Elementi di C++ di base

Arrays (vettori), Matrici, Stringhe e Memoria

Operatore *sizeof*

- **sizeof(tipo)** restituisce il numero di bytes necessari a immagazzinare il tipo specificato
- **sizeof(var)** controlla il tipo di var e restituisce il numero di bytes necessari al tipo a cui var appartiene.

```
#include <iostream.h>
main() {
    int a=0;
    cout<< "dimensione di un intero : "<<sizeof(int)<<"\n";
    cout<< "dimensione della variabile a : "<<sizeof(a)<<"\n";
}
```

```
Output:
dimensione di un intero : 4
dimensione della variabile a : 4
```

Esercizio

- Stampare la grandezza in byte dei tipi di dato più comuni in C++
- Provare a far uscire dal range definito l'indice di un vettore e vedere che succede.

Elementi di C++ di base

Arrays (vettori)

Array

Gli array sono collezioni di elementi omogenei

```
int valori[10];
char v[200], coll[4000];
```

Un array di *k* elementi di tipo *T* in è un blocco di memoria contiguo di grandezza

```
(k*sizeof(T))
```


Array - 2

Ogni singolo elemento di un array può essere utilizzato esattamente come una variabile con la notazione:

```
valori[indice]
```

dove indice stabilisce quale posizione considerare all'interno dell'array

Limitazioni

- Gli indici spaziano sempre tra *0* e *k-1*
- Il numero di elementi è fisso - deciso a livello di compilazione
- Il numero di elementi non può variare durante l'esecuzione
- Non c'è nessun controllo sugli indici durante l'esecuzione

Catastrofe (potenziale)

```
....
int a[10];
a[256]=40;
a[-12]=50;
....
```

Inizializzazione

Gli array possono essere inizializzati durante la compilazione e durante l'esecuzione:

```
int primi[] = { 2, 3, 5, 7, 11, 13, 17 };

int a[10];
for(int i=0;i<10;i++) a[i]=i*2;
```


Vettori

Vettore Uni-dimensionale di interi

```
#include <iostream.h>
int main() {
    int v[4];
    int i,k;
    k=0;
    cout<<"Base:"<<endl
      <<&(v[0])<<" "<<v<<endl<<endl;
    for (i=0;i<4;i++) {
      v[i]=k++;
      cout<<&v[i]<<" "<<v[i]<<endl;
    }
    for (i=0;i<4;i++)
       cout<<(v+i)<<" "<< *(v+i)<<endl;
    return 0;
}
```

```
Base:
0012FF74 0012FF74

0012FF74 0
0012FF78 1
0012FF7C 2
0012FF80 3
0012FF74 0
0012FF78 1
0012FF7C 2
0012FF80 3
```

Vettori

Vettore Uni-dimensionale di double

```
Base:
0012FF64 0012FF64

0012FF64 0
0012FF6C 1
0012FF74 2
0012FF7C 3
0012FF64 0
0012FF6C 1
0012FF74 2
0012FF7C 3
```

```
#include <iostream.h>
int main() {
    double v[4];
    int i,k;
    k=0;
    cout<<"Base:"<<endl
      <<&(v[0])<<" "<<v<<endl<<endl;
    for (i=0;i<4;i++) {
      v[i]=k++;
      cout<<&v[i]<<" "<<v[i]<<endl;
    }
    for (i=0;i<4;i++)
       cout<<(v+i)<<" "<< *(v+i)<<endl;
    return 0;
}
```

Vettori

Vettore bidimensionale di int

```
Base:
0012FF70
0012FF70
0012FF70

0 0 0012FF70 0
0 1 0012FF74 1
1 0 0012FF78 2
1 1 0012FF7C 3
0012FF70 0012FF70
0012FF78 0012FF78
0012FF80 0012FF80
0012FF88 0012FF88
```

```
#include <iostream.h>
int main() {
    int v[2][2];
    int i,k=0;
    cout<<"Base:"<<endl<<&(v[0][0])
        << endl <<v<< endl <<&(v[0])
        <<endl<<endl;
    for (i=0;i<2;i++) {
        for (int j=0;j<2;j++) {
           v[i][j]=k++;
           cout<<i<<" "<<j<<" "
              <<&v[i][j]<<" "<<v[i][j]
              <<endl;
        }
    }
    for (i=0;i<4;i++)
       cout<<(v+i)<<" "<< *(v+i)
            <<endl;
    return 0;
}
```

NO!

Vettori

Vettore bidimensionale di int

```
Base:
0012FF70
0012FF70
0012FF70

0 0 0012FF70 0
0 1 0012FF74 1
1 0 0012FF78 2
1 1 0012FF7C 3
0012FF70 0
0012FF78 2
0012FF80 2
0012FF88 3
```

```
#include <iostream.h>
int main() {
    int v[2][2];
    int i,k=0;
    cout<<"Base:"<<endl<<&(v[0][0])
        << endl <<v<< endl <<&(v[0])
        <<endl<<endl;
    for (i=0;i<2;i++) {
        for (int j=0;j<2;j++) {
           v[i][j]=k++;
           cout<<i<<" "<<j<<" "
              <<&v[i][j]<<" "<<v[i][j]
              <<endl;
        }
    }
    for (i=0;i<4;i++)
       cout<<*(v+i)<<" "<< **(v+i)
            <<endl;
    return 0;
}
```

NO!

Vettori

Vettore bidimensionale di int

```
Base:
0012FF70
0012FF70
0012FF70

0 0 0012FF70 0
0 1 0012FF74 1
1 0 0012FF78 2
1 1 0012FF7C 3
0012FF70 0
0012FF74 1
0012FF78 2
0012FF7C 3
```

```
#include <iostream.h>
int main() {
    int v[2][2];
    int i,k=0;
    cout<<"Base:"<<endl<<&(v[0][0])
        << endl <<v<< endl <<&(v[0])
        <<endl<<endl;
    for (i=0;i<2;i++) {
        for (int j=0;j<2;j++) {
           v[i][j]=k++;
           cout<<i<<" "<<j<<" "
              <<&v[i][j]<<" "<<v[i][j]
              <<endl;
        }
    }
    for (i=0;i<4;i++)
       cout<<(*v+i)<<" "<< *(*v+i)
            <<endl;
    return 0;
}
```

OK!

Vettori

Vettore bidimensionale di int

```
0012FF70    v       v[0]    &v[0][0]
0012FF74                    &v[0][1]
0012FF78            v[1]    &v[1][0]
0012FF7C                    &v[1][1]
```

```
#include <iostream.h>
int main() {
    int v[2][2];
    cout<<"sizeof(v)="<<sizeof(v)<<endl;
    cout<<"sizeof(v[0])="<<sizeof(v[0])<<endl;
    cout<<"sizeof(v[0][0])="<<sizeof(v[0][0])<<endl;
    return 0;
}
```

```
sizeof(v)=16
sizeof(v[0])=8
sizeof(v[0][0])=4
```


Vettori

Vettore bidimensionale di int

```
#include <iostream.h>
int main() {
    int v[2][2];
    int i,k=0;
    for (i=0;i<2;i++) {
        cout<<&v[i]<<" "<<(v+i)<<endl;
    }
    cout<<endl;
    for (i=0;i<2;i++) {
        for (int j=0;j<2;j++) {
            cout<<&v[i][j]<<" "
                <<(*(v+i)+j)<<endl;
        }
    }
    return 0;
}
```

```
0012FF70 0012FF70
0012FF78 0012FF78

0012FF70 0012FF70
0012FF74 0012FF74
0012FF78 0012FF78
0012FF7C 0012FF7C
```

Vettori e funzioni

```
#include <iostream.h>
const int N=4;
void printLargest(int v[]) {
// void printLargest(int *v){ è equivalente
// void printLargest(int v[2]){ è equivalente
    int largest=v[0];
    for(int i=1;i<N;i++)
      if (largest<v[i]) largest=v[i];
    cout<< "Il massimo e': "<<largest<<"\n";
}
main() {
    int v[N];
    cout << "Introduci "<<N<<" numeri: ";
    for (int i=0;i<N;i++) cin>>v[i];
    printLargest(v);
    return 0;
}
```

```
Introduci 4 numeri: 3 9 5 1
Il massimo e': 9
```


Vettori e funzioni 2

```
main() {
    int v[N];
    cout<<"dammi "<<N<<
          " numeri:"<<endl;
    for (int i=0;i<N;i++) {
      cin >> v[i];
      } printVector(v);
    invertMax(v);
    printVector(&v[0]);
}
```

```
#include <iostream.h>
const int N=4;
void invertMax(int v[]) {
    int max,indexOfMax;
    indexOfMax=0;
    max=v[0];
    for  (int i=1;i<N;i++)
      if (max<v[i]) {
          max=v[i];
          indexOfMax=i;
          }
    v[indexOfMax]=
      -v[indexOfMax];
}
void printVector(int *v) {
    for  (int i=0;i<N;i++)
      cout << v[i] <<" ";
    cout << endl;
}
```

```
dammi 4 numeri : 3 5 8 1
  3   5   8   1
  3   5  -8   1
```

Vettori e funzioni – 3a

Programma - parte prima:

```
#include <iostream.h>
const int N=4;
void stampaIndirizzi(int m,int* p,int w[],int z) {
    int i;
    cout<<"\nFUNZIONE stampaIndirizzi\n";
    cout<< "indirizzo di m:  "<<&m<<"\n";
    cout<< "indirizzo puntato da p: "<<p<<"\n";
    cout<< "indirizzi del vettore w:\n";
    for (i=0;i<N;i++) {
       cout<<" "<<&w[i];
       }
    cout<<"\n";
    cout<< "indirizzo di z:  "<<&z<<"\n";
}
```


Vettori e funzioni – 3b

Programma - parte seconda:

```
main() {
    int v[N];
    int i, k;
    cout<<"\nMAIN PROGRAM\n";
    cout<< "indirizzo di i:  "<<&i<<"\n";
    cout<< "indirizzo di k: "<<&k<<"\n";
    cout<< "indirizzi del vettore v:\n";
    for (i=0;i<N;i++) {
       cout<<" "<<&v[i];
       }
    cout<<"\n";
    cout<< "indirizzo di v[2]: "<<&v[2]<<"\n";
    stampaIndirizzi(i,&k,v,v[2]);
}
```

Vettori e funzioni - 3c

Output:

```
MAIN PROGRAM
indirizzo di i: 0012FF88
indirizzo di k: 0012FF84
indirizzi del vettore v:
 0012FF74 0012FF78 0012FF7C 0012FF80
indirizzo di v[2]: 0012FF7C

FUNZIONE stampaIndirizzi
indirizzo di m:  0012FF64
indirizzo puntato da p: 0012FF84
indirizzi del vettore w:
 0012FF74 0012FF78 0012FF7C 0012FF80
indirizzo di z:  0012FF70
```

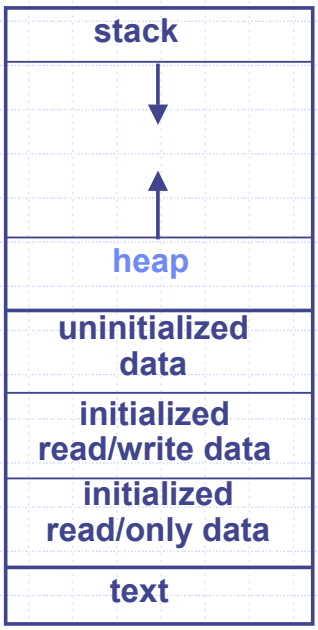

Operatori *new* e *delete*

`new type` alloca `sizeof(type)` bytes in memoria (heap) e restituisce un puntatore alla base della memoria allocata. (esiste una funzione simile usata in C e chiamata `malloc`)

`delete(* p)` dealloca la memoria puntata dal puntatore p. (Funziona solo con memoria dinamica allocata tramite new. Esiste un'analoga funzione in C chiamata `free`).

Il mancato uso della `delete` provoca un insidioso tipo di errore: il **memory leak**.

Il modello di memoria



memoria allocata dalle funzioni (Variabili automatiche)

memoria allocata dinamicamente dal programmatore

Variabili globali e statiche

<- questo é supportato solo da alcuni hardware

Codice eseguibile

Allocazione della memoria

Allocazione statica di memoria (at compile time)

```
main() {
    int a;
    cout<<a<<endl; //NO!
    a=3;
    cout<<a<<endl;
}
```

```
OUTPUT:   1
          3
```

Allocazione dinamica di memoria (at run time)

```
main() {
    int *pa;
    pa=new int;
    cout<<*pa<<endl; //NO!
    *pa=3;
    cout<<*pa<<endl;
    delete(pa);
    cout<<*pa<<endl; //NO!
}
```

```
OUTPUT:   4322472
          3
          8126664
```

Vettori a dimensione variabile

```
#include <iostream.h>
void printVector(int n,int v[]) {
     for  (int i=0;i<n;i++)
       cout<<v[i]<<" ";
     cout<<endl;
}
int main() {
     int *v,i,n;
     cout<<"Quanti elementi? ";
     cin>>n;
     v=new int[n];
     cout<<"dammi "<<n<<" numeri : ";
     for (i=0;i<n;i++) {
       cin>>v[i];
     }
     printVector(n,v);
     delete(v);
     return 0;
}
```

```
Quanti elementi? 4
dammi 4 numeri : 2 4 6 8
2 4 6 8
```


Vettori rivistati

Dichiarare un vettore è in un certo senso come dichiarare un puntatore.

```
v[0]   è equivalente a  *v
```

Attenzione però alla differenza!

```
int v[100];  è "equivalente" a:
                          int *v; v=new int[100];
```

ATTENZIONE!
la prima versione alloca spazio STATICAMENTE (Stack)
la seconda versione alloca spazio DINAMICAMENTE (Heap)

Costrutti idiomatici 1

Inizializzazione di un vettore

```
int i, n=100, a[100], *p; ...
for (p=a; p<a+n; p++) *p=0;
```

o in alternativa

```
int i, n=100, a[100], *p; ...
for (p=&a[0]; p<&a[n]; p++) *p=0;
```

equivale a scrivere:

```
for (i=0; i<n; i++) a[i]=0;
```


Elementi di C++ di base

Stringhe

Costrutti idiomatici 2

Assegnazione condizionale

```
int a,b,c;
a = (b>c) ? b : c;
```

è equivalente a:

```
if (b>c)
    a = b;
else
    a = c;
```



```
#include <iostream.h>
main() {
  for (int i=0; i<127 ; i++)
    cout<<"  "<<(char)i << ((i%8==7 || i==n-1)? '\n':' ');
  return 0;
}
```

Costrutti idiomatici 2

Generazione Della Tabella ASCII

☺ ☻ ♥ ♦ ♣ ♠ ♫ ☼

▶	◀	↕	!!	¶	§	▬	↨
↑	↓	→	←	L	↔	▲	▼
	!	"	#	$	%	&	'
(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7
8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G
H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W
X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g
h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w
x	y	z	{	|	}	~	

Stringhe

In C e C++ non esiste il tipo di dato primitivo "stringa".

Tuttavia le funzioni di libreria di I/O trattano in modo speciale le regioni di memoria contenenti dei "char" (arrays di caratteri)

Sono considerate "stringhe" i vettori di caratteri terminati da un elemento contenente il carattere '\0', indicato anche come NULL.

Un array di lunghezza N può contenere una stringa di lunghezza massima N-1! **(l'N-esimo carattere serve per il NULL)**

Stringhe: vettori di caratteri

```
#include <iostream.h>
#define DIM 8
main() {
     char parola[DIM];
     cout<<"dammi una stringa :";
     cin>>parola;
     cout<<"La stringa e' "<<parola<<endl;
     for (int i=0;i<DIM;i++)
       cout<<parola[i]<<" "<<(int)parola[i]<<endl;
     return 0;
}
```

```
dammi una stringa :pippo
La stringa e' pippo
p 112
i 105
p 112
p 112
o 111
  0
B 66
☺ 1
```

```
dammi una stringa : pi po
La stringa e' pi
p 112
i 105
  0
  0
X 88
B 66
☺ 1
  0
```

Stringhe: vettori di caratteri

```
#include <iostream.h>
main() {
     const int DIM=8;
     char parola[DIM];
     cout<<"dammi una stringa :";
     cin.getline(parola,DIM);
     cout<<"La stringa e' "<<parola<<endl;
     for (int i=0;i<DIM;i++)
       cout<<parola[i]<<" "<<(int)parola[i]<<endl;
     return 0;
}
```

```
dammi una stringa :pippo
La stringa e' pippo
p 112
i 105
p 112
p 112
o 111
  0
B 66
☺ 1
```

```
dammi una stringa : pi po
La stringa e'  pi po
  32
p 112
i 105
  32
p 112
o 111
  0
B 66
```


Stringhe: vettori di caratteri


```
#include <iostream.h>
main() {
     const int DIM=8;
     char parola[DIM];
     cout<<"dammi una stringa :";
     cin<<ws;
     cin.getline(parola,DIM);
     cout<<"La stringa e' "<<parola<<endl;
     for (int i=0;i<DIM;i++)
       cout<<parola[i]<<" "<<(int)parola[i]<<endl;
     return 0;
}
```

```
dammi una stringa :pippo
La stringa e' pippo
p 112
i 105
p 112
p 112
o 111
  0
B 66
☺ 1
```

```
dammi una stringa : pi po
La stringa e' pi po
p 112
i 105
  32
p 112
o 111
  0
B 66
☺ 1
```

Operatori su stringhe

Nella libreria string.h sono predefinite una serie di funzioni operanti su stringhe.

La libreria va inclusa con il comando `#include <string.h>`

Le funzioni di uso più frequente sono:

```
char *strcpy(a,b); /*copia b su a*/
int strcmp(a,b);   // restituisce<0 se a<b,0 se a=b,>0 se a>b
char *strcat(a,b); /* appende b in coda ad a*/
size_t strlen(a);  //restituisce la lunghezza della stringa a
```

(Abbiamo assunto la definizione: `char *a, *b;`)

Esercizio:

Implementare queste funzioni (ricordando la definizione di stringa)

Stringhe e funzioni

```
#include <iostream.h>
#include <string.h>
#define DIM 10
main() {
    char parola[DIM];
  //char * altraparola;  NO!
    char altraParola[DIM];
    cout<<"dammi una stringa :";
    cin>>ws;
    cin.getline(parola,DIM);
    cout<<"La stringa inserita e' \""
      <<parola<<"\"\n";
    strcpy(altraParola,parola);
    cout<<"Il contenuto di altraParola e' \""
      <<altraParola<<"\"\n";
    return 0;
}
```

Stringhe

Attenzione alle sottigliezze!

```
char *mystring="Pippo";        è "equivalente" a:
      char *mystring;
      mystring=new char[6];
      mystring[0]='P';mystring[1]='i';...;
      ;…; mystring[4]='o';mystring[5]='\0';
```

(in realtà c'è una differenza riguardo a dove in memoria viene riservato il posto: stack o heap)

```
char *frase;
```

definisce il puntatore ma NON alloca spazio in memoria!:

Costrutti idiomatici 3

Copia di zone di memoria

(Utile per stringhe, vettori ecc.)

```
int n;
char *p, *q; ...
while (n--) *p++ = *q++;
```

copia n celle di memoria a partire dal puntatore q nella zona puntata da p.

Attenzione! alla fine p e q puntano alla fine delle rispettive zone!

Costrutti idiomatici 4

Assegnazioni

Le due righe seguenti hanno SIGNIFICATI DIVERSI

(ATTENZIONE! sorgente di MOLTI errori)

```
if (a==b) cout << "OK\n";

if (a=b) cout << "OK\n";
```

La seconda è equivalente a:

```
a=b;
if (a!=0) cout << "OK\n";
```