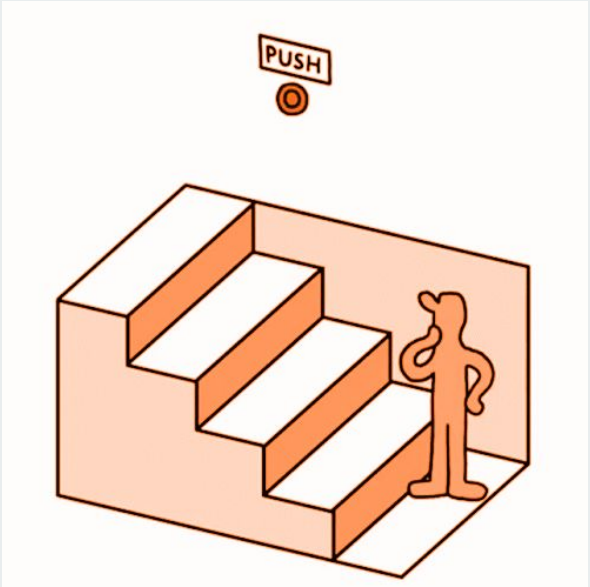
Funzioni ricorsive

Informatica A - 15/10/2024



Esempio 1 - Fattoriale

```c
// fig05_09.c
// Funzione fattoriale ricorsiva.
#include <stdio.h>

unsigned long long int  factorial( int  number);

int main(void) {
   // calcolo dei fattoriali e stampa del risultato
   for (int i = 0; i <= 21; ++i) {
      printf("%d! = %llu\n", i, factorial(i) );
   }
 }

 // definizione ricorsiva della funzione fattoriale
 unsigned long long int  factorial( int  number) {
    if  (number <=  1 ) {  // caso di base
       return   1 ;
    }
    else  {  // passo ricorsivo
       return  (number * factorial(number  - 1 ));
    }
 }
```

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
```

Esempio 2 - Fibonacci

```c
// fig05_10.c
// Funzione ricorsiva fibonacci.
#include <stdio.h>

unsigned long long int  fibonacci( int  n); // prototipo di funzione

int main(void) {
   // calcola e stampa fibonacci(number) per 0-10
   for (int number = 0; number <= 10; number++) {
      printf("Fibonacci(%d) = %llu\n", number, fibonacci(number) );
   }

    printf("Fibonacci(20) = %llu\n", fibonacci( 20 ) );
    printf("Fibonacci(30) = %llu\n", fibonacci( 30 ) );
    printf("Fibonacci(40) = %llu\n", fibonacci( 40 ) );
}

// Definizione ricorsiva della funzione fibonacci
unsigned long long int  fibonacci(int n) {
   if ( 0  == n ||  1  == n) {  // caso di base
      return  n;
   }
    else  {  // passo ricorsivo
      return  fibonacci(n -  1 ) + fibonacci(n -  2 );
   }
}
```

```
Fibonacci(0) = 0
Fibonacci(1) = 1
Fibonacci(2) = 1
Fibonacci(3) = 2
Fibonacci(4) = 3
Fibonacci(5) = 5
Fibonacci(6) = 8
Fibonacci(7) = 13
Fibonacci(8) = 21
Fibonacci(9) = 34
Fibonacci(10) = 55
Fibonacci(20) = 6765
Fibonacci(30) = 832040
Fibonacci(40) = 102334155
```

Esercizio 1

Scrivere tramite un sistema di funzioni ricorsive un programma stampi i primi n numeri primi.

```c
#include <stdio.h>
#include <stdbool.h>

bool isPrime(int num, int divisor)
{
    if (divisor == 1)
        return true;
    if (num % divisor == 0)
        return false;
    return isPrime(num, divisor - 1);
}

void printPrimeNumbers(int n, int current, int count)
{
    if (count == n)
        return;
    if (isPrime(current, current - 1))
    {
        printf("%d ", current);
        count++;
    }
    printPrimeNumbers(n, current + 1, count);
}
```

```c
int main()
{
    int n;
    printf("Enter the value of n: ");
    scanf("%d", &n);
    printf("First %d prime numbers are: ", n);
    printPrimeNumbers(n, 2, 0);
    return 0;
}
```

Esercizio 2

Scrivere una funzione ricorsiva in C che conti il numero di cifre di un numero in input.

```c
int noOfDigits(int n1)
{
    static int ctr=0;

     if(n1!=0)
     {
          ctr++;
         noOfDigits(n1/10);
    }

    return ctr;
}
```

Esercizio 3

Una piramide è fatta da diversi blocchi. Il primo piano più in cima ha un mattone, il secondo ne ha 2, il terzo ne ha tre e così via. Calcola ricorsivamente il numero totale di mattoni della piramide, dato il numero piani. (piramide(0) → 0, piramide(1) → 1, piramide(2) → 3)

```
int triangle(int rows) {
  if(rows==0){
    return 0;
  }
  else return rows + triangle(rows-1);
}

int triangle_tail(int rows){
    return triangle_helper(rows, 0);
}
```

```
int triangle_helper(int rows, int acc){
    if(rows == 0){
        return acc;
    } else return triangle_helper(rows-1, acc+rows);
}
```

Esercizio 4

Scrivere un programma C che, dato un numero N calcola la somma dei primi N numeri pari positivi in maniera ricorsiva.

- Specifica Liv 1: La somma dei primi N numeri pari è data dalla seguente,

  $S_N = 2*1 + 2*2 + 2*3 + \ldots + 2*i + \ldots + 2*(N-1) + 2*N.$
- Specifica Liv 2:
  - se N =1, $S_N = 2$, **(CASO BASE)**
  - se N >1, $S_N = 2 * N + S_{N-1}$ **(PASSO INDUTTIVO)**

    (somma dell'N-esimo numero pari + la sommatoria dei primi N-1 numeri pari.)

```
int somma_pari(int N) {
    if (N == 1)
    return 2;
    else
    return 2*N + somma_pari(N-1);
}
```

Esercizio 5

Scrivere una funzione C che prenda in ingresso un numero decimale n ed un numero decimale b e stampi a video la rappresentazione di n in base b. Scrivere anche un main per testare la funzione.

```
void d2b(int n,int b){
        if (n >= b){
                d2b(n/b,b);
                printf("%d",n%b);
        }
        else
                printf("%d",n);
}
```

Esercizio 6

Progettare e codificare un programma in C che calcoli il quoziente della divisione tra interi, utilizzando una funzione ricorsiva che prende in ingresso due interi x, y e restituisce il quoziente della divisione x/y.

Algoritmo ricorsivo per il calcolo del quoziente:

$$x/y = (x - y + y)/y = 1 + (x - y)/y.$$

```
#include <stdio.h>

int div(int x, int y){
    if (x>=y)
            return 1+div(x-y,y);
    else
            return 0;
}

main() {
    int x,y,ris;
    do{
        printf("Inserire dividendo: ");
        scanf("%d",&x);
    }while (x<0);
    do{
        printf("Inserire divisore: ");
        scanf("%d",&y);
    }while (y<1);
    ris = div(x,y);
    printf("%d/%d = %d",x,y,ris);
}
```

Esercizio 7

Scrivere un programma C che stampi sullo standard output tutti i valori del triangolo di Tartaglia per un certo ordine N, utilizzando una funzione ricorsiva.

```
1                          n = 0
1   1                      n = 1
1   2   1                  n = 2
1   3   3   1              n = 3
1   4   6   4   1          n = 4
1   5   10  10  5   1      n = 5
1   6   15  20  15  6   1  n = 6
.....
```

Leggendo la figura del triangolo di Tartaglia riga per riga, è possibile dedurre come il calcolo di ognuna di esse sia funzione della riga precedente. Il calcolo dei coefficienti binomiali segue dunque le seguenti regole:

con k == 0 e k == n, cobin(n,k) = 1. **(caso base)**

ogni coefficiente è la somma del suo "soprastante" e del predecessore di quest'ultimo. **(passo induttivo)**

```c
#include <stdio.h>
#define N 7

int cobin(int n, int k);

int main() {
        int n, k;

        for (n=0; n<=N; n++) {
                for (k=0; k<=n; k++)
                        printf(" %5d", cobin(n, k));
                printf("\n");
        }

        return 0;
}
```

```c
int cobin(int n, int k) {
        if (n<k || n<0 || k<0) {
                printf("Errore\n");
                return 0;
        }

        if (k==0 || k==n)
                return 1;
        else
                return cobin(n-1, k-1) + cobin(n-1, k);
}
```

Esercizio 8

Le funzioni pila(n) e torre(n) sono definite per n>0 come: $\mathbf{pila(n) = n^{(n-1)^{(n-2)^{\cdots}}}}$ $\mathbf{torre(n) = n^{n^{n^{\cdots}}}}$

(Esempi: pila(1) = 1, pila(2) = 2, pila(3)=3^2=9 pila(4)=4^9, torre(1) = 1, torre(2) = 4, torre(3)=3^27)
Si diano le opportune funzioni ricorsive di pila e torre, codificandole in C.

N.B.: l'associatività è dall'alto al basso: pila(4) = $4^{(3^2)} = 4^9$, torre(3) = $3^{(3^3)} = 3^{27}$ [e ***non*** $(4^3)^2$ *e* $(3^3)^3$]

Elevamento a Potenza: usiamo la funzione pot(a, b) che implementa il calcolo di a^b per a,b>=0

Diamone una definizione ricorsiva: caso base: se b = 0 → 1
pot(a,0) = 1

passo induttivo: $a^b = a*a^{b-1}$
pot(a,b) = b * pot(a, b-1)

Pila: caso base: pila(1) = 1
passo induttivo: $pila(n) = n^{pila(n-1)}$ = pot(n, pila(n-1))

Introduciamo poi la funzione tower(n, x) con n,x>0 tale che tower(n,1)=n, tower(n,2)= n^n, tower(n,3)= n^{n^n}...

Tower(n,x): eleva n a se stesso x-1 volte (cioè n compare x volte nella rappresentazione esponenziale)
caso base: tower(n,1) = n
passo induttivo: tower(n,x) = pot(n, tower(n,x-1))

Torre: caso base: torre(1) = 1
passo induttivo: torre(n) = tower(n,n) = $n^{tower(n,n-1)}$ = $n^{(n^{tower(n,n-2)})}$...

```
int pot( int b, int e ) {
 if( e > 0 )
  return b * pot(b, e-1);
 return 1;
}

int pila( int n ) {
 if( n == 1 )
  return 1;
 return pot(n, pila(n-1));
}
```

```
int tower( int n, int x ) {
 if( x == 1 )
  return n;
 return pot( n, tower( n, x-1 ) );
      // attenzione: x-1, NON n-1
}

int torre( int n ) {
 return tower( n, n );
}
```

Esercizio 9

Scrivere una funzione C che calcola sin(x) utilizzando lo sviluppo di Taylor fino al termine n-esimo. Realizzare utilizzando la ricorsione sia la funzione per il calcolo del fattoriale, sia la funzione che calcola sin(x). Fornire anche un main per testare la funzione.

$$sin(x) \approx \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159

int fatt(int n){
    if (n>1)
        return n*fatt(n-1);
    else
        return 1;
}

double sin_taylor(double x, int n){
    if (n>=0)
        return ( pow(-1,n)*pow(x,2*n+1)/fatt(2*n+1) + sin_taylor(x,n-1) );
    else
        return 0;
}
```

```
main(){
        double x,ris;
        int n;
        do{
                printf("Inserire angolo in radianti: ");
                scanf("%lf",&x);
        }while (x<0 || x>=2*PI);

        printf("Inserire grado del polinomio: ");
        scanf("%d",&n);
        ris = sin_taylor(x,n);
        printf("sin_taylor(%lf,%d) = %lf\n",x,n,ris);
        printf("sin(%lf) = %lf",x,sin(x));
}
```

Esercizio 10

Scrivere una funzione C che chieda in input all'utente una parola e ne restituisca la parola specchiata. (Esempio: 'abc'->'cba')

```
void stampaInv(){
   char a;

   scanf("%c",&a);
   if(a != '\n'){
      stampaInv();
      printf("%c",a);
   }
}
```