
void *v_ptr2 = static_cast<void*>(f_ptr);
```

Ma se proprio volessi fare conversioni strane...

Con un **typecast** sto dicendo al compilatore: voglio assolutamente fare questa conversione, quindi non disturbarmi

//right typing

```
float *f_ptr = new float(3.1415);
```

//forcing the conversion with a typecast, written in C style

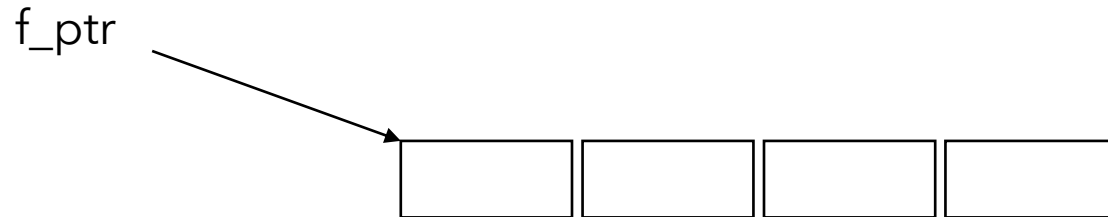
```
void *v_ptr1 = (void*) f_ptr;
```

//forcing the conversion with a typecast, written in C++ style

```
void *v_ptr2 = static_cast<void*>(f_ptr);
```


Ma se proprio volessi fare conversioni strane...

Grazie al casting da `float*` a `void*`, posso accedere ai singoli byte utilizzati per rappresentare 3.1415 sulla macchina. Sul mio computer, `sizeof(float)` è 4 byte



Ma se proprio volessi fare conversioni strane...

```
printf("%p\t%p\n", f_ptr, f_ptr + 1);  
printf("%p\t%p\n", v_ptr1, v_ptr1 + 1);  
printf("%p\t%p\n", v_ptr2, v_ptr2 + 1);  
printf("%6.4f\n", *f_ptr);  
printf("0x%02x\t0x%02x\t0x%02x\t0x%02x\n", *(char*)v_ptr1, *(char*)(v_ptr1 + 1),  
      *(char*)(v_ptr1 + 2), *(char*)(v_ptr1 + 3));  
printf("0x%02x\t0x%02x\t0x%02x\t0x%02x\n", *(char*)(v_ptr2), *(char*)(v_ptr2 + 1),  
      *(char*)(v_ptr2 + 2), *(char*)(v_ptr2 + 3));
```

```
0x557886df3eb0  0x557886df3eb4  
0x557886df3eb0  0x557886df3eb1  
0x557886df3eb0  0x557886df3eb1  
3.1415  
0x56          0x0e          0x49          0x40  
0x56          0x0e          0x49          0x40
```