

## Esercizio 1

### Somma di numeri complessi

- Realizzare una funzione che riceva in ingresso due numeri complessi
  - Un numero complesso è dato da una coppia rappresentante la parte reale e la parte immaginaria
- La funzione deve restituire la somma di tali valori (ancora una coppia)
- Realizzare anche `main()` di esempio

1

## Esercizio 1 - Soluzione

```
void sommaComplex(
    float reA, float imA,
    float reB, float imB,
    float * reResult, float * imResult)
{
    *reResult = reA + reB;
    *imResult = imA + imB;
    return;
}

int main() {
    float reResult, imResult;
    sommaComplex(1.0, 1.0, 2.0, 2.0, &reResult, &imResult);
    printf("Parte reale: %f\n
           Parte Immaginaria: %f\n", reResult, imResult);
}
```

2

## Esercizio 2

### Area e perimetro di un triangolo

- Realizzare una funzione che, date le lunghezze dei tre lati di un triangolo
  - Restituisca uno fra tre codici
    - PRIMO CASO: triangolo non valido
      - Un triangolo è invalido se uno dei tre lati è più lungo della somma degli altri due, oppure se uno dei tre lati è negativo
    - SECONDO CASO: triangolo degenere
      - Un triangolo è degenere se uno dei tre lati è nullo, oppure uno dei tre lati è uguale alla somma degli altri due
    - TERZO CASO: triangolo valido
  - Nel caso di triangolo valido, la funzione deve anche restituire il perimetro e l'area del triangolo
    - Per l'area, si utilizzi la formula (con  $s$  semiperimetro del triangolo)

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

- Realizzare una procedura per la gestione del risultato

3

## Esercizio 2 - Soluzione (1)

```
#define CALCULATE_RET_TYPE int
#define REGULAR_TRIANGLE 0
#define INVALID_TRIANGLE 1
#define LIMIT_TRIANGLE 2
CALCULATE_RET_TYPE calculate(float a, float b, float c, float* area,
                             float* perimeter)
{
    float s;
    if(a < 0 || b < 0 || c < 0 || a > b+c || b > a+c || c > a+b)
        return INVALID_TRIANGLE;
    if(a == 0 || b == 0 || c == 0 || a == b + c || b == a + c
        || c == a + b)
        return LIMIT_TRIANGLE;
    *perimeter = a + b + c;
    s = *perimeter / 2;
    *area = sqrt( s * (s-a) * (s-b) * (s-c));
    return REGULAR_TRIANGLE;
}
```

4

## Esercizio 2 - Soluzione (2)

```
void printCalculation(float a, float b, float c)
{
    float area, perimeter;
    CALCULATE_RET_TYPE result;
    result = calculate(a, b, c, &area, &perimeter);
    switch(result)
    {
        case INVALID_TRIANGLE:
            printf("Triangolo non valido\n");
            break;
        case LIMIT_TRIANGLE:
            printf("Triangolo limite\n");
            break;
        case REGULAR_TRIANGLE:
            printf("Perimetro: %f, Area: %f\n", perimeter, area);
    }
}
```

5

## Esercizio 3 vettori

### Esempio - Test di uguaglianza fra vettori, elemento per elemento

- Creare una funzione che, dati in input due vettori e le rispettive lunghezze, determina se i due vettori sono uguali
- IPOTESI: l'uguaglianza va testata in maniera "ordinata", **ovvero elemento per elemento**

6

## Esercizio 3 - Soluzione (1)

```
#define RESULT int
#define DIFFERENT_LENGTH -1
#define EQUAL 0
#define DIFFERENT 1
RESULT compareTol(TYPE v1[], TYPE v2[], int dim1, int dim2)
{
    int i;
    if(dim1 != dim2)
        return DIFFERENT_LENGTH;
    for(i = 0; i < dim1; i++)
    {
        if( !equals(v1[i], v2[i]) )
            return DIFFERENT;
    }
    return EQUAL;
}
```

7

## Esercizio 3 - Soluzione (2)

- L'uguaglianza fra elementi deve ovviamente conoscere il loro tipo...
- Es: per vettori di interi

```
#define TYPE int
#define BOOLEAN int
#define TRUE 1
#define FALSE 0

BOOLEAN equals(TYPE e1, TYPE e2)
{
    return(e1 == e2); //uguaglianza dipendente dal tipo
}
```

8

## Esercizio 4

### Test di uguaglianza fra vettori con elementi non ripetuti

- Questa volta il test deve verificare che i vettori contengano gli stessi elementi, **NON NECESSARIAMENTE NELLO STESSO ORDINE**
- Ipotesi semplificativa: ***i vettori non hanno elementi ripetuti***

9

## Esercizio 4 - Soluzione

```
RESULT compareTo2(TYPE v1[], TYPE v2[], int dim1, int dim2)
{
    int i, j;
    BOOLEAN currentEquality = TRUE;
    if(dim1 != dim2)
        return DIFFERENT_LENGTH;
    for(i = 0; i < dim1 && currentEquality; i++)
    {
        contains(v2, v1[i])
        {
            currentEquality = FALSE;
            for(j = 0; j < dim2 && !currentEquality; j++)
            {
                currentEquality = equals(v1[i], v2[j]);
            }
            if(!currentEquality)
                return DIFFERENT;
        }
    }
    if(currentEquality)
        return EQUAL;
    else
        return DIFFERENT;
}
```

10

## Esercizio 5

### Test di uguaglianza fra vettori con elementi ripetuti

- Ora rimuoviamo anche l'ipotesi sulla possibilità di avere elementi ripetuti
- Che cosa dobbiamo modificare della precedente funzione?
  - A volte, **NON SEMPRE**, è utile partire da ciò che abbiamo già realizzato

11

## Esercizio 5 - Schema di soluzione

Per risolvere il problema della corrispondenza uno a uno

- Definiamo un **vettore di booleani**, con dimensione logica pari a quella dei vettori in esame
- Inizializziamo ogni elemento del vettore a FALSE
- L'elemento *i*-mo di tale vettore indica se l'elemento del secondo vettore in esame **è già stato utilizzato** per un confronto di successo
- Quindi il test di appartenenza di un elemento del primo vettore nel secondo deve considerare unicamente gli **elementi non ancora utilizzati per confronti di successo**
  - Ovvero quelli per cui il corrispondente valore booleano è ancora FALSE

12

## Esercizio 5 - Soluzione (1)

```
#define MAX_DIM 50
RESULT compareTo3(TYPE v1[], TYPE v2[], int dim1, int dim2)
{
    BOOLEAN checked[MAX_DIM];
    int i, j;
    BOOLEAN currentEquality = TRUE;
    if(dim1 != dim2)
        return DIFFERENT_LENGTH;
    for(i = 0; i < dim1; i++) //uso la dimensione logica
        checked[i] = FALSE;
    ...
}
```

13

## Esercizio 5 - Soluzione (2)

```
for(i = 0; i < dim1 && currentEquality; i++)
{
    currentEquality = FALSE;
    for(j = 0; j < dim2 && !currentEquality; j++)
    {
        if(!checked[j])
        {
            currentEquality = equals(v1[i], v2[j]);
            if(currentEquality)
            {
                checked[j] = TRUE;
            }
        }
    }
    if(currentEquality)
        return EQUAL;
    else
        return DIFFERENT;
}
```

Usa solo gli elementi non ancora utilizzati per un confronto di successo

v2[j] è stato utilizzato per un confronto di successo  
→ Devo aggiornare il vettore di booleani

14

## Esercizio 5 - Soluzione (3)

### Esempio

- Al test sull'ultimo elemento del primo vettore (valore 1) ...

1	1	3	5	1
3	5	1	5	1
TRUE	TRUE	TRUE	FALSE	TRUE

- L'unico elemento del secondo vettore non ancora utilizzato per una corrispondenza è 5
- 1 viene confrontato con 5 e non viene trovata nessuna corrispondenza per esso

15