UNIVERSITÀ
DI PARMA

C++ dati strutturati

dati strutturati

- strutture per trattare in modo unitario dati fra loro correlati
  - ***array*** : ripetizione di elementi omogenei
    - l'array è il meccanismo utilizzato per definire vettori e matrici
  - ***record***: giustapposizione di elementi (anche non omogenei)
    - il record è il meccanismo di base utilizzato negli archivi di dati (basi di dati):
      - ad esempio, per rappresentare le informazioni relative ad uno studente, quali la matricola, il nome, l'anno di corso, si usa una struttura record con i campi matricola, nome, …

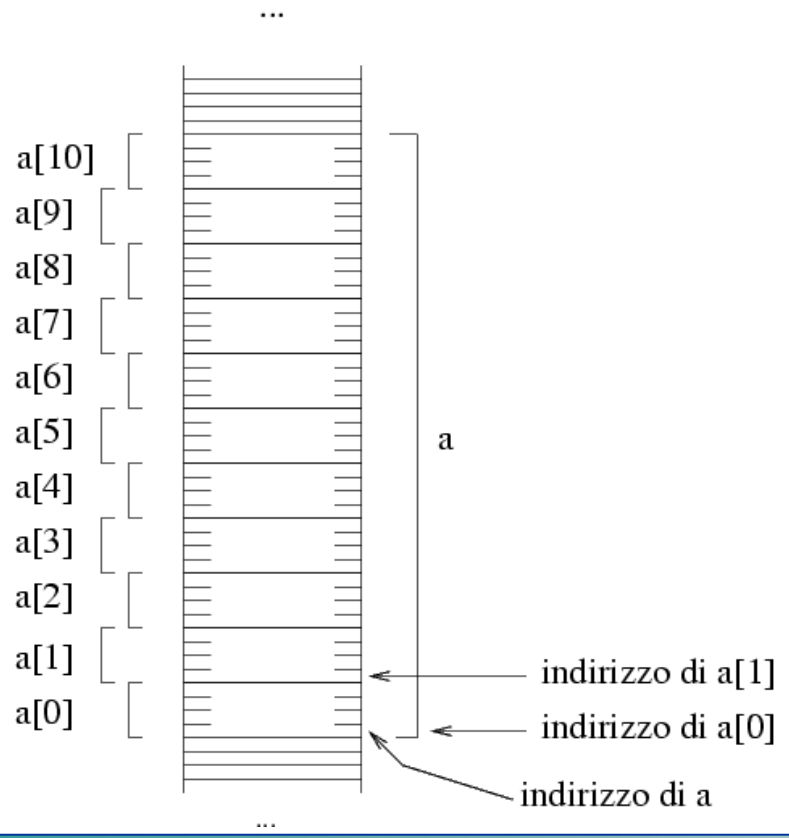
- ***vettori*** e ***matrici*** si dichiarano tramite il costruttore di tipo **[ ]** (costruttore array)

```
<var-array> ::= <tipo-elementi> <identif> <array>;
<array> ::= <costr-array> | <costr-array> <array>
<costr-array> ::= [<espres_costante_intera>]
```

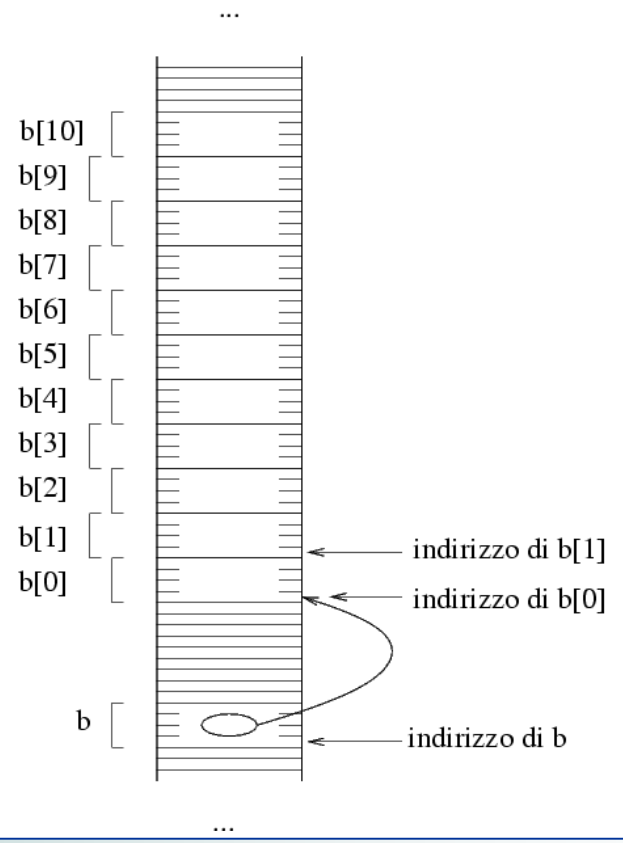
- il valore **<espres_costante_intera>** stabilisce il numero degli elementi del vettore (dimensione del vettore)
- esempi

```
int v[10];              /* variabile v come vettore di 10 int */
int v1[10],v2[5];       /* variabili array v1 e v2 */
float m[10][10];        /* variabile m, matrice 10X10 float*/
```

array ***statico***



array ***dinamico***



indice: accesso agli elementi

- l'indice di accesso a un elemento è un'espressione con valore ***intero***
- l'indice assume valori interi compresi tra ***0*** e ***dimensione-1***
- ***attenzione*** : non c'è controllo per il superamento dei limiti (***buffer overrun***)
- è buona pratica di programmazione utilizzare una costante simbolica per definire la lunghezza di un array:

```
#define N 100
int a[N];
```

inizializzazione

- ***inizializzazione*** di un array

```
int v[5] = {1,2,3,4,6};
int v[5] = {1,6}; /* solo i primi due elementi, gli altri hanno valore indefinito */
int v[5] = {1,2,3,4,6,8}; /* errore */
int v[] = {1,2,121 }; /* array di tre elementi */
```

- `v[i]` identifica una «variabile» che denota l'elemento ***i-esimo*** di `v`
- ***assegnamento*** del valore k all'elemento i-esimo di v : `v[i] = k;`
- ***input*** dell'elemento i-esimo di v : `cin >> v[i];`

vettori: memorizzazione

- i singoli elementi di un array (monodimensionale) sono memorizzati consecutivamente in memoria:

```
int A[100]={3,4,8};
```

- l'accesso ad un elemento avviene specificando l'indice

A[0]	3	0FFC
A[1]	4	1000
A[2]	8	1004
A[3]	0	1008
A[4]	0	100C
A[5]	0	1010
A[6]	0	1014
A[7]	0	1018
A[8]	0	101C

vettori: esercizi di esempio

- acquisire da input un vettore ***v1*** di ***n1*** numeri interi ***pari*** compresi tra ***a*** e ***b*** (estremi inclusi) [controllare l'input]
- determinare e stampare a video il valore massimo del vettore ***v1***; quindi determinare e stampare a video le posizioni del vettore ***v1*** che contengono tale valore
- acquisire da input un vettore ***v2*** di ***n2*** numeri interi ***dispari non decrescenti*** [controllare l'input]
- stampare a video il contenuto dell'array ***v2***

soluzione (1)

```
/* acquisire da input  un vettore  v1 di n1 numeri interi pari compresi  tra a e b
(estremi inclusi) [controllare l'input] */
#include <iostream>
using namespace std;
int main() {
   const int n1 = 5;
   int a,b,max; int v1[n1];
   cout << "valore inferiore: "; cin >> a;
   cout << "valore superiore: "; cin >> b;
   for (int i=0;i<n1;i++)
      do {
         cout << "elemento di indice " << I << " : ";
         cin >> v1[i];
      } while (v1[i] < a || v1[i] > b || v1[i]%2 != 0);
…
```

soluzione (2)

```
/* determinare  e stampare a video il valore massimo del vettore v1; quindi
determinare  e stampare a video le posizioni del vettore v1 che contengono tale
valore
 */
int max = v1[0];
for (int i=1;i<n1;i++)
   if (v1[i] > max)
      max = v1[i];
cout << "valore massimo: " << max << endl;
for (int i=0;i<n1;i++)
   if (v1[i] == max)
      cout << "l'elemento di indice " << i << " ha valore massimo" << endl;
```

soluzione (3)

```
/*acquisire da input  un vettore  v2 di n2 numeri interi dispari non decrescenti
[controllare l'input] stampare a video il contenuto dell'array v2
 */
const int n2=8; int v2[n2];
do {
   cout << "Inserisci l'elemento di indice 0: ";
   cin >> v2[0];
} while (v2[0] % 2 == 0);
for (int i=1;i<n2;i++)
   do {
      cout << "Inserisci l'elemento di indice " << i << " : ";
      cin >> v2[i];
   } while (v2[i] % 2 == 0 || v2[i] < v2[i-1]);
for (int i=0;i<n2;i++)
   cout << "v2[" << i << "] = " << v2[i] << endl;
```

funzioni con parametro array

- per i parametri di tipo array il metodo utilizzato è ***call by reference***
- è necessario fornire alla funzione anche la ***dimensione*** dell'array
- per evitare effetti collaterali non voluti è possibile definire il parametro come ***const*** impedendone così la modifica

```
/* visualizza l'array a di n elementi */
void visualizza(const double a[], int n) {
        // cout << "dimensione array a " << sizeof a << " bytes " << endl;
        for(int i=0;i<n;i++)
                cout << "a[" << i << "] = " << a[i] << endl;
}
…
visualizza(v,10);                    // in fase di esecuzione della funzione si specifica il
                                     // nome dell'array (senza []) e la sua dimensione
```

esempi di funzioni

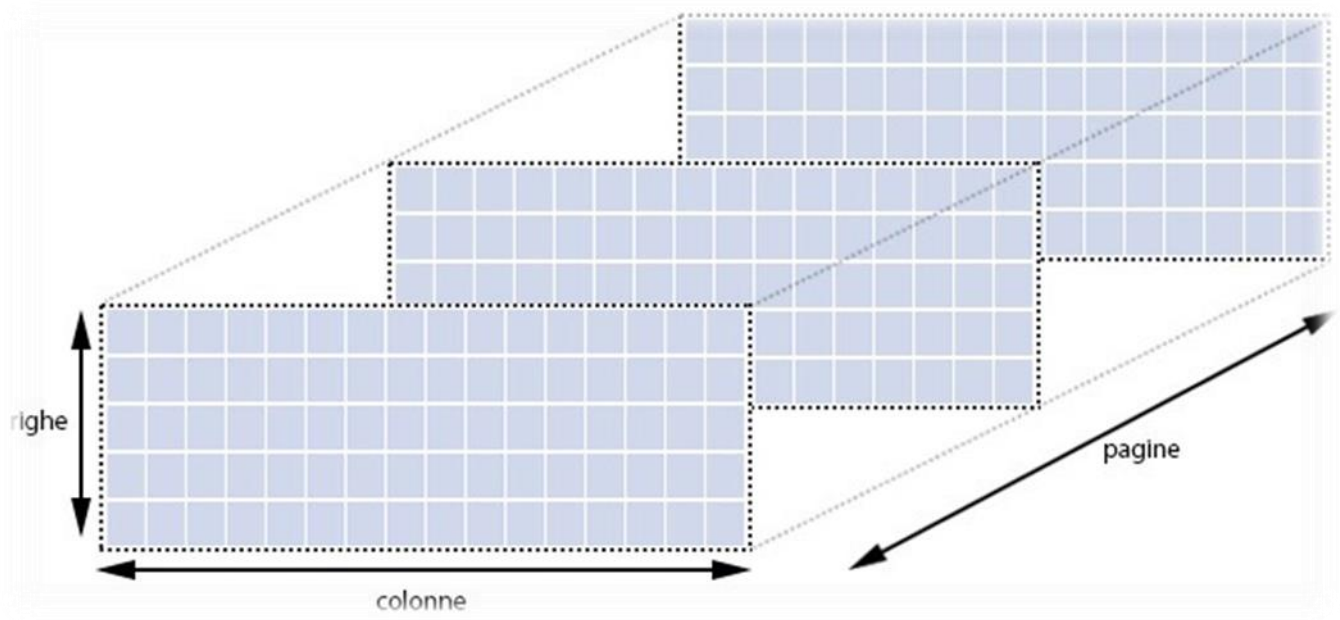
```
void visualizza(const double a[], int n) {
   // cout << "dimensione array a " << sizeof a << " bytes " << endl;
   for(int i=0;i<n;i++)
   cout << "a[" << i << "] = " << a[i] << endl;
}

void dimezza(double a[], int n) {
   for(int i=0;i<n;i++)
      a[i] = a[i] / 2;
}

double max(double a[], int n){
   double m = a[0];
   for(int i=1;i<n;i++)
      if (a[i]>m)
         m = a[i];
   return m;
}
```

```
int main() {
   srand(std::time(nullptr));
   double v[10];
   for(int i=0;i<10;i++)
      v[i] = (rand() % 100 + 1) / 10.0;
   cout << "dimensione array v " << sizeof v
        << " bytes " << endl;
   visualizza(v,10);
   dimezza(v,10);
   visualizza(v,10);
   cout << "valore massimo " << max(v,10) << endl;
}
```

array multidimensionali



array multidimensionali

- gli array multidimensionali possono essere considerati array di array di array …

```
int a[100];          /* array monodimensionale */
int b[2][7];         /* array bidimensionale */
int c[5][3][2];      /* array tridimensionale */
```

- gli array multidimensionali sono memorizzati in sequenza lineare ma nel caso di array bidimensionali (matrici) risulta utile pensarli organizzati in righe e colonne

```
int a[3][5];
```

	col.1	col.2	col.3	col.4	col.5
riga 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
riga 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
riga 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

inizializzazione

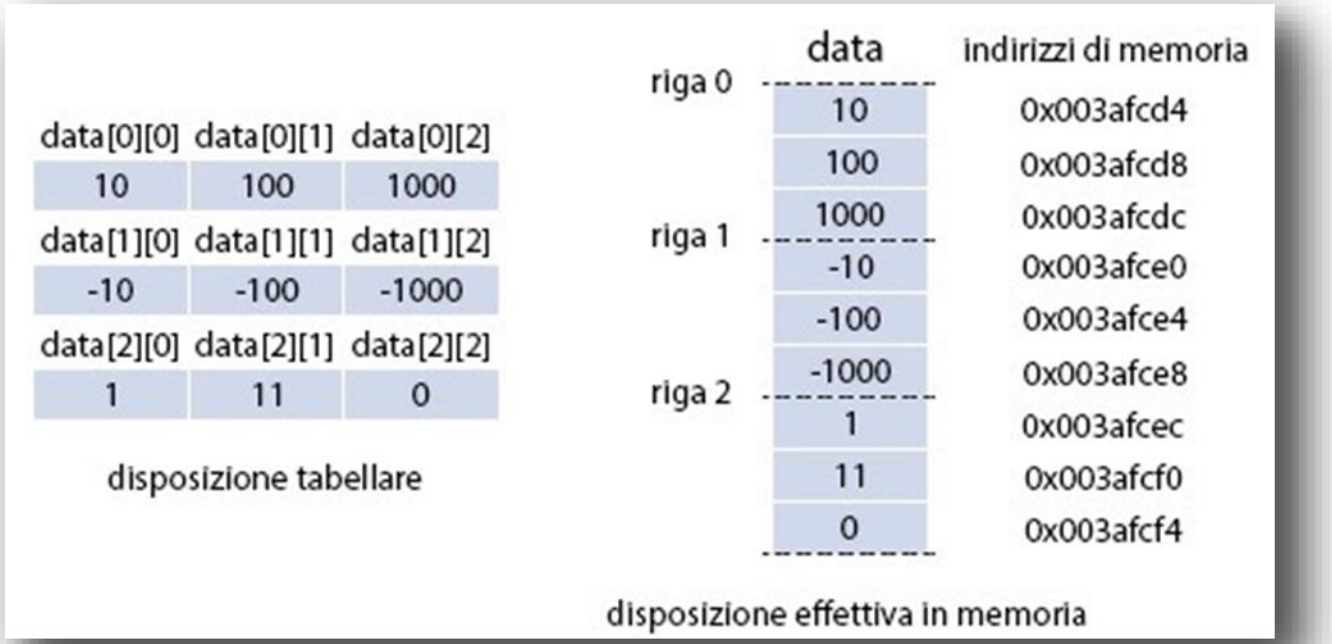
- esistono vari modi, tra loro equivalenti:

```
int a[2][3] = {1,2,3,4,5,6};
int a[2][3] = {{1,2,3},{4,5,6}};
int a[ ][3] = {{1,2,3},{4,5,6}};
```

- se non vengono inizializzati i valori presenti nell'array sono impredicibili, è possibile ottenere velocemente l'azzeramento in questo modo:

```
int a[2][2][3] = {0};
```

- gli elementi vengono memorizzati ***per righe*** in indirizzi contigui di memoria



data[0][0]	data[0][1]	data[0][2]
10	100	1000
data[1][0]	data[1][1]	data[1][2]
-10	-100	-1000
data[2][0]	data[2][1]	data[2][2]
1	11	0

disposizione tabellare

	data	indirizzi di memoria
riga 0	10	0x003afcd4
	100	0x003afcd8
	1000	0x003afcdc
riga 1	-10	0x003afce0
	-100	0x003afce4
	-1000	0x003afce8
riga 2	1	0x003afcec
	11	0x003afcf0
	0	0x003afcf4

disposizione effettiva in memoria

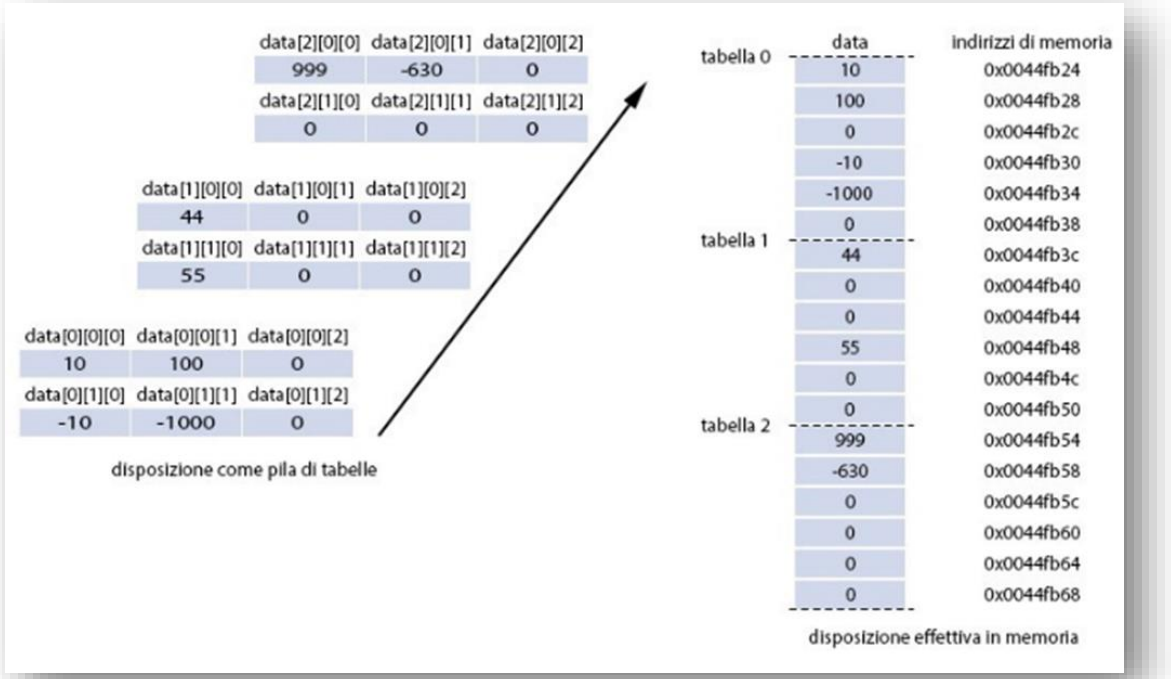
matrice in memoria

```
#include <iostream>
using namespace std;
int main() {
	int m[5][4];
	int *pm;
	pm = &m[0][0];
	for (int r=0;r<5;r++)
		for (int c=0;c<4;c++)
			m[r][c] = 10*r+c;
	for (int i=0;i<5*4;i++) {
		cout << "indirizzo " << pm << " valore " << *pm << endl;
		pm++;
	}
}
```

```
indirizzo 0x6dfea0 valore 0
indirizzo 0x6dfea4 valore 1
indirizzo 0x6dfea8 valore 2
indirizzo 0x6dfeac valore 3
indirizzo 0x6dfeb0 valore 10
indirizzo 0x6dfeb4 valore 11
indirizzo 0x6dfeb8 valore 12
indirizzo 0x6dfebc valore 13
indirizzo 0x6dfec0 valore 20
indirizzo 0x6dfec4 valore 21
indirizzo 0x6dfec8 valore 22
indirizzo 0x6dfecc valore 23
indirizzo 0x6dfed0 valore 30
indirizzo 0x6dfed4 valore 31
indirizzo 0x6dfed8 valore 32
indirizzo 0x6dfedc valore 33
indirizzo 0x6dfee0 valore 40
indirizzo 0x6dfee4 valore 41
indirizzo 0x6dfee8 valore 42
indirizzo 0x6dfeec valore 43
```

array multidimensionali in memoria

- nella memoria del computer gli elementi di un array (indipendentemente dal numero delle sue dimensioni) sono memorizzati ***sequenzialmente*** in ***indirizzi contigui*** di memoria



matrici come parametro

- se si vuole passare una matrice come parametro ad una funzione è ***necessario*** specificare il ***numero di colonne***
- questo permette alla funzione di interpretare correttamente la posizione degli elementi
- ***esempio***: funzione che carica nell'array ***somma*** la somma degli elementi presenti in ogni ***riga*** della matrice m

esempio

```
void sommaRighe(double m[][10], int nr, double somma[]) {
   double s; // somma
   int r,c;  // indici di riga e colonna
   for(r=0;r<nr;r++) {     // per ogni riga
      s = 0.0;             // inizializzazione somma
      for (c=0;c<10;c++)   // per ogni elemento della riga
            s = s+ m[r][c];
      somma[r] = s;
   }
}
```

overloading

- una famiglia di funzioni aventi lo ***stesso nome***, ma un ***diverso*** set di argomenti (***signature***) è detta in rapporto di ***overloading***, o ***sovraccaricata***
- si può parlare di overloading di ***funzioni***, di ***costruttori*** e di ***operatori***
- *sovraccaricare il costruttore di una classe è una pratica comune per gli sviluppatori di librerie, in quanto permette di fornire allo sviluppatore finale diverse modalità per inizializzare gli oggetti.*
- in C++ è ammesso l'***overloading degli operatori***

la classe

STRING

string

- una stringa è una sequenza di 0 o più caratteri racchiusi fra doppi apici
  - `string corso = "programmazione di applicazioni software";`
- la classe string non fa parte del linguaggio C++ ma è inclusa nella libreria standard
- per utilizzare oggetti della classe string (variabili di tipo string) è necessario includere la libreria
  - `#include <string>`

operatori sulle stringhe

- operazione di indicizzazione **[ ]**
  - l'indice del primo carattere è 0 e quello dell'ultimo è uguale alla lunghezza della stringa -1
- operatore di concatenazione **+**
  - almeno uno dei due operandi deve essere un oggetto di tipo string
- funzioni definite sulle stringhe
  - **length**() o **size**()
    - restituisce il numero di caratteri presenti nella stringa
  - **find**(**s**)
    - ricerca la prima occorrenza della stinga s nella stringa in cui è invocata
  - **substr**(**i_inizio**,**lung**)
    - restituisce la sottostringa di lunghezza lung a partire dal carattere di indice i_inizio

string - esempio

```
#include <iostream>
#include <string>
using namespace std;
int main() {
            string s1,s2,s3;
            s1 = "programmazione";
            s2 = s1 + " di applicazioni " + "software";
            // error: invalid operands of types '...' and '...' to binary 'operator+'
            // s3 = "programmazione" + " di applicazioni software";
            s2[0] = 'P';                        // sostituzione del primo carattere della stringa
            cout << "contenuto della stringa s2: " << s2 << endl;
            cout << "numero di caratteri della stringa s2: " << s2.length() << endl;
            s3 = "Ingegneria dei sistemi informativi";
            int pos;
            pos = s3.find("in");
            cout << "nella stinga " << s3 << endl << "la sottostringa " << "in"
                 << " si trova in posizione " << pos << endl;
            pos = s3.find("out");
            cout << "nella stinga " << s3 << endl << "la sottostringa " << "out"
                 << " si trova in posizione " << pos << endl;
            cout << s3.substr(15,7) << endl;
    return 0;
}
```

http://www.cplusplus.com/reference/string/string/