POLITECNICO
MILANO 1863

Lab 3:

Funzioni

Funzioni e procedure

Le funzioni sono *sottoprogrammi*.

- Procedure

  Eseguono operazioni senza ritornare un risultato alla funzione chiamante.

- Funzioni

  Ritornano un risultato al chiamante

  (Il tipo di ritorno deve essere indicato nella *signature*)

Funzioni e procedure

Procedura:

```
void StampaSomma(int n1, int n2)
{
    printf("%d", n1+n2);
}
```

Funzione:

```
int CalcolaSomma(int n1, int n2)
{
    return n1+n2;
}
```

La keyword `return` termina immediatamente l'esecuzione del sottoprogramma, ritornando il valore al chiamante.

Chiamata a funzione

Le funzioni hanno una *signature* che ne definisce parametri e tipo di ritorno.

```
void StampaSomma(int n1, int n2)
{
    printf("%d", n1+n2);
}
```

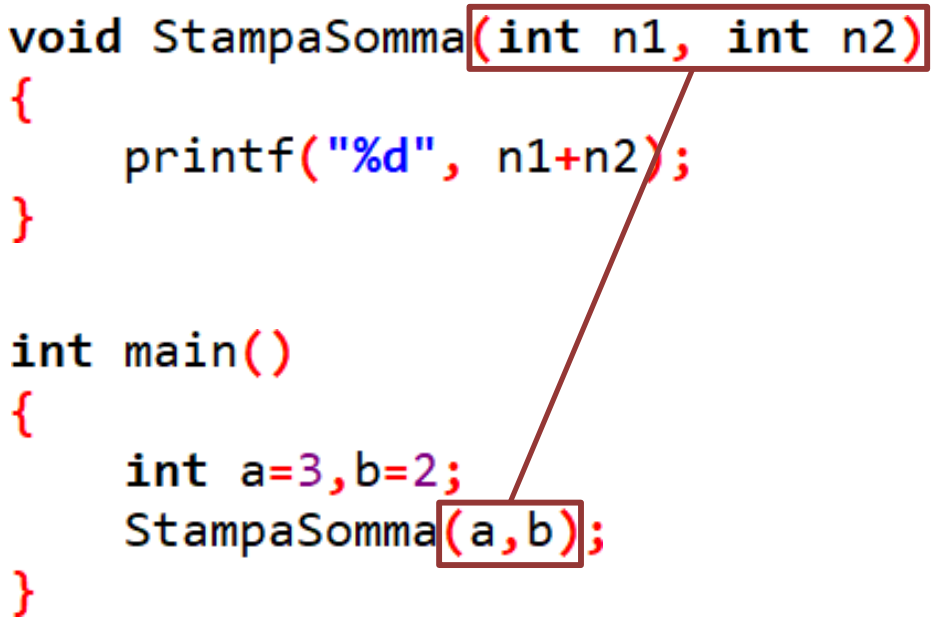
Per chiamare una funzione, scriviamo il suo nome e tra parentesi i parametri, separati da virgole.

```
int main()
{
    int a=3,b=2;
    StampaSomma(a,b);
}
```

Variabili locali alla funzione

I parametri nella signature definiscono variabili **locali** alla funzione.

Al momento della chiamata, i valori con cui è stata chiamata la funzione verranno copiati nelle variabili rappresentanti i parametri della funzione.



Variabili locali alla funzione

I parametri in ingresso sono già variabili della funzione, è **sbagliato** ridefinirle!

```
char Maiuscolo(char a)
{
    char a;
    if(a>96 && a<123)
    {
        return a-32;
    }
    return a;
}
```

Prototipi

Le chiamate a funzione devono essere precedute dalla definizione della funzione chiamata.

Se questo non è possibile, si può ricorrere ai prototipi che definiscono la signature delle funzioni prima della loro definizione.

```
char Maiuscolo(char a);

        :
        :
        :

Codice con chiamata
    a funzione

        :
        :
        :

char Maiuscolo(char a)
{
   ...
}
```

Esercizio 1

Scrivere un sottoprogramma che ritorna **1** se il numero in ingresso è primo, **0** altrimenti.

Esercizio 1 - Soluzione

```c
int isPrime(int n)
{
    if(n<2)
    {
        return 0;
    }

    int i;
    for(i=2;i<=(n/2);i++)
    {
        if(!(n%i))
        {
            return 0;
        }
    }
    return 1;
}

int main()
{
    int res = isPrime(11);
    printf(res ? "Primo" : "Non primo");

}
```

Esercizio 2

Scrivere un sottoprogramma che stampa la versione binaria di un numero.

CHALLENGE: Usare la rappresentazione binaria già in memoria.

Gli operatori `<<` e `>>` eseguono lo shift logico.

I bit del primo operando vengono spostati nella direzione dell'operatore di tanti posti quanto è il valore del secondo operatore.

Le variabili di tipo **`unsigned int`**, come dice il nome, sono variabili intere senza segno. Per queste, il primo bit non rappresenta il segno, è invece un'altra cifra.

Esercizio 2 - Soluzione

```
void bin(unsigned int n)
{
    unsigned int i, tmp, start=0;
    for(i=0;i<32;i++)
    {
        tmp = n << i;
        tmp = tmp >> 31;
        if(tmp)
        {
            start=1;
        }
        if(start)
        {
            printf("%d", tmp);
        }
    }
}

int main()
{
    int k=121;
    bin(k);
}
```

Esercizio 2 – Soluzione (senza shift)

```
void bin(int n)
{
    int ris=0, resto, pos=1;
    while(n)
    {
        resto = n%2;
        ris = ris + resto * pos;
        pos *= 10;
        n /= 2;
    }
    printf("%d", ris);
}
```

Indirizzi e puntatori

I puntatori sono variabili il cui valore rappresenta un indirizzo di memoria.

```
int* ptr1
```

Anche i puntatori in c sono *tipizzati*, dobbiamo quindi indicare di che tipo è la variabile puntata.

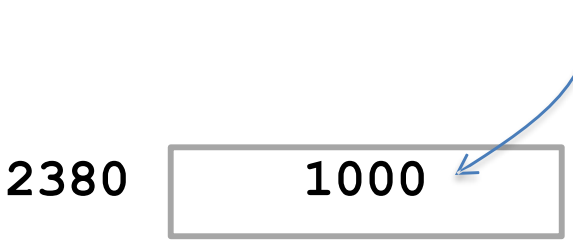
Referenziazione e dereferenziazione

Data una qualsiasi variabile, l'operatore `&` (di referenziazione) ritorna il puntatore a quella variabile.

Dato un puntatore, l'operatore `*` (di dereferenziazione) ritorna il valore della variabile contenuta all'indirizzo di memoria Salvato nel puntatore.

Referenziazione e dereferenziazione

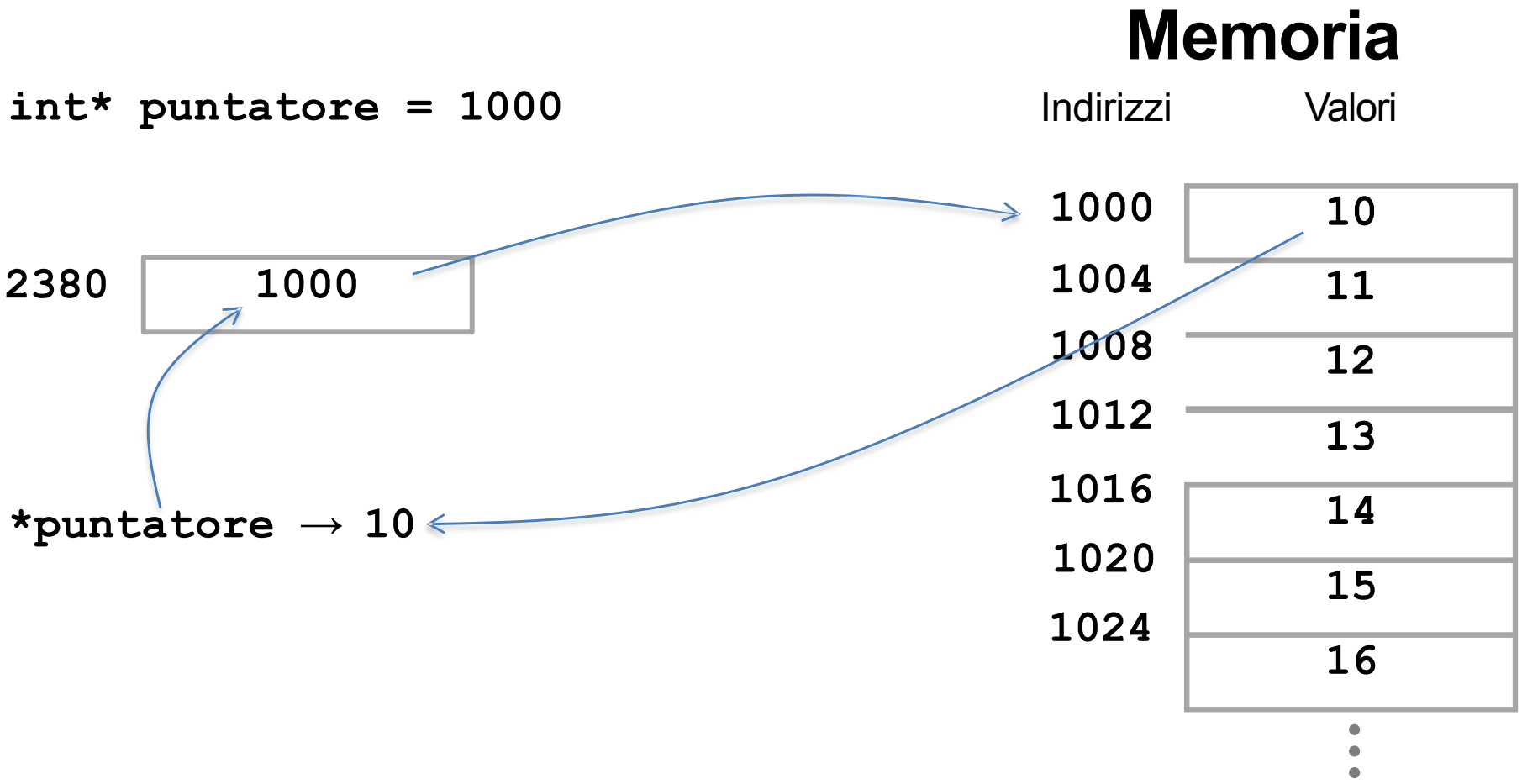
```
int* puntatore = 1000
```



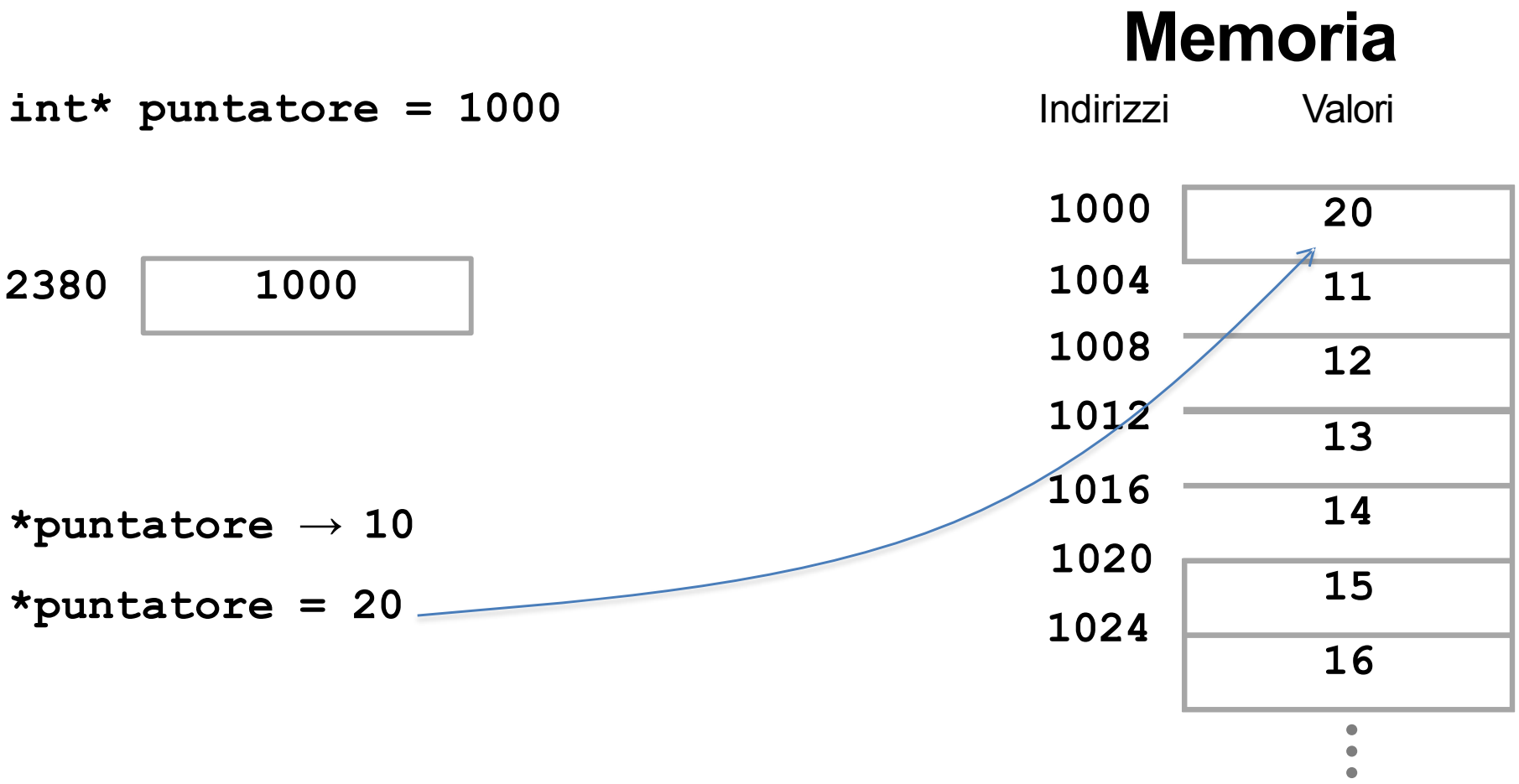
Memoria

Indirizzi	Valori
1000	10
1004	11
1008	12
1012	13
1016	14
1020	15
1024	16

Referenziazione e dereferenziazione



Referenziazione e dereferenziazione

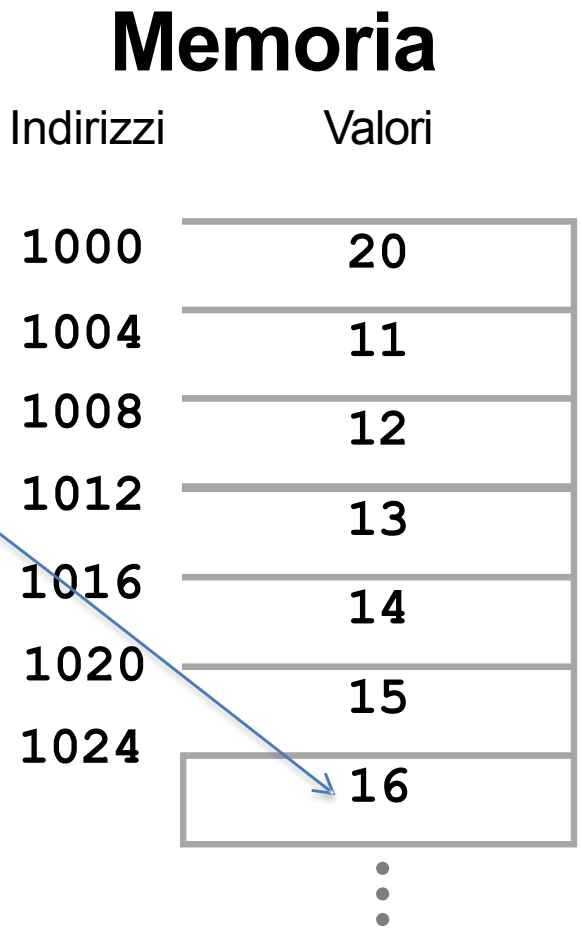


Referenziazione e dereferenziazione

```
int variabile = 16
int* puntatore = 1000
```

2380	1000

*puntatore → 10

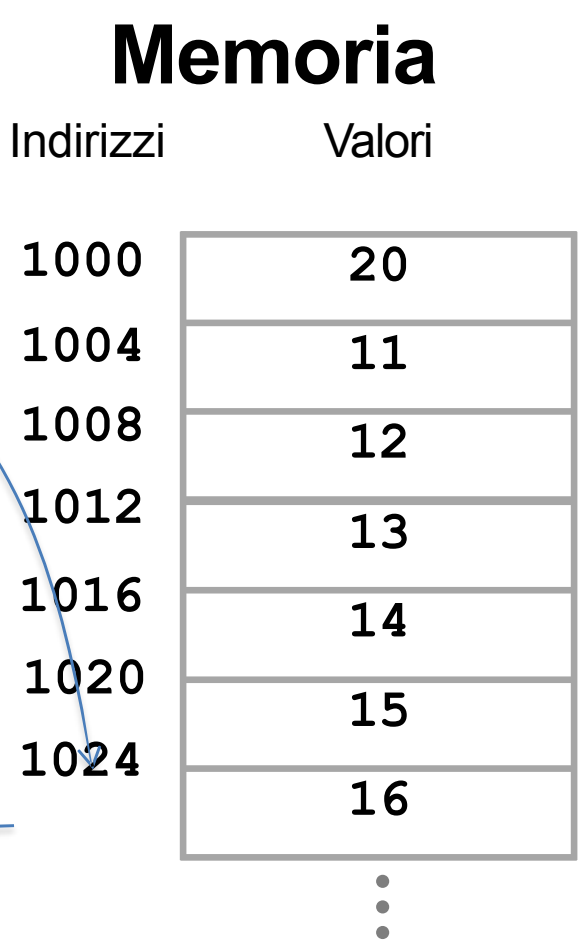


Referenziazione e dereferenziazione

```
int variabile = 16
int* puntatore = 1000
```

2380 | 1024

```
puntatore = &variabile
```



Passaggio di parametri

I valori passati come parametri sono copiati nelle variabili locali della funzione.

- Passaggio parametri per valore

  Come abbiamo visto finora, se il parametro è una variabile rappresentante un valore

- Passaggio parametri per indirizzo

  Se la variabile in ingresso al parametro rappresenta un indirizzo di memoria

Passaggio di parametri

Per valore:

```
int sommaVal(int n1, int n2)
{
    return n1+n2;
}
```

Per indirizzo:

```
int sommaRif(int* ptr1, int* ptr2, int* ptrres)
{
    *ptrres = *ptr1 + *ptr2;
}
```

Scanf usa passaggio per indirizzi

Abbiamo già usato funzioni che utilizzano il passaggio per indirizzo!

```
scanf("%d", &n);
```

La `scanf` richiede l'indirizzo della variabile in cui andare a inserire il valore.

Questo significa che una funzione può modificare valori non locali alla funzione!

Side effects

⚠ Se utilizziamo il passaggio per indirizzo, andiamo a modificare una cella di memoria, che non è locale alla funzione.

```
void funzione(int* puntatore)
{
    *puntatore = 4;
}

int main()
{
    int a = 2;
    funzione(&a);
    printf("%d", a);
}
```

Ritornare più valori

Questo però significa anche che possiamo avere più di un valore di ritorno se usiamo il passaggio per indirizzo.

```c
void operazioni(int a, int b, int* somma, int* prodotto)
{
    *somma = a+b;
    *prodotto = a*b;
}

int main()
{
    int a = 2;
    int b = 3;
    int ris_somma, ris_prodotto;

    operazioni(a, b, &ris_somma, &ris_prodotto);
    printf("La somma e': %d\n", ris_somma);
    printf("Il prodotto e': %d\n", ris_prodotto);
}
```

Esecizio 3

Scrivere un sottoprogramma che determina se due stringhe sono una l'anagramma dell'altra.

Esercizio 3 - Soluzione

```c
int anagrammi(char* s1, char* s2)
{
    int char1[256] = {0}, char2[256] = {0};
    int i;

    for(i=0;i<strlen(s1);i++)
    {
        char1[s1[i]]++;
    }
    for(i=0;i<strlen(s2);i++)
    {
        char2[s2[i]]++;
    }

    for(i=0;i<256;i++)
    {
        if(char1[i]!=char2[i])
        {
            return 0;
        }
    }
    return 1;
}
```

```c
int main()
{
    char s1[100], s2[100];
    printf("Inserisci stringa 1: ");
    scanf("%s", s1);
    printf("Inserisci stringa 2: ");
    scanf("%s", s2);

    int res = anagrammi(s1, s2);
    printf(res ? "Si" : "No");

}
```

Esercizio 4

Scrivere un sottoprogramma che ritorna l'n-esimo elemento di un array. (Parametri sono un array e n)

Esercizio 4 - Soluzione

```
int get(int* vettore, int posizione)
{
    return *(vettore+posizione);
}

int main()
{
    int a[10] = {0,1,2,3,4,5,6,7,8,9};
    printf("%d", get(a, 8));
}
```

Chiamate da altre funzioni

Le funzioni possono chiamare anche altre funzioni.

In questa maniera, si riesce a fattorizzare il codice ed evitare di ripetere inutilmente blocchi che eseguono operazioni comuni nel programma.

```
float supSfera(float raggio)
{
    return 4*PI*pow(raggio, 2);
}

float volSfera(float raggio)
{
    return (4./3.)*PI*pow(raggio, 3);
}

void infoSfera(float raggio)
{
    float sup = supSfera(raggio);
    float vol = volSfera(raggio);

    printf("Informazioni sfera );
    printf("Superficie: %f\n", sup);
    printf("Volume: %f", vol);
}

int main()
{
    infoSfera(1);
}
```

Chiamate da altre funzioni

Le chiamate vengono risolte in maniera LIFO (last-in-first-out). Quindi, se una funzione chiama un'altra funzione, quest'ultima sarà eseguita prima di continuare con la funzione chiamante.

```c
float supSfera(float raggio)
{
    return 4*PI*pow(raggio, 2);
}

float volSfera(float raggio)
{
    return (4./3.)*PI*pow(raggio, 3);
}

void infoSfera(float raggio)
{
    float sup = supSfera(raggio);
    float vol = volSfera(raggio);

    printf("Informazioni sfera );
    printf("Superficie: %f\n", sup);
    printf("Volume: %f", vol);
}

int main()
{
    infoSfera(1);
}
```

Ricorsione (cenni)

Le funzioni possono anche chiamare se stesse.

⚠ Le chiamate ricorsive sono utili, ma bisogna stare attenti poiché si potrebbe finire in un ciclo infinito di chiamate.

Ricorsione (cenni)

Cosa fa questo programma??

```
int sommaInteri(int n)
{
    if(n==0)
    {
        return 0;
    }
    else
    {
        return n + sommaInteri(n-1);
    }
}

int main()
{
    printf("%d", sommaInteri(4));
}
```

Ricorsione (cenni)

Nelle funzioni ricorsive, è importante avere un caso base, dove la funzione non effettua una chiamata ricorsiva

Ed un passo induttivo, per tutti gli altri casi, dove viene effettuata la chiamata ricorsiva.

Quando la funzione ricorsiva viene chiamata esternamente, il risultato finale sarà a valle di tutte le chiamate ricorsive.

```
int sommaInteri(int n)
{
    if(n==0)
    {
        return 0;
    }
    else
    {
        return n + sommaInteri(n-1);
    }
}

int main()
{
    printf("%d", sommaInteri(4));
}
```

Esercizio 5

Scrivere una funzione ricorsiva che calcola il fattoriale di un numero.

Esercizio 5 - Soluzione

```
int fact(int n)
{
    if(n<0)
    {
        return -1;
    }
    if(n==0)
    {
        return 1;
    }
    else
    {
        return n * fact(n-1);
    }
}
```

Esercizio 6

Scrivere una funzione ricorsiva che stampa la sequenza di Collatz a partire da un numero dato.

Il successivo di un numero n nella sequenza è n/2 se n è pari, e 3n+1 altrimenti. La sequenza termina con l'elemento di valore 1.

Es: 12, 6, 3, 10, 5, 16, 8, 4, 2, 1

Esercizio 6 - Soluzione

```
void collatz(int n)
{
    printf("%d\n", n);
    if(n==1)
    {
        return;
    }
    else if(n%2)
    {
        collatz(3*n+1);
    }
    else
    {
        collatz(n/2);
    }
}
```

max di un array (passato con */ [])

Dato un array di int, si scriva una f. Max con seguente prototipo:

Int maxArr(int *valori);

```
t maxArr(int (*valori) ){
```

Il "blue" è in int.

```
t maxArr(int          ){
```

max di un array (passato con */ [])

```c
//  maxArray
//  Created by ing.conti on 27/10/23.

#include <stdio.h>

#define DIM 4

// protos:
int maxArr(int *valori);
int maxArr2(int valori[]); // as array

int main(int argc, const char * argv[]) {

    int misure[DIM];

    // printf scanf.. leggo valori.
    misure[0] = 33;
    misure[1] = 44;
    misure[2] = 1;
    misure[3] = 2;

    int maxx = maxArr(misure);
    int maxx2 = maxArr2(misure);
    return 0;
}

int maxArr(int *valori){
    int i, max;
    max = *valori;

    for (i=1; i<DIM; i++) {
        valori++;
        if(*valori>max)
            max = *valori;
    }
    return max;
}

// come array:
int maxArr2(int valori[]){

    int i, max;
    max = valori[0];
    // but also max = valori[0]; is valid!
    for (i=1; i<DIM; i++) {
        if(valori[i]>max)
            max = valori[i];
    }
    return max;
}
```

strlen “a mano” in C

Proto:

```c
size_t strlen(const char *str);
```

Per noi:

```c
int mystrlen(char *str);
```

strlen “a mano” in C

```c
//  main.c
//  strlen
//
//  Created by ing.conti on 27/10/23.

#include <stdio.h>

int mystrlen(char *str);
int mystrlen2(char str[]);

#define DIM 256

int main(int argc, const char * argv[]) {
    char s1[DIM] = "ciao";

    int l = mystrlen(s1);
    int l2 = mystrlen2(s1);

    return 0;
}

int mystrlen(char *str){
    int len;
    len = 0;
    while(*str != '\0'){
        str++;
        len++;
    }
    return len;
}

int mystrlen2(char str[]){
    int i;
    i = 0;
    while (str[i]!='\0') {
        i++;
    }
    return i;
}
```

strcopy“a mano” in C

Proto:

```
char *strcpy(char *dest, const char *src);
```

Per noi:

```
 void mystrcpy(char *dest, char *src);
```

Variante con lunghezza:

```
 int mystrcpy(char *dest, char *src);
```

strcopy“a mano” in C

```c
//
//  main.c
//  strcpy
//
//  Created by ing.conti on 27/10/23.
//

#include <stdio.h>

#define  DIM 256

void mystrcpy(char *dest, char *src);
void mystrcpyQ(char dest[], char src[]);


int main(int argc, const char * argv[]) {
    char dest[DIM];
    char src[DIM];

    printf("dammi sorgente: ");
    scanf("%s", src);
    mystrcpy(dest, src);

    mystrcpyQ(dest, src);
    return 0;
}

void mystrcpy(char *dest, char *src){
    while (*src != '\0') {
        *dest = *src;
        src++;
        dest++;
    }
    // *dest = *src;
    // OR:
    *dest = '\0';
}

//(Segue)
```

strcopy“a mano” in C

```c
int mystrcpy2(char *dest, char *src){
    int cont = 0;
    while (*src != '\0') {
        *dest = *src;
        src++;
        dest++;
        cont++;
    }
    // *dest = *src;
    // OR:
    *dest = '\0';

    return  cont;
}


void mystrcpyQ(char dest[], char src[]){
    int i;
    i = 0;
    while (src[i] != '\0') {
        dest[i] = src[i];
        i++;
    }
    dest[i] = '\0';

}
```

Si definisca una struttura contenente coordinate geografiche (latitudine longitudine) e il nome della corrispondente località.

Es: Località geografiche: definizione

```c
//
//  main.c
//
//  Created by ing.conti on 20/10/23.

#define LEN_NOME 32
#define MAX_LOC 3

typedef struct Localita{
    char nome[LEN_NOME];
    float lat, lon; // rome :  41.902782, 12.496366.
    float temperatura;
}Localita;
```

Dati di esempio

```
Localita localita[MAX_LOC] = {

        {"rome",41.902782, 12.496366, 23 },
        {"milan",45.464664, 9.188540, 12 },
        {"naples",40.853294, 14.305573, 25 },
    };
```

f. che stampa array di campioni (passato con * / [])

Proto:..

```
void stampa(Localita citta[]);
void stampa1(Localita *citta);
void stampa2(Localita citta[]);
```

Un po' di varianti..

f. che stampa array di campioni (passato con * / [])

```c
//
//  main.c
//  maxtemp
//
//  Created by ing.conti on 27/10/23.

#include <stdio.h>
#define LEN_NOME 32
#define MAX_LOC 3

typedef struct Localita{
    char nome[LEN_NOME];
    float lat, lon; // rome :  41.902782, 12.496366.
    float temperatura;
}Localita;

void stampa(Localita citta[]);
void stampa1(Localita *citta);
void stampa2(Localita citta[]);

void stampacitta(Localita citta);

int main(int argc, const char * argv[]) {

    Localita localita[MAX_LOC] = {
            {"rome",41.902782, 12.496366, 23 },
            {"milan",45.464664, 9.188540, 12 },
            {"naples",40.853294, 14.305573, 25 },
        };

    stampa(localita);
    stampa1(localita);
    stampa2(localita);
    return 0;
}
(Segue)
```

f. che stampa array di campioni (passato con * / [])

```c
void stampa(Localita citta[]){
    int i;
    Localita qui;
    for (i=0; i<MAX_LOC; i++) {
        qui = citta[i];
        printf("%s ", qui.nome);
        printf("%f ", qui.lat);
        printf("%f ", qui.lon);
        printf("%f ", qui.temperatura);
        printf("\n");
    }
}

void stampa1(Localita *citta){
    int i;
    Localita qui;
    for (i=0; i<MAX_LOC; i++) {
        qui = *citta;
        printf("%s ", qui.nome);
        printf("%f ", qui.lat);
        printf("%f ", qui.lon);
        printf("%f ", qui.temperatura);
        printf("\n");
        citta++;
    }
}

void stampa2(Localita citta[]){
    int i;
    for (i=0; i<MAX_LOC; i++) {
        stampacitta(citta[i]);
    }
}


void stampacitta(Localita citta){
    printf("%s ", citta.nome);
    printf("%f ", citta.lat);
    printf("%f ", citta.lon);
    printf("%f ", citta.temperatura);
    printf("\n");
}
```

f. Che trova la città con la temperatura piu alta del 20% della media

Come prima:

```
#define LEN_NOME 32
#define MAX_LOC 3

typedef struct Localita{
    char nome[LEN_NOME];
    float lat, lon; // rome :  41.902782, 12.496366.
    float temperatura;
}Localita;
```

Es su ricorsione

1) Dato un array di int, si scriva una f. Max RICORSIVA con seguente prototipo:

Int maxArr(int *valori, int n);

2) scriva una f. Max RICORSIVA simile alla strelen con seguente prototipo:

int mystrlen(char *str);