Sommario

- Tipo stringa
- Tipo booleano

- Ricorsione:
  - Definizione
  - Chiamata di funzione
  - Ricorsione in coda
  - Ricorsione non in coda
  - Ricorsione eccessiva
  - Esempio di ricorsione: ricerca binaria

Tipo stringa

- Il C++ fornisce due rappresentazioni delle stringhe: la stringa di caratteri stile C e la *classe stringa* introdotta nella libreria standard del C++.

Stile C.

- La stringa stile C è memorizzata in un *array di caratteri* ed è in genere manipolata attraverso un puntatore `char *`.
- La libreria standard del C offre una serie di funzioni per manipolare tale tipo di stringhe.

Tipo stringa: stile C

- In genere si percorre una stringa stile C usando l'aritmetica dei puntatori, fino a quando si raggiunge il *carattere nullo di terminazione*.

```
char str[] = {'p','r','o','v','a','\0'};
char *p=str;
int i=0;
while(*p++)  i++;
cout<<"La stringa "<<str<<" e` lunga "<<i<<" caratteri"<<endl;
```

- Il codice produce il risultato

```
La stringa prova e` lunga 5 caratteri
```


Tipo stringa: `string`

- Tuttavia a causa della sua rappresentazione a *basso livello* la stringa stile C è usata solo in contesti particolari.
- In genere si usa la classe stringa della libreria standard C++. Per esempio, alcuni comportamenti di tale *tipo di dato astratto* sono:
  - Inizializzazione con una sequenza di caratteri o un altro *oggetto* stringa.
  - Supporto per la copia (funzione `strcpy()` in C).
  - Supporto per l'accesso in lettura scrittura di singoli caratteri.
  - Supporto per il confronto (funzione `strcmp()` in C).

Tipo stringa: `string`

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

void test1();

int main(){
   test1();
   return 0;
}

void test1(){
   string st("prova");
   cout<<"La stringa "<<st<<" e` lunga "<<st.size()<<" caratteri"<<endl;
   string s1;
   if (s1.empty()) cout<<"La stringa e` vuota"<<endl;
   string s2(st);
   if (s2==st) cout<<"Le stringhe sono uguali"<<endl;
   s2[0]='P';
   string s3=st+" "+s2;
   cout<<s3<<endl;
}
```

Una possibile uscita

```
La stringa prova e` lunga 5 caratteri
La stringa e` vuota
Le stringhe sono uguali
prova Prova
```

`st` è un oggetto di tipo `string`

Confronto

Come un array

Concatenazione

Tipo stringa: `string`

- Per convertire un oggetto di tipo `string` in una stringa di caratteri stile C è necessario invocare esplicitamente il metodo `c_str()` sull'oggetto.

```
void test2(){
      string st("Informatica Medica");
      const char *p = st.c_str();
      cout<<p<<endl;
}
```

Una possibile uscita

```
Informatica Medica
```


Tipo `bool`

- Ad un oggetto di tipo `bool` è possibile assegnare i valori logici `true` e `false`.
- Le variabili di tipo `bool` sono convertire al tipo `int` quando è necessario un valore aritmetico: `false` diventa `0` e `true` diventa `1`.
- Se necessario, un valore zero o un puntatore nullo sono convertiti in `false`, tutti gli altri valori sono convertiti in `true`.

Tipo `bool`

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

int main(){
  bool f = true;
  int i=3;
  if (f){
      i+=f;
      cout<<"i = "<<i<<endl;
  }

  int j = 33;
  if ( j )
      cout <<"Vero"<<endl;
}
```

Una possibile uscita

```
i = 4
Vero
```

Ricorsione

- La ricorsione è la capacità di una funzione (metodo) di *richiamare se stesso*.

- Una definizione ricorsiva consiste in due parti:
  - *Caso base*: vengono elencati gli elementi di base che sono i blocchi costitutivi dell'insieme;
  - *Caso induttivo/ricorsivo*: vengono fornite le regole per costruire nuovi elementi a partire dagli elementi di base o da elementi che sono già stati costruiti.

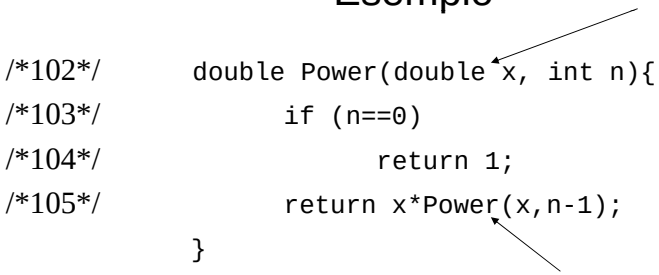
Ricorsione

- La funzione che definisce l'elevamento di un numero *x* ad una potenza non negativa *n* è un esempio di funzione ricorsiva:

$$x^n = \begin{cases} 1 & \text{se } n = 0 \text{ (caso base)} \\ x \cdot x^{n-1} & \text{se } n > 0 \text{ (caso ricorsivo)} \end{cases}$$

- Implementiamo questa funzione in C++ e analizziamo come avviene un'invocazione ricorsiva.

Esempio



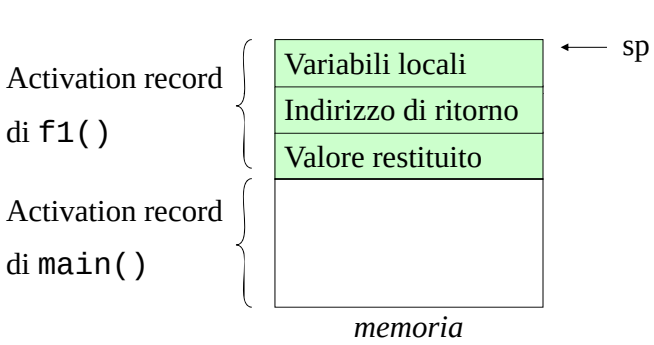
```
/*102*/    double Power(double x, int n){
/*103*/         if (n==0)
/*104*/              return 1;
/*105*/         return x*Power(x,n-1);
            }

             int main() {
                ...
/*136*/         y = Power(5.6,2);
               ...
             }
```


Invocazione di una funzione

- Cosa succede quando una funzione viene invocata?
- Il *sistema* deve *memorizzare* lo stato del chiamante, creare lo spazio per lo stato del chiamato e sapere dove riprendere l'esecuzione del programma dopo che il metodo è terminato.
- Lo stato della funzione/metodo, cioè la documentazione di attivazione (*activation record*), viene allocato sulla *pila* di esecuzione (*run-time stack*) e un puntatore (*stack pointer*) tiene traccia della posizione nella *pila*.

Invocazione di una funzione

Per esempio se il `main()` chiama la funzione `f1()`



Chiamata ricorsiva

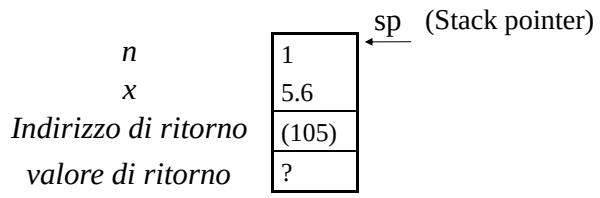
Una traccia delle chiamate ricorsive dell'esempio:

```
Invocazione 1      Power(5.6,2)
Invocazione 2           Power(5.6,1)
Invocazione 3                Power(5.6,0)
Invocazione 3                1
Invocazione 2           5.6
Invocazione 1      31.36
```

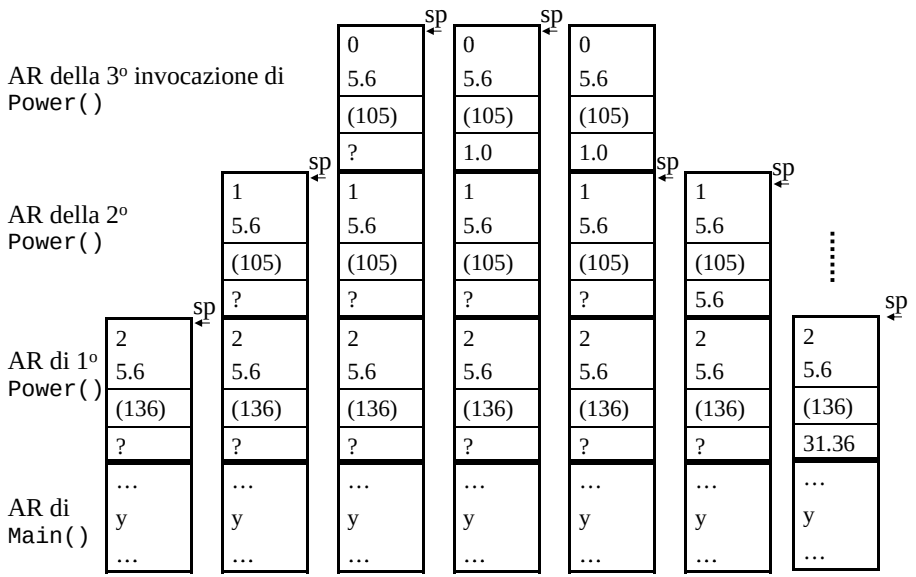
Il caso base deve essere sempre presente per interrompere la ricorsione.

Chiamata ricorsiva

- Vediamo una possibile rappresentazione grafica della pila di esecuzione dell'esempio precedente:
  - I numeri nei commenti (per es. /*136*/) rappresentano l'indirizzo dato dal sistema a quella "linea di codice";
  - Il ? indica il valore di ritorno non ancora calcolato;
  - Memoria locale: *n* e *x*



Chiamata ricorsiva



Ricorsione in coda

- L'esempio visto rappresenta un tipo di ricorsione chiamato *ricorsione in coda*.
- La ricorsione in coda è caratterizzata dall'uso di una sola invocazione ricorsiva al termine del metodo/funzione.
- Tale *ricorsione* può essere trasformata in *iterazione* attraverso l'uso di un ciclo.
- Quale vantaggio ad usare la ricorsione? La ricorsione sembra essere più intuitiva, perchè più simile alla definizione originale, e permette di scrivere codice coinciso.

Ricorsione in coda

- Tuttavia bisogna porre attenzione alle *risorse* che vengono utilizzate per produrre le chiamate ricorsive: lo spazio di memoria utilizzato e il tempo necessario ad attivare nuove chiamate.
- In generale la ricorsione in coda non è una caratteristica consigliabile.
- La potenza della ricorsione è legata a particolari strutture dati o ad algoritmi specifici.

Ricorsione non in coda

- Un esempio di tale ricorsione è la visualizzazione di una stringa di ingresso in ordine inverso:

```
int main(){
    cout<<"Inserisci una stringa: ";
    Reverse();
}
```

Una possibile uscita

```
Inserisci una stringa: 123456
654321
```

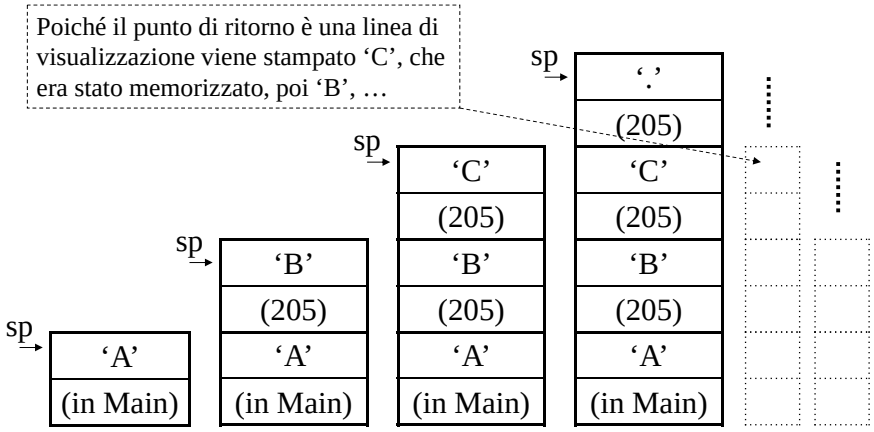
Una possibile uscita

```
Inserisci una stringa: qwert
trewq
```

```
/*200*/  void Reverse(){
/*201*/       char ch;
/*202*/       cin>>ch;
/*203*/       if (ch!='.'){
/*204*/            Reverse();
/*205*/            cout<<ch;}
            }
```


Ricorsione non in coda

Vediamo la pila di esecuzione con la stringa "ABC".



Ricorsione non in coda

- La trasformazione della ricorsione in iterazione di solito richiede la *gestione esplicita* di una *pila*. Vediamo un esempio che usa un *array* per memorizzare i caratteri:

```
void ReverseIter()
{
    char *stack = new char[80];
    int top = 0;
    cin>>stack[top] ;
    while (stack[top] != '.')
        cin>>stack[++top] ;
    for (top -= 1; top >= 0; top--)
        cout<<stack[top];
    delete [] stack;
}
```


Ricorsione eccessiva

- Se una funzione ricorsiva *ripete* il calcolo di alcuni parametri, il tempo di esecuzione può diventare elevato anche per casi molto semplici.
- Consideriamo i numeri di Fibonacci:

$$F(n) = \begin{cases} n & \text{se } n < 2 \\ F(n-2) + F(n-1) & \text{altrimenti} \end{cases}$$

- Una possibile implementazione ricorsiva è la seguente.

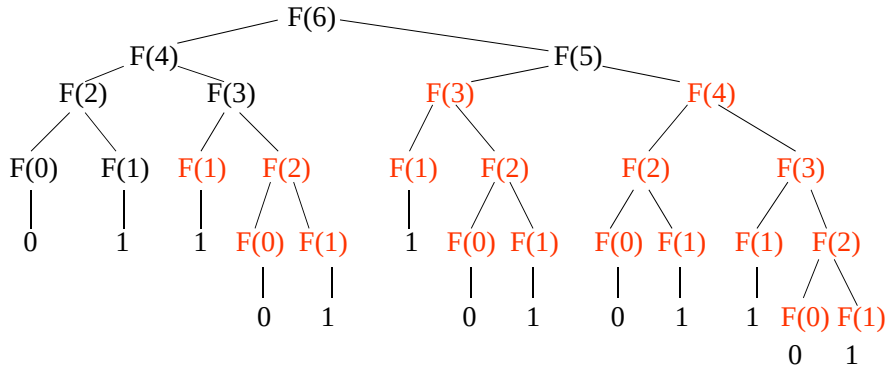
Ricorsione eccessiva

```
int Fib(int n){
    if (n<=0) return 0;
    if (n==1) return 1;
    return Fib(n-1)+Fib(n-2);
}
```

- Il metodo è semplice e di facile comprensione ma estremamente inefficiente. Per esempio, il calcolo di `Fib(31)` richiede 2178308 addizioni e 4356617 chiamate.

Ricorsione eccessiva

- L'origine di questa inefficienza risiede nella *ripetizione degli stessi calcoli*, come può essere evidenziato dall'albero delle invocazioni:



Esempio di ricorsione: ricerca binaria

- L'algoritmo di ricerca binaria può essere implementato in modo semplice ed efficiente usando la ricorsione: non ripete mai gli stessi calcoli su dati uguali.

Esempio di ricorsione: ricerca binaria

```
int BinarySearch(int * data, int key, int first, int last){
    if (first > last)
        return -1;
    else
    {
        int mid = (first + last) / 2;

        if (key==data[mid])
            return mid;
        else if (key<data[mid])
            return BinarySearch(data, key, first, mid - 1);
        else
            return BinarySearch(data, key, mid + 1, last);
    }
}
```