1

Tipi di dati

Tipi di dati personalizzati

```
typedef float coordinata;
coordinata z,t;
```

Tipi di dati composti

```
struct punto {
        coordinata x;
        coordinata y;
        }
punto origine;
origine.x=0.0;
origine.y=0.0;
```

2

Puntatori a strutture

Operat. di dereferenziazione di struttura: –>

```
main()
{
   struct point {
      int x;
      int y;
      enum color {BLACK, BLUE, RED, GREEN};
    };
    struct point a, *pa;
    a.x = 3; a.y = 5; a.color=GREEN;
    pa = &a;
    cout<<a.x<<" "<<pa->y <<" "<<(*pa).color;
}
```

3

Elementi di C++ di base

Arrays (vettori)

4

Array

Gli array sono collezioni di elementi omogenei

```
int valori[10];
char v[200], col1[4000];
```

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

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

5

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

6

Limitazioni

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

7

Catastrofe (potenziale)

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

8

Vettori

Vettore Uni-dimensionale di interi

```
Base:
0012FF74 0012FF74

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

```
#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;
}
```

9

Vettori

Vettore Uni-dimensionale di double

```
#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;
}
```

```
Base:
0012FF64 0012FF64

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

10

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
```

11

Vettori e funzioni 2

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

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

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

12

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;
```

13

Operatori *new* e *delete*

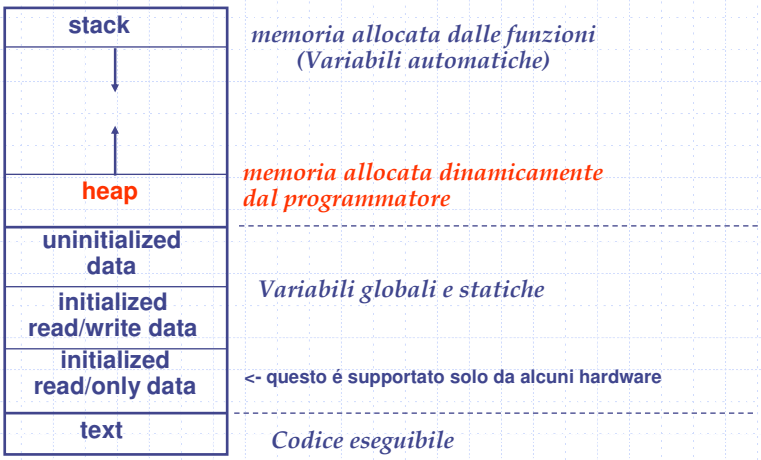
`new type` alloca `sizeof(type)` bytes in memoria (heap) e restituisce un puntatore alla base della memoria allocata. (esiste una funzione simile usata in C e chiamata `malloc`)

`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**.

14

Il modello di memoria



15

Memory leak

```
main() {
   while (true) {
      …
      f();
      …
   };
}
```

```
f() {
   int *pa;
   pa=new int;
   …
   // delete(pa);
   …
}
```

Per ogni new ci deve essere una delete!

16

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
```

17

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)

18

Elementi di C++ di base

Stringhe

19

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
```

23

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)

24

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;
}
```

25

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,heap o zona variabili globali)

```
char *frase;
```

definisce il puntatore ma NON alloca spazio in memoria!:

26

Costrutti idiomatici 2

Copia di zone di memoria

(Utile per stringhe, vettori ecc.)

```
int n=6;
char * z="pippo"; char w[6];
char * p=z; char * q=w; ...
while (n--) *q++ = *p++;
```

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

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

27

Costrutti idiomatici 3

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:

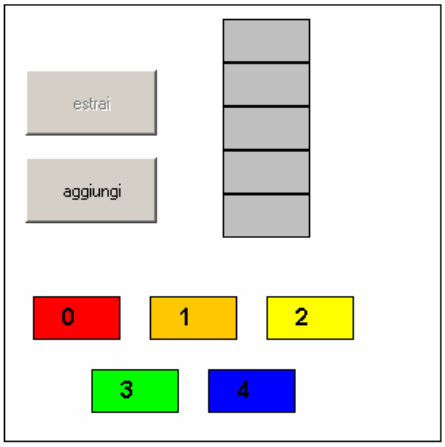
```
a=b;
if (a!=0) cout << "OK\n";
```

28

Il nostro esempio guida: La costruzione di uno stack

29

Costruiamo uno stack

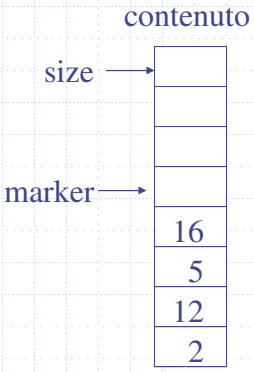


stackapplet.html

30

```
#include <iostream.h>
#include <cassert>
#define DEBUG

struct Pila {
  int size;
  int defaultGrowthSize;
  int marker;
  int * contenuto;
} ;
```



31

```
Pila  * crea(int initialSize) {
//crea una Pila
#ifdef DEBUG
  cout<<"entro in crea"<<endl;
#endif
  Pila  * s= new Pila ;
  s->size=initialSize;
  s->defaultGrowthSize=initialSize;
  s->marker=0;
  s-> contenuto=new int[initialSize];
 return s;
}
```

32

```
void distruggi(Pila * s) {
//distruggi lo Pila
#ifdef DEBUG
  cout<<"entro in destroy"<<endl;
#endif
  delete [](s->contenuto);
  delete s;
}
```