UNIVERSITY OF TRENTO - Italy

Esercitazione 13

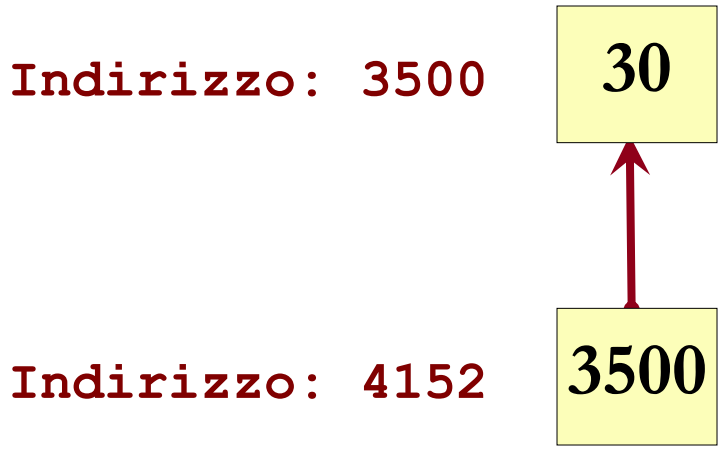
DISI – aa 2024/2025

Pierluigi Roberti
Carmelo Ferrante

Università degli Studi di Trento
pierluigi.roberti@unitn.it

Indirizzo di una Variabile

```
int x=30;
```




```
X=30
&x=3500
```

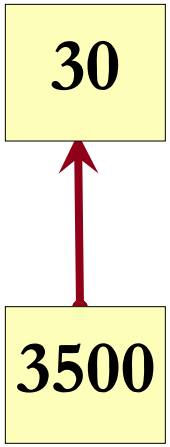
- Si può memorizzare in una variabile l'indirizzo della variabile x, in questo caso è necessario dichiarare una variabile di tipo puntatore

Variabile di tipo puntatore

```
int x;
x=30;
int *px;
px = &x;
```

x Indirizzo:3500

px Indirizzo:4152



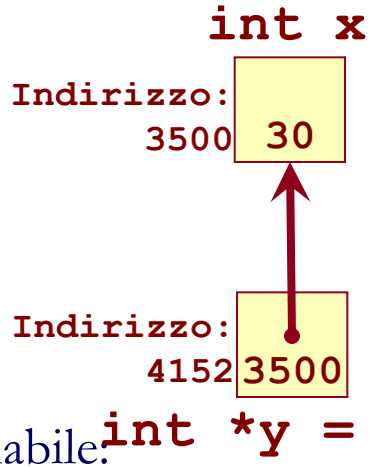
x ➔ 30 px ➔ 3500

&x ➔ 3500 *px ➔ 30

- Concetto di indirizzo (var) &
- Operatore di deferenziazione *
- Dichiarazione puntatore, inizializzazione

```
int *a;
int b=4;
a=&b;
printf("%d %d %d %d",*a,b, a, &b);
```

Passaggio parametri per indirizzo



1. Parametro formale definito di tipo ***puntatore*** al tipo del parametro attuale
   - Un puntatore non contiene i **dati** di una variabile: contiene l'indirizzo della cella di memoria dove i dati sono contenuti
   - Il tipo "puntatore a T" (dove T è un qualsiasi tipo) si indica con **`T*`**
   - Es: **`fun(int *p)`**
2. All'atto dell'invocazione, il parametro attuale viene ***passato per indirizzo*** usando **`&`**
   - Es: **`fun(&x)`**
   - **`&x`** contiene l'**indirizzo** della variabile **`x`** (e non il **contenuto** di **`x`**)
   - L'argomento di **`&`** deve essere una variabile, non una generica espressione
   - Ricordate la sintassi di **`scanf`**?
3. Nel corpo della funzione viene usato l'***operatore di de-referenziazione*** **`*`** per accedere al contenuto della variabile (e quindi al parametro attuale vero e proprio)
   - Es: **`*p=5`** opera sul contenuto della cella puntata dal valore di **`p`**

Esempio 1

```
int stampa(int a){
  a=a+1;
  printf("(stampa) %d\n",a);
}

int stampa1(int *a){
  *a=*a+1;
  printf("(stampa1) %d\n",*a);
}

int stampa2(int *b){
  *b=*b+1;
  printf("(stampa2) %d\n",*b);
}
```

```
int main() {
 int a=4;
 printf("(main) a %d\n",a);

 stampa(a);
 printf("(main) a %d\n",a);

 stampa1(&a);
 printf("(main) a %d\n",a);

stampa2(&a);
 printf("(main) a %d\n",a);
}
```

OUTPUT

```
(main) a 4
(stampa) 5
(main) a 4
(stampa1) 5
(main) a 5
(stampa2) 6
(main) a 6
```

Esempio 2

```
void somma_uno(int x) {
    x += 1;
    printf("\nValore var x: %d\n", x);
}
void aggiungi_uno(int * x) {
    *x += 1;
    printf("\nValore var x: %d\n", *x);
}
int main() {
 int a = 12;
 printf("var a - Val: %d - Ind: %d (0x%0x)\n", a, &a, &a);
 somma_uno(a);
 printf("var a - Val: %d - Ind: %d (0x%0x)\n", a, &a, &a);
 aggiungi_uno(&a);
 printf("var a - Val: %d - Ind: %d (0x%0x)\n", a, &a, &a);
}
```

Output

```
var a - Valore: 8 - Indirizzo: 1606415932 (0x5fbff63c)
somma_uno Valore var x: 9
var a - Valore: 8 - Indirizzo: 1606415932 (0x5fbff63c)
aggiungi_uno Valore var x: 9
var a - Valore: 9 - Indirizzo: 1606415932 (0x5fbff63c)
```

Esempio 3 – swap: scambio dei valori di due interi

```
void swap (int *x, int *y) {
  int temp;
  temp = *x;
  *x = *y;
  *y = temp;
}
int main() {
 int a=10, b=20;
 printf("a: val %d - ind %d (0x%0x)\n", a, &a, &a);
 printf("b: val %d - ind %d (0x%0x)\n", b, &b, &b);
 swap(&a, &b);
 printf("a: val %d - ind %d (0x%0x)\n", a, &a, &a);
 printf("b: val %d - ind %d (0x%0x)\n", b, &b, &b);
}
```

Effetto collaterale: il valore dei parametri cambia in seguito all'invocazione della funzione

Ciò non accade con le funzioni matematiche…

```
OUTPUT
a: val 10, ind: 1606415932 (0x5fbff63c)
b: val 20, ind: 1606415928 (0x5fbff638)
a: val 20, ind: 1606415932 (0x5fbff63c)
b: val 10, ind: 1606415928 (0x5fbff638)
```

struct come parametri

- Si possono passare sia per valore che per indirizzo…
- … ed essere usate come risultato di una funzione
- Se la struct è passata per valore, l'intero contenuto viene copiato
  - anche eventuali campi array (statici), che vengono copiati interamente

struct e puntatori: sintassi

- In C/C++, l'operatore "**.**" che consente di accedere ai campi di una struct ha precedenza sull'operatore di de-referenziazione "*"
- Per accedere a un campo di una struct, sono dunque necessarie le parentesi : Es. **`(*v).actualSize;`**
- Il C fornisce anche una sintassi alternativa, più compatta

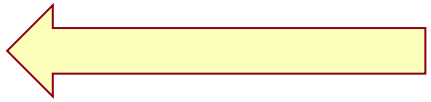
Le due notazioni sono equivalenti

```
insert(Vector *v, int el) {
  if((*v).actualSize < SIZE) {
    (*v).contents[(*v).actualSize] = el;
    (*v).actualSize++;
  }
}
```

```
insert(Vector *v, int el) {
  if(v->actualSize < SIZE) {
    v->contents[v->actualSize] = el;
    v->actualSize++;
  }
}
```

Esempio 4 – struct

```
#define SIZE 100
//struttura non anonima
typedef struct Tcollezione{
    int contents[SIZE];
    int actualSize;
}Tcollezione;
//Definizione della procedura insert
void insert(Tcollezione *v, int el) {
  if( (*v).actualSize < SIZE ) {
    (*v).contents[actualSize] = el;
    (*v).actualSize++;
    //oppure
    //v->actualSize++;
  }
}
int main() {
 Tcollezione vector;
 int elem = 27;
 //invocazione
 insert(&vector, elem);
}
```



NB!

Esempi di invocazione

```
typedef struct Data{
  int x,y,z;
} Data;

typedef struct Tcoll{
  Data d[N];
  int n_elem;
} Tcoll;

//dichiarazione header funzioni
void f1(int a, int b);
char f2(int a, int b);
void f3(int *a, int b);
void f4(int *a, int *b);
void f5(Tcoll c);
void f6(Tcoll *c);
void f7(const Tcoll* c);
Tcoll f8(Tcoll c);
void f9(int v[N], int d);
```

```
int main() {
 int a, b;
 char k;
 Tcoll myColl, var;
 int vet[N];
 int *p;
 Tcoll *pc;
 //Esempi di invocazione
 f1(a, b);
 f1(a, 5);
 k = f2(b, 18);
 f3(&a, b);
 f3(p, 122);
 f4(&a, p);
 f5(myColl);
 f5(*pc);
 f6(&myColl);
 f6(pc);
 f7(&myColl);
 var = f8(myColl);
 f9(vet, N-2);
 f9(&vet[0], N/2);
}
```

Procedure e risultati

- Il passaggio parametri per indirizzo consente di ritornare uno o più risultati al chiamante
- È facile trasformare una funzione in procedura, rappresentando il risultato come un parametro aggiuntivo, passato per indirizzo
- Ad esempio, la funzione …

```
int f(int p1) {
   ...
   return risultato;
}
...
y = f(x);
```

- … può essere riscritta come procedura

```
void f(int p1, int *p2) {
   ...
   *p2 = risultato;
}
...
f(x,&y);
```

Algebra puntatori

```
int a=12;
int* pa = NULL;
pa = &a;
//oppure
int* pa = &a;
```

- Incremento
  - **pa++**
  - **pa +=1**
- Decremento
  - **pa--**
  - **pa -= 1**
- Incremento / Decremento di numero di byte pari alla dimensione del **tipo** del puntatore

Esercizio 1 - parte 1

#define MAX 10

- Definire una struct Tpunto

```
typedef struct Tpunto{
   float x, y, z;
} Tpunto;
```

- Definire una struct Tgrafico

```
typedef struct Tgrafico{
   Tpunto punti[MAX]; int dim;
} Tgrafico;
```

- Dichiarare variabile **g** di tipo **Tgrafico**
- Inizializzare g (funzione `init` o `p_init`)
  - Chiedere all'utente la dimensione dim
  - Valori casuali di x, y e z compresi tra -10.00 e 10.00 (funzione `init_punto` o `p_init_punto`)
- Stampare il contenuto di **g** (funzione `stampa` o `p_stampa`)
- Collezionare tutti i punti che sono entro una certa distanza (limite) dall'origine e poi stamparli a video (funzione `filtra` o `p_filtra`)

Esercizio 1 - parte 2

Versione 1: uso dei parametri – passaggio per valore

```
Tgrafico init();
Tpunto init_punto();
void stampa(Tgrafico a);
Tgrafico filtra(Tgrafico a, float limite);
```

Versione 2: uso dei puntatori – passaggio per riferimento

```
void p_init(Tgrafico *a);
void p_init_punto(Tpunto *p);
void p_stampa(const Tgrafico *a);
Tgrafico p_filtra(const Tgrafico *a, float limite);
```

Esercizio 1 - parte 2 - main

Versione 1: uso dei parametri – passaggio per valore

```
Tgrafico g;
g = init();
stampa(g);
Tgrafico w;
w = filtra(g, 12.25);
stampa(w);
```

Versione 2: uso dei puntatori – passaggio per riferimento

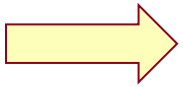
```
Tgrafico g;
p_init(&g);
p_stampa(&g);
Tgrafico w;
w = p_filtra(&g, 12.25);
p_stampa(&w);
```

Esercizio 1 - parte 3

Separare il codice scritto in Esercizio 1 in più file

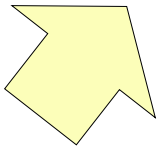
- main.c
- punti.c
- punti.h

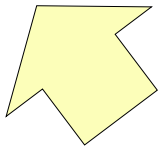
Riga vuota!!!



punti.h

```
#ifndef __PUNTI_H__
#define __PUNTI_H__
// #define …
// typedef …
// I prototipi delle funzioni
#endif
```





main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "punti.h"

int main() {
…
}
```

punti.c

```
#include <stdio.h>
#include <stdlib.h>
#include "punti.h"

//implementazione delle
funzioni
```

Sintassi direttive al compilatore

Sintassi utilizzata dal preprocessore

Serve per includere o escludere alcune parti di codice, nel nostro caso per evitare la multipla inclusione di librerie con duplicazione di variabili, tipi di dato e funzioni che causerebbe un errore in compilazione

#ifndef identificatore

- Se **non** è definito l'identificatore allora si prosegue ad analizzare le righe sucessive
- Se l'identificatore è stato definito si ignorano le righe successive fino ad #endif

#define identificatore

- Definisce l'identificatore

#endif

- Termine del blocco relativo alla compilazione condizionata
- NOTA: aggiungere una riga vuota dopo #endif

identificatore (formato standard nel caso di un file denominato dato.h):

__DATO_H__

_	_	D	A	T	O	_	H	_	_

Sintassi completa

#if espressione

Condizione verificata se espressione diversa da zero

#ifdef identificatore

Condizione verificata se identificatore definito

#ifndef identificatore

Condizione verificata se identificatore **non** definito

#else

Porzione considerata se espressione precedente non verificata

#elif espressione

Sintassi analoga a else + if

#endif

Termina il blocco relativo alla precedente condizione

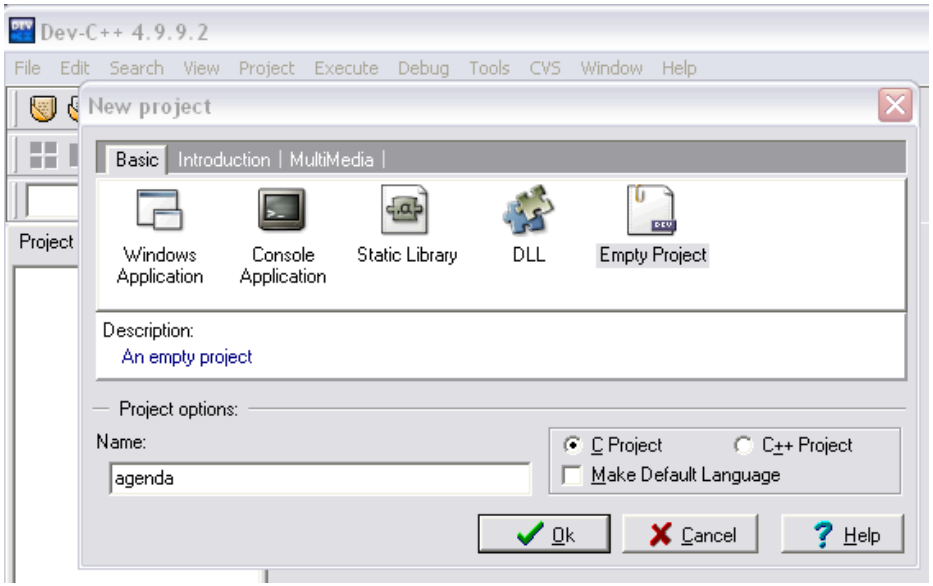
Un agenda in più file

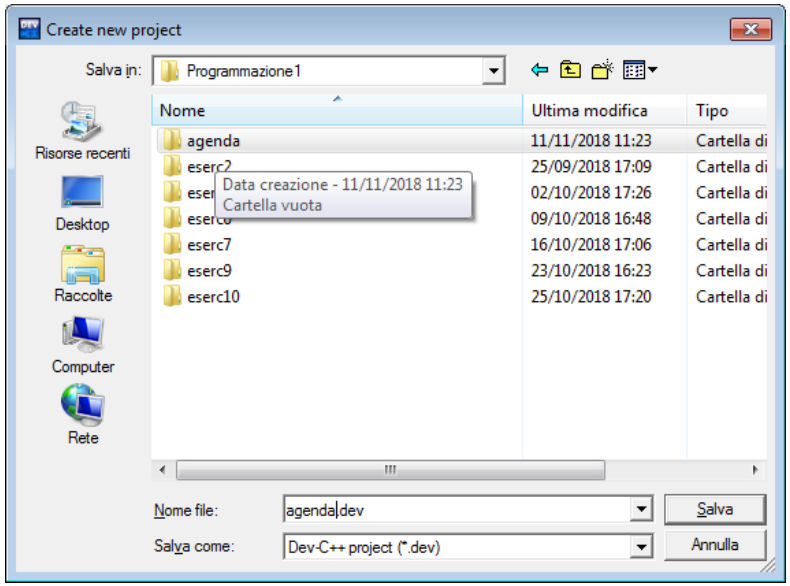
- Quando un programma diventa complesso (come nel caso dell'agenda) conviene suddividerlo in più file (anche perche così è possibile riutilizzare parte del codice per altri programmi).
- La cosa più semplice dal punto di vista organizzativo è spostare tutte le funzioni relative all'agenda in un file (agenda.c) e tenere la funzione main in un altro file (main.c).
- Tutte le funzioni devono però conoscere i **typedef**, **#define** e i **prototipi delle funzioni** che abbiamo scritto. Per evitare di scrivere sia in main.c che in agenda.c spostiamo tutte queste cose in un altro file che chiameremo agenda.h (*header file*).
- Questo file sarà "incluso" in agenda.c e in main.c con

  #include "agenda.h" // notare le virgolette
- Per semplicità è meglio che tutti questi file stiano nella stessa cartella.

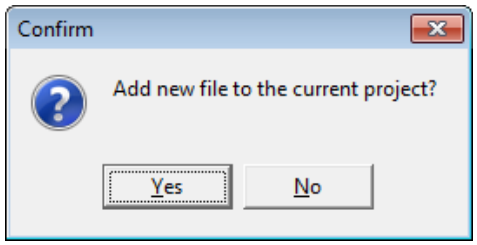
Esercizio 2

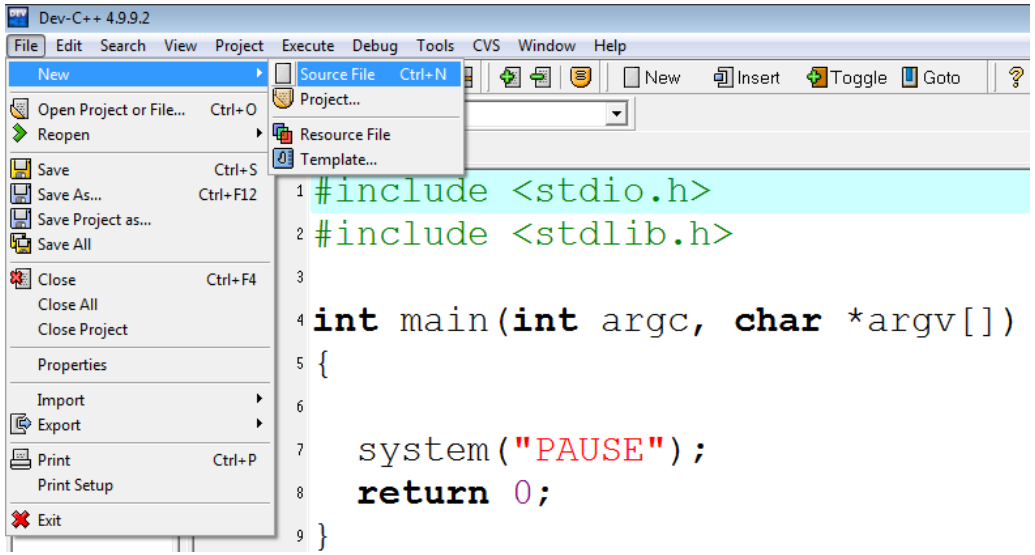
- Creare una nuova cartella, che chiameremo “agenda”.
- Recuperare il file E13.0_agenda_inizio.c e salvarlo in questa cartella.
- Scopo dell’esercizio è suddividerne il contenuto correttamente in tre file in modo da creare quello che Dev-C++ chiama un “progetto”.
- Dal menu di Dev-C++ selezioniamo File->New->Project.
- selezioniamo l’icona “Console Application” e il “radio button” “C project”. Poi clicchiamo su OK.
- Selezioniamo la cartella «agenda» e diamo «agenda» come nome del progetto

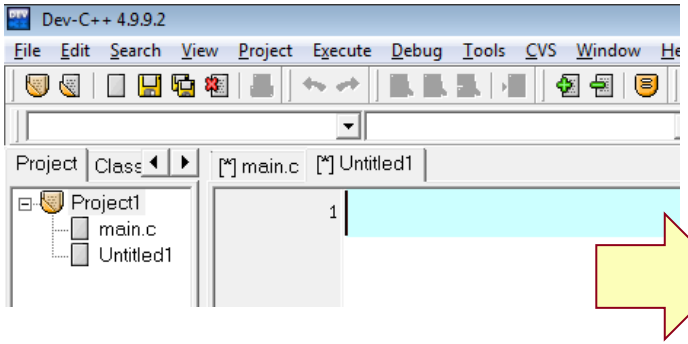


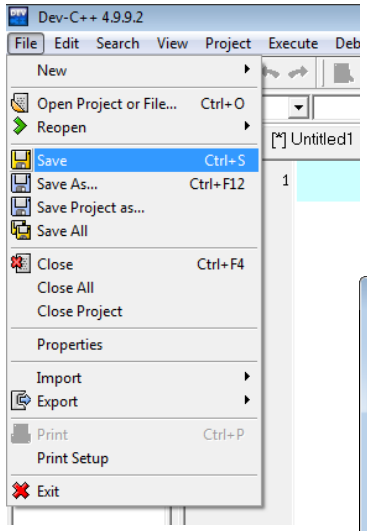


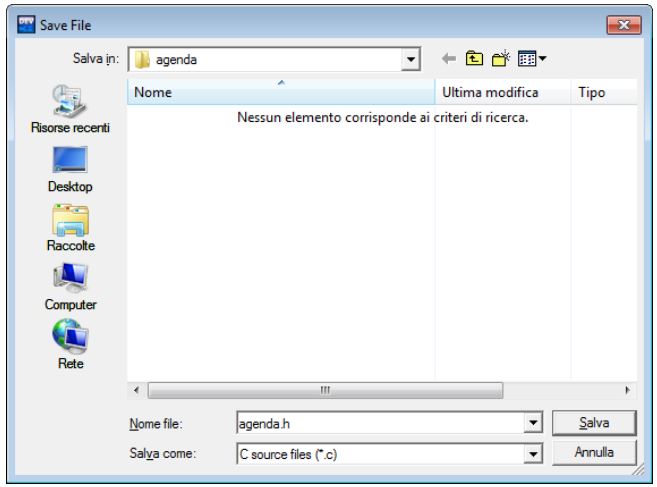
Esercizio 2



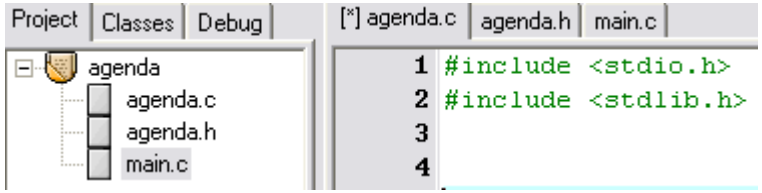








Ripetere procedimento per creare
2 file vuoti
agenda.h e agenda.c



Esercizio 2

Partendo dal file relativo al programma dell'Agenda (E13.0_agenda_inizio.c) suddividere il programma in più file

agenda.h

```
#ifndef __AGENDA_H__
#define __AGENDA_H__
// #define …
// typedef …
// I prototipi delle funzioni
#endif
```

Riga vuota!!!

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "agenda.h"

int main() {
…
}
```

agenda.c

```
#include <stdio.h>
#include <stdlib.h>
#include "agenda.h"

// implementazione delle funzioni
```

Esercizio 3 - continuazione esercizio 2

Si riscrivano le seguenti funzioni (usare il codice delle funzioni esistenti) utilizzando il formalismo del passaggio per indirizzo:

```
void p_aggiungiEvento(Tagenda* pa, Tevento e);

void p_stampaAgenda(const Tagenda* pa);

void p_inizializzaAgenda(Tagenda* pa);
```

si modifichi opportunamente il main per verificarne il funzionamento.

Funzioni esistenti

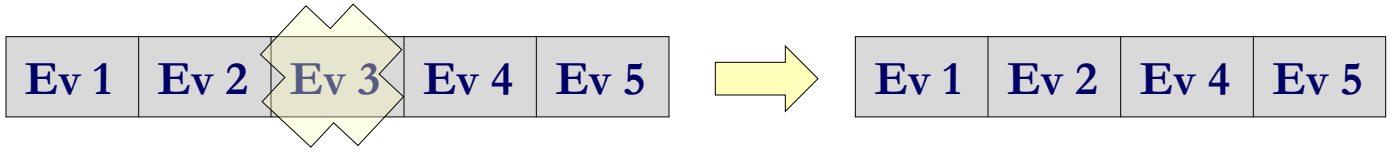
```
Agenda aggiungiEvento (Tagenda a, Tevento e){
   if (a.n_eventi >= N_MAX_EVENTI){
      printf("Errore, l'agenda e' piena\n");
   } else {
      a.eventi[a.n_eventi] = e;
      a.n_eventi++;
   }
   return a;
}
void stampaAgenda (Tagenda a){
   int i;
   for (i=0 ; i<a.n_eventi; i++){
       printf("Evento in posizione %d:\n", i);
       stampaEvento(a.eventi[i]);
       printf("\n\n");
   }
}
Agenda inizializzaAgenda (){
   Tagenda daRitornare;
   daRitornare.n_eventi = 0;
   return daRitornare;
}
```

Esercizio 4 - continuazione esercizio 2

- Si scriva e si provi la funzione **`cancellaUltimoEvento`**.

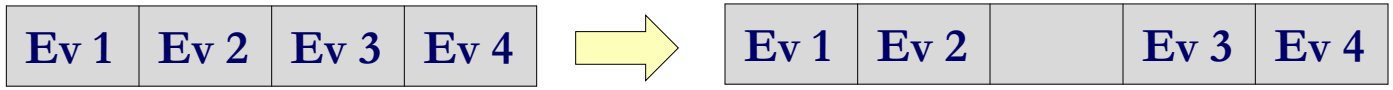
```
void cancellaUltimoEvento(Tagenda * pa);
```

- Si scriva la funzione **`cancellaEvento`** che abbia come argomento la posizione in cui si voglia cancellare. Ricordarsi di spostare a sinistra gli elementi dell'array di eventi dopo la cancellazione.




```
void cancellaEvento(Tagenda * pa, int pos);
```

- Si scriva la funzione **`inserisciEvento`** che abbia come argomento la posizione in cui si voglia inserire. Ricordarsi di spostare a destra gli elementi dell'array di eventi prima dell'inserimento.




```
void inserisciEvento(Tagenda * pa, int pos, Tevento e);
```

Esercizio 4

- Utilizzare la funzione "main" nella pagina successiva per verificare la correttezza del codice scritto.
- L'output che ci si aspetta dal programma è il seguente:

```
Evento in posizione 0:
Inizio: 10/06/2009 10:30
Fine: 10/06/2009 11:30
Piscina

Evento in posizione 1:
Inizio: 11/06/2009 14:30
Fine: 11/06/2009 16:00
Studio

Evento in posizione 2:
Inizio: 11/06/2009 16:30
Fine: 11/06/2009 17:30
Appuntamento

--------- cancello ultimo -----------
Evento in posizione 0:
Inizio: 10/06/2009 10:30
Fine: 10/06/2009 11:30
Piscina

Evento in posizione 1:
Inizio: 11/06/2009 14:30
Fine: 11/06/2009 16:00
Studio
```

```
------- inserisco in pos. 1 ---------
Evento in posizione 0:
Inizio: 10/06/2009 10:30
Fine: 10/06/2009 11:30
Piscina

Evento in posizione 1:
Inizio: 10/06/2009 13:30
Fine: 10/06/2009 15:30
Appuntamento

Evento in posizione 2:
Inizio: 11/06/2009 14:30
Fine: 11/06/2009 16:00
Studio

-------- cancello il primo ----------
Evento in posizione 0:
Inizio: 10/06/2009 13:30
Fine: 10/06/2009 15:30
Appuntamento

Evento in posizione 1:
Inizio: 11/06/2009 14:30
Fine: 11/06/2009 16:00
Studio
```

Esercizio 4 (main di prova)

```
int main(void) {
  TdataOra d1, d2;
  Tevento e;
  Tagenda a;

  a = inizializzaAgenda();
  d1 = inizializzaData(2009,6,10,10,30);  d2 = inizializzaData(2009,6,10,11,30);
  e = inizializzaEvento(d1,d2,PISCINA);
  aggiungiEvento(&a, e);
  d1 = inizializzaData(2009,6,11,14,30);  d2 = inizializzaData(2009,6,11,16,0);
  e = inizializzaEvento(d1,d2,STUDIO);
  aggiungiEvento(&a, e);
  d1 = inizializzaData(2009,6,11,16,30);  d2 = inizializzaData(2009,6,11,17,30);
  e = inizializzaEvento(d1,d2,APPUNTAMENTO);
  aggiungiEvento(&a, e);
  stampaAgenda(a);
  printf("--------- cancello ultimo -----------\n");
  cancellaUltimoEvento(&a);
  stampaAgenda(a);
  printf("------- inserisco in pos. 1 ---------\n");
  d1 = inizializzaData(2009,6,10,13,30);  d2 = inizializzaData(2009,6,10,15,30);
  e = inizializzaEvento(d1,d2,APPUNTAMENTO);
  inserisciEvento(&a, 1, e);
  stampaAgenda(a);
  printf("-------- cancello il primo ----------\n");
  cancellaEvento(&a, 0);
  stampaAgenda(a);

  system("PAUSE");
  return 1;
}
```

Esercizi aggiuntivi
Funzioni

Negli esercizi successivi è possibile definire ulteriori funzioni che si reputano utili e/o necessarie

Esercizio 5

- Definire una struttura per contenere i voti di al massimo N studenti

```
typedef struct Tvoti{ int voti[N]; int dim; } Tvoti;
```

- Il programma deve
  - 1) Inizializzare i voti con valori casuali compresi tra 18 e 30
  - 2) Stamapre i voti
  - 3) Collezionare i voti compresi tra 18 e 22 (compresi)
  - 4) Stampare la collezione del punto precedente
- Definire ed implementare le seguenti funzioni
  - inizializza dim elementi dell'array v

```
void initvoti(int v[],int dim);
```

  - stampa i primi dim dati presenti in array v

```
void stampa(int v[], int dim);
```

  - stampa i primi dim (campo di v) dati presenti in array voti (campo di v)

```
void p_stampa(const Tvoti *v);
```

  - restiuisce i voti presenti in v e compresi tra 18 e 22

```
Tvoti selezioneVoti(Tvoti v);
```

Esercizio 6

- Definire una struttura dati per memorizzare una persona `typedef struct Tpersona{ char nome[21]; int anno;} Tpersona;`
- Definire una struttura dati per memorizzare un insieme di persone `typedef struct Tvotanti{ Tpersona persone[N]; int dim; } Tvotanti;`
  - 1) Definire un array di N persone: Tpersona persone[N]
  - 2) Inizializzare l'array persone (dim effettiva = dim massima)
  - 3) Stampare l'insieme delle persone
  - 4) Collezionare le persone maggiorenni nell'anno 2017
- Implementare le seguenti funzioni
  - inizializza dim elementi di array p
    `void initpers(Tpersona p[],int dim);`
  - restituisce persone con eta' maggiore di 18 nell'anno a
    `Tcollezione selezionePers(const Tpersona p[], int dim, int a);`
  - stampa i primi dim elementi di array p
    `void stampa(const Tpersona p[], int dim);`

Esercizio 7

- Definire una struttura dati per memorizzare il voto di uno studente

```
typedef struct Tstudente{ int voto; char nome[20];
} Tstudente;
```

  - 1) Definire un array di N studenti: Tstudente studenti[N];
  - 2) Inizializzare array con voti compresi tra 16 e 30 e nomi casuali
  - 3) Stampare il contenuto dell'array
  - 4) Collezionare gli studenti che hanno un voto pari a 16 o 17
  - 5) Stampare il contenuto degli studenti con voti 16 o 17
- Definire le seguenti funzioni
  - `int casuale(int valmin, int valmax);`
  - stampa i dati di uno studente
    `void stampa(Tstudente s);`
  - stampa i dati di tutti gli studenti presenti nell'array s
    `void stampaTutto(const Tstudente s[], int dim);`

Esercizio 8

Definite le seguenti strutture dati

```
typedef struct Tstudente{ char nome[20]; int voto;
} Tstudente;
typedef Tstudente studenti[N];
typedef struct S_respinti{ Tstudente studenti[N]; int dim;
}S_respinti;
```

- 1) Inizializzare array studenti s;
- 2) Selezionare studenti con voto < 18
- 3) Stampare l'elenco di tutti gli studenti e poi l'elenco di quelli con voto <18

Definire le seguenti funzioni

- inizializzare dim studenti con nome e voto chiesto all'utente
  `void init(studenti mieis,int dim);`
- stampa insieme studenti
  `void stampaLista(const studenti mieis, int dim);`
- stampa i dati di un singolo studente
  `void stampaStudente(Tstudente mios);`
- colleziona gli studenti il cui voti e' minore di 18
  `S_respinti DaInterrogare(studenti mieis, int dim);`