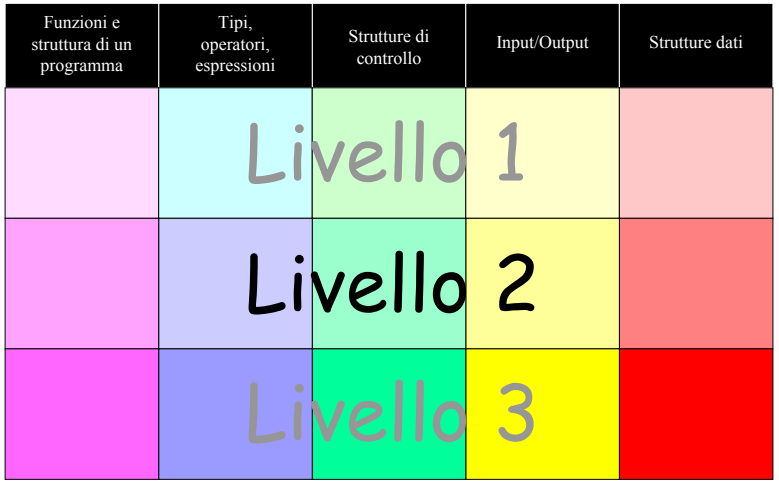
ROADMAP



FUNZIONI E STRUTTURA DI UN PROGRAMMA

- funzioni
- procedure
- visibilità e tempo di vita delle variabili

STRUTTURA DI UN PROGRAMMA

- In generale, abbiamo già visto:
  - il concetto di sottoprogramma
  - la sua definizione all'interno o all'esterno del programma principale
  - meccanismo di chiamata (*call*)
  - la necessita` di definire delle regole per gestire la comunicazione con il programma principale

SOTTOPROGRAMMI IN "C"

- Dove vengono definiti?
- Come vengono definiti?
- Come vengono richiamati?
- Come viene gestita la comunicazione fra (sotto)programmi chiamanti e sottoprogrammi chiamati?

- Vedremo due tipi di sottoprogrammi:
  - funzioni (vere e proprie)
  - procedure (un caso particolare di funzioni)

STRUTTURA DI UN PROGRAMMA "C"

Parte direttiva

Programma principale (main)

STRUTTURA DI UN PROGRAMMA "C"

Parte direttiva

```
main() {  /*programma principale*/
Parte dichiarativa
   ▪ dichiarazioni di costanti
   ▪ dichiarazioni di variabili
Parte esecutiva
}
```


STRUTTURA DI UN PROGRAMMA "C"

Parte direttiva

Programma principale (main)

Funzioni

Procedure

STRUTTURA DI UN PROGRAMMA "C"

Parte direttiva

Parte dichiarativa globale

Programma principale (main)

Funzioni

Procedure

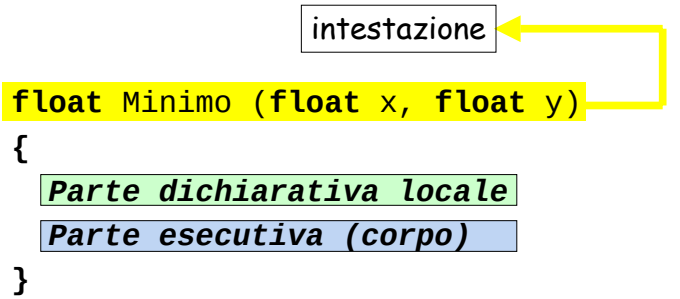
- Contiene la dichiarazione di tutti gli elementi che sono condivisi, ossia usati in comune, dal programma principale e dai sottoprogrammi

DEFINIZIONE DELLE FUNZIONI

- Una funzione opera su dati e **ritorna un risultato**
- Ha la forma:

```
tipo-ritornato nome-funz (dichiarazione argomenti)
{
      dichiarazioni ed istruzioni
}
```


DEFINIZIONE DELLE FUNZIONI



- Si osservino le similitudini con la struttura del programma principale `main`
- In "C" lo stesso `main` è una funzione!

L'INTESTAZIONE

- Contiene le informazioni più rilevanti per l'uso corretto della funzione
- E' costituita da:
  - il **tipo del risultato**
  - l'**identificatore** della funzione
  - la **lista dei parametri** racchiusa tra **( )**

```
float Minimo (float x, float y)
```


L'INTESTAZIONE

- Il **tipo del risultato** di una funzione può essere
  - predefinito
  - definito dall'utente
- L'**identificatore** della funzione viene utilizzato nelle *call*
  - identifica la "variabile risultato"
- La **lista dei parametri** elenca gli *argomenti* della funzione specificandone il tipo
  - successione di dichiarazioni di **parametri formali** separate tra loro da una virgola

LE DICHIARAZIONI LOCALI

- Definiscono tutti gli oggetti necessari alla realizzazione dell'operazione astratta
- Obbediscono alle stesse leggi per la costruzione della parte dichiarativa del `main`

PARTE ESECUTIVA (CORPO)

- Costruito con le stesse regole del `main`
- Al suo interno si trova un'istruzione **`return`**

```
return espressione;
```

  - assegna alla variabile risultato (cioe` alla funzione stessa) il valore dell'espressione
  - fa si` che l'esecuzione della funzione termini
- Il valore dell'espressione nell'istante di esecuzione dell'istruzione **`return`** sara` il risultato della funzione

PARTE ESECUTIVA (CORPO)

- Possono essere presenti piu` istruzioni **`return`**
  - solo la prima che si incontra nel corso dell'esecuzione della funzione verra` eseguita
- Se nessuna istruzione di **`return`** e` presente nel corpo della funzione o se quelle presenti non vengono eseguite
  - la funzione termina in corrispondenza del simbolo **`}`** che conclude il corpo della funzione
  - il risultato della funzione risulta indefinito provocando una segnalazione di errore

ESEMPIO 8

```
/*funzione che produce la radice intera
  del parametro*/
int RadiceIntera (int par)
{
   int cont;

   cont=0;
   while (cont*cont <= par)
      cont++;
   return cont-1;
}
```

CHIAMATA DELLE FUNZIONI

- All'interno di una espressione in un programma o un sottoprogramma
- La sintassi di una chiamata di funzione consiste
  - nell'identificatore della funzione seguito dalla
  - lista dei **parametri attuali** racchiusa tra **( )**
    - indicano i *valori degli argomenti* rispetto ai quali la funzione deve essere calcolata
    - la *corrispondenza* tra parametri segue l'ordine della dichiarazione (**devono essere dello stesso numero!**)
    - il *tipo* dei parametri attuali deve essere compatibile con quello dei corrispondenti parametri formali

ESECUZIONE E PASSAGGIO DEI PARAMETRI DELLE FUNZIONI

Lo vediamo attraverso un esempio

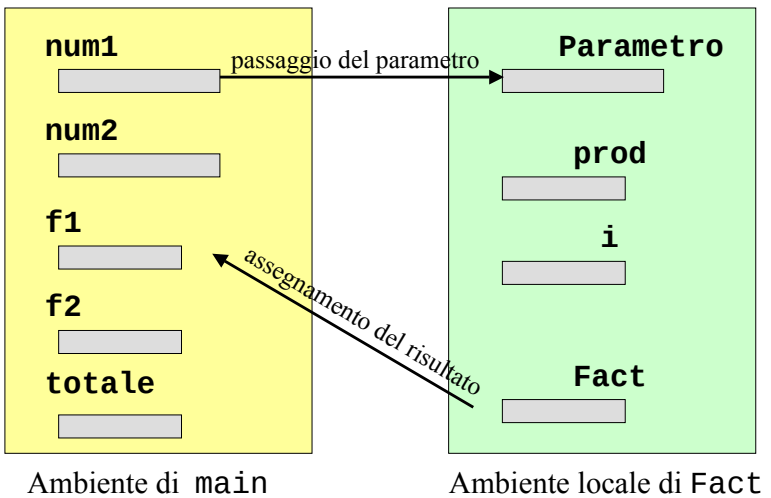
ESEMPIO 9

```
main()                                  {
  int num1, num2, f1, f2, totale;
  scanf ("%d", &num1);
  scanf ("%d", &num2);
  f1=Fact(num1); f2=Fact(num2);
  totale = f1 + f2;
                                        }
int Fact(int Parametro)                 {
  int i, prod;
  prod=1;
  for(i=1;i<=Parametro;i++) prod = prod*i;
  return prod;                          }
```

Chiamate di funzione

Dichiarazioni locali

ESEMPIO 9 (cont.)



NOTE

- L'**ambiente locale** di una funzione e` dato
  - da tutte le variabili dichiarate al suo interno
  - dai suoi parametri formali
  - da una variabile "identificata" dal nome della funzione non direttamente referenziabile all'interno del corpo della funzione
- La **chiamata** comporta
  - passaggio dei parametri
  - cessione del controllo alla funzione
  - ritorno del valore della funzione al programma chiamante

NOTE

- Il passaggio dei parametri copia il valore di ogni parametro attuale nella cella del corrispondente parametro formale
- A questo punto l'esecuzione del `main` viene sospesa e inizia quella della funzione `Fact`
  - ogni riferimento al parametro formale determina l'accesso alla corrispondente variabile locale
  - al termine la cella `Fact` contiene il risultato dalla funzione rispetto al valore degli argomenti
- All'istruzione **`return`** il controllo torna al `main` che preleva il valore della cella `Fact` e lo assegna a `f1`

```
f1=Fact(num1);
```


ESEMPIO 10

```
main()                          {
  int num;
  scanf ("%d", &num);
  if(Pari(num))
   istruzioneA ;
  else istruzioneB ;            }

int Pari(int parametro)         {
  int Decisione, i, somma;
  Decisione=0; somma=0;
  for(i=1;i<=parametro;i++) somma = somma +i;
  if ((somma%2)==0) Decisione=1;
  return Decisione;             }
```

Chiamata di funzione

Dichiarazioni locali

DICHIARAZIONI DI FUNZIONI

- Le funzioni possono essere definite
  - in coda ad un programma principale
  - in file distinti
    - scritti dall'utente
    - propri del sistema "C" → **funzioni di libreria**
- In generale, in un programma "C" una funzione puo` essere chiamata purche` risulti **definita** oppure **dichiarata**

  → la dichiarazione di una funzione e` detta **prototipo**

PROTOTIPI DI FUNZIONI

- Si limitano a richiamare l'**intestazione** di una funzione:
  `tipo-ritornato nome (dich. argomenti);`
- Vengono inseriti
  - nella parte dichiarativa globale, oppure
  - nella parte dichiarativa del codice che richiama la funzione
- Il compilatore puo` cosi` controllare immediatamente la correttezza della *call*
  - il numero e il tipo dei parametri utilizzati
  - il tipo del risultato restituito dalla funzione

FUNZIONI DI LIBRERIA

- Standard library del linguaggio "C", costituite per
  - operazioni di I/O
  - operazioni matematiche/aritmetiche
  - operazioni di gestione della memoria
  - operazioni di gestione di caratteri e di stringhe
  - operazioni di ricerca e ordinamento
  - operazioni di comunicazione con il sistema operativo
  - operazione per la gestione degli errori
  - operazioni di gestione di date e tempo
  - operazioni di utilita` generali

FUNZIONI DI LIBRERIA

- Le funzioni di libreria "C" sono disponibili come file di codice compilato
  → non leggibili direttamente dal programmatore
- La dichiarazione di tali funzioni avviene tramite **l'inclusione dei loro prototipi** raggruppati in appositi file del sistema "C" detti **header**

File header	Funzioni di libreria
`stdio.h`	`printf, scanf, fread, fwrite, ...`
`math.h`	`sin, cos, log, fabs, floor, ...`
`. . .`	`. . .`

IL RUOLO DEGLI #include

- L'inclusione degli **header** avviene nella prima fase di compilazione per mezzo del preprocessore C e viene specificato dall'istruzione `#include`

  Esempio:

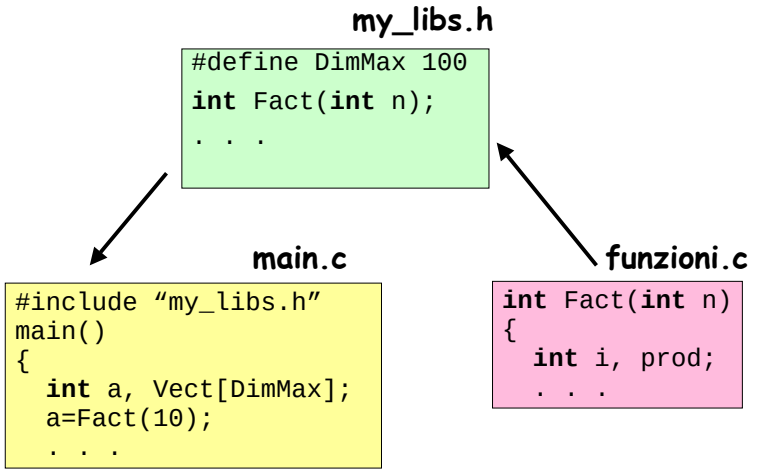
  ```
  #include <stdio.h>
  ```

  - copia il contenuto del file `stdio.h` nel programma
  - inserisce i prototipi delle funzioni che appartengono al gruppo di cui `stdio.h` e` file header

NOTE

- Sintatticamente esistono due forme di inclusione:
  `#include “nome-file”` oppure
  `#include <nome-file>`
- Un file incluso puo` a sua volta contenere delle linee di codice di tipo `#include`
- In generale, gli **header file** vengono utilizzati per raccogliere la definizione e la dichiarazione di “oggetti” (variabili, funzioni, tipi, ecc) condivisibili da vari file

HEADER FILE



DEFINIZIONE DELLE PROCEDURE

- Una procedura e` una particolare funzione che opera su dati ma **non ritorna alcun risultato**

```
void nome-funz (dichiarazione argomenti)
{
      dichiarazioni ed istruzioni
}
```

- **`void`** e` uno speciale tipo che indica l’assenza di un risultato
- In generale, produce in uscita una manipolazione di dati condivisi con il programma chiamante (*ambiente globale*)

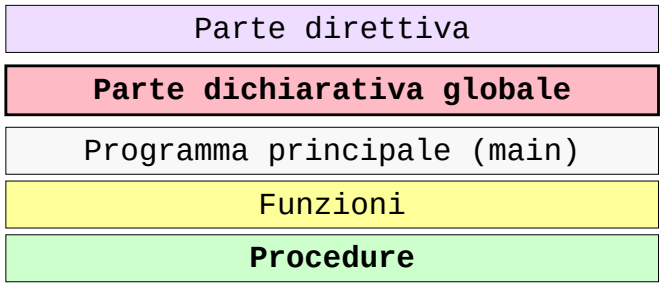
ESEMPI DI INTESTAZIONE

```
void Stampa (char lettera);

void Inserisci (char lettera, int pos);
```

- Il tipo **`void`** puo` essere impiegato anche come tipo dei parametri formali nel caso non esistano parametri passati alla funzione o alla procedura

```
void Pippo(void);
void main(void);
```

AMBIENTE GLOBALE



- La presenza di dichiarazioni nella parte dichiarativa globale del programma genera un ambiente globale del programma

ESECUZIONE E PASSAGGIO DEI PARAMETRI DELLE PROCEDURE

Lo vediamo attraverso un esempio

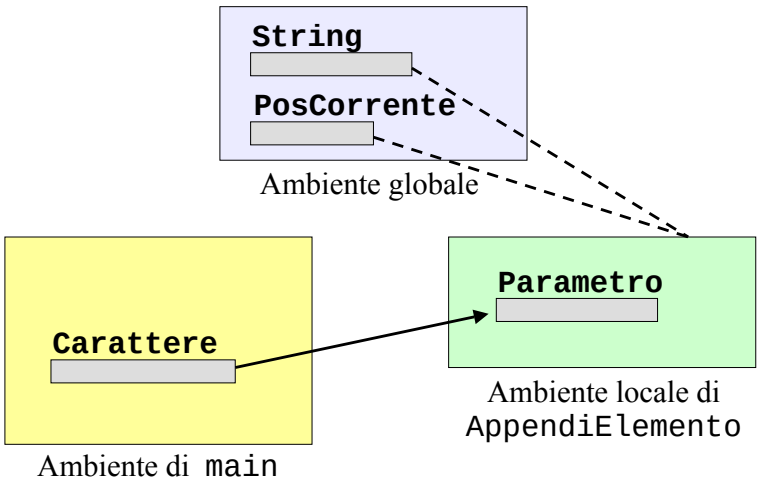
ESEMPIO 11

```
#include <stdio.h>

int PosCorrente;
char String[30];
void AppendiElemento(char Parametro);
main()                                  {
  char Carattere;
  scanf ("%c", &Carattere);
                          . . .
  AppendiElemento(Carattere);           }

void AppendiElemento(char Parametro)    {
  PosCorrente++;
  String[PosCorrente]=Parametro;        }
```

Dichiarazioni globali

Call

ESEMPIO 11 (cont.)



NOTE

- Una procedura durante la sua esecuzione *accede all'ambiente globale, modificandolo*
- Al termine dell'esecuzione cede di nuovo il controllo al programma principale, *senza produrre alcun risultato*
- L'effetto desiderato e` gia` stato ottenuto come modifica delle variabili globali `PosCorrente` e `String`, e quindi accessibili, anche al programma principale

FUNZIONI *vs* PROCEDURE

- Il rapporto tra un programma principale e un sottoprogramma e` simile a quello tra "padrone" e "servitore"
  - A un servitore di tipo funzione vengono dati incarichi del tipo *"Vai a comperare tre chili di mele con questi soldi"*
  - A un servitore di tipo procedura vengono invece dati incarichi del tipo *"Inserisci questo libro nella mia libreria"*
- Chiaramente, il secondo incarico richiede maggior fiducia, poiche' comporta l'accesso a una risorsa di proprieta` del padrone

ASPETTI AVANZATI NELL'USO DEI SOTTOPROGRAMMI

Un sottoprogramma puo` accedere a variabili che sono dichiarate nel programma principale?

Una variabile dichiarata in un sottoprogramma esiste mentre il programma principale viene eseguito?

- Esistono regole che gestiscono la
  - **visibilità** delle variabili (*scope rules*)
  - **tempo di vita** delle variabili durante l'esecuzione di un programma

IL CONCETTO DI BLOCCO

- In C un blocco e` composto da due parti racchiuse tra **`{ }`**
  - una parte dichiarativa (facoltativa)
  - una sequenza di istruzioni
- I blocchi possono essere **paralleli** o **annidati**
- Lo stesso `main` e le singole funzioni di un programma, presentano, dopo la propria intestazione, un blocco → blocchi particolari:
  - dotati di **identificatore**, di **parametri di ingresso** e di un **eventuale risultato**

ESEMPIO 12

```
#include <stdio.h>                          Parte direttiva
int g1, g2;
char g3;                                    Dichiarazioni globali
int f1(int par1, int par2);                 prototipo della funzione f1
main()                                  {
 int a,b;                                   Dichiarazioni del main
 int f2(int par3, int par4);                prototipo della funzione f2
 . . .
 {                                          blocco1
   char a,c;
   . . .
  {                                         blocco2 annidato nel blocco1
    float a;
    . . .
  }                                         fine blocco2
 }                                          fine blocco1
                                        }   fine main
```

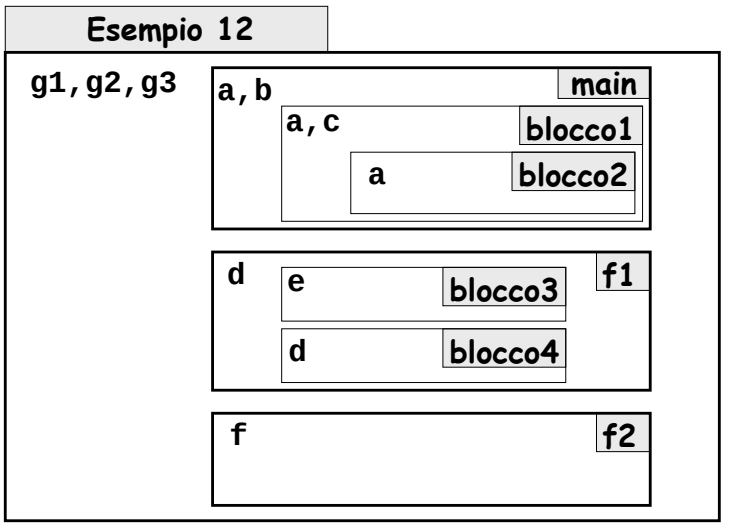

ESEMPIO 12 (cont.)

```
int f1(int par1, int par2) {    Definizione della funzione f1
   int d;
    . . .
   {                                        blocco3
     int e;
     . . .
   }                                        fine blocco3
   {                                        blocco4
     int d;
     . . .
   }                                        fine blocco4
                                }   fine f1

int f2(int par3, int par4)  {   Definizione della funzione f2
   int f;
   . . .
                                }   fine f2
```


AMBITO VISIBILITA` VARIABILI

- **Ambiente globale di un programma:**
  - l'insieme di tutti gli elementi dichiarati nella sua parte dichiarativa globale
- **Ambiente locale di una funzione:**
  - l'insieme di tutti gli elementi dichiarati nella sua parte dichiarativa e nella sua intestazione
- **Ambiente di un blocco:**
  - l'insieme di tutti gli elementi dichiarati nella sua parte dichiarativa

I DIVERSI AMBIENTI NELL'ES. 12



REGOLE DI VISIBILITA` IN "C"

- In generale, **l'ambito di visibilita`** di una variabile e` determinato dalla posizione della dichiarazione:

  per l'ambiente globale

  – tutte le funzioni e i blocchi che costituiscono il programma

  per l'ambiente locale di una funzione

  – le istruzioni e le parti eseguibili dei blocchi in essa contenuti

  per l'ambiente di un blocco

  – le istruzioni proprie del blocco e di quelli annidati

REGOLE DI VISIBILITA` IN "C"

- Il concetto di ambiente e le regole di visibilità non riguardano solamente le variabili, ma anche le funzioni e gli identificatori ad esse associati
  – uso dei prototipi

  Se la chiamata non e` preceduta dalla dichiarazione dell'intestazione della funzione o dalla sua definizione il compilatore e` obbligato a **formulare delle ipotesi sul tipo dei parametri del risultato**, ipotesi che deve poi verificare in un secondo tempo

LA DURATA DELLE VARIABILI

- Due categorie di variabili:
  – **variabili fisse** o **statiche**
    - vengono allocate una sola volta e vengono distrutte solo quando termina l'esecuzione del programma
    - i loro valori vengono mantenuti anche all'esterno del loro ambito di visibilita`
  – **variabili automatiche**
    - vengono create ogni volta che si entra nel loro ambito di visibilita` e vengono distrutte all'uscita da tale ambiente
    - i loro valori non vengono mantenuti all'esterno del proprio ambito di visibilita`

LA DURATA DELLE VARIABILI

- Sono, in generale, **variabili fisse**:
  – le variabili globali del programma
- Sono, in generale, **variabili automatiche**:
  – quelle dichiarate a livello di funzione (inclusi i parametri di ingresso e la variabile del risultato) e quelle dichiarate a livello di blocco

Le variabili appartenenti ad ambiente di funzione o ad ambiente di blocco possono essere dichiarate a durata fissa se la loro definizione e` preceduta dalla parola chiave **`static`**

```
static int d;
```

TIPI, OPERATORI ED ESPRESSIONI

- dichiarazione `typedef`
- qualificatori di tipo

DICHIARAZIONE DI TIPO

- Per creare nuovi nomi di tipi di dati, il "C" fornisce uno strumento chiamato **`typedef`**
- I nuovi nomi devono essere dichiarati attraverso una specifica **dichiarazione di tipo**
- Per esempio, la dichiarazione

```
typedef int Anno;
```

definisce `Anno` come sinonimo di `int`

- Il tipo `Anno` può essere utilizzato nelle dichiarazioni, nei cast, ecc., esattamente come il tipo `int`

DICHIARAZIONI DI TIPO

```
typedef  float Vettore[20];
typedef  char Carattere_Speciale;
typedef  char Stringa[20];
```

- Con queste dichiarazioni di tipo è possibile dichiarare variabili come:

```
Vettore  v1, v2;
Carattere_Speciale  ch1;
Stringa  Nome, Cognome;
```


DICHIARAZIONI DI TIPO

- La dichiarazione **`typedef`** non crea un nuovo tipo, bensì aggiunge un **nuovo nome** a un tipo già esistente
  - Le variabili dichiarate in questo modo hanno le stesse caratteristiche di quelle dichiarate in modo esplicito
  - **Non consente di definire nuove operazioni** applicabili all'insieme dei valori specifici del nuovo tipo
  - a volte possono rendere il codice più leggibile
- La dichiarazione dei tipi avviene tra la dichiarazione delle costanti e quella delle variabili

QUALIFICATORI DI TIPO

Tipo base	Qualificatore
`char`	`signed` / `unsigned`
`int`	`signed` / `unsigned` `short` / `long`
`float`	-
`double`	`long`

QUALIFICATORI DI TIPO

- **`short`** e **`long`** condizionano
  - lo **spazio allocato** dal compilatore per la memorizzazione delle variabili tipizzate

`short int` ≤ `int` ≤ `long int`
`float` ≤ `double` ≤ `long double`

- lo spazio allocato dipende dalla macchina

QUALIFICATORI DI TIPO

- **`signed`** e **`unsigned`** condizionano
  - **l'insieme dei valori** che la variabile tipizzata può assumere nonché il valore massimo e minimo assumibile dalla stessa
- La rappresentazione binaria di un tipo di dato qualificato come **`signed`** prevede l'uso di un *bit* per la rappresentazione del segno
  - si ha a disposizione un bit in meno per la rappresentazione del valore
- Non cambia lo spazio allocato in memoria

QUALIFICATORI DI TIPO

Tipo	Allocazione memoria	Insieme di rappresentazione
`unsigned int`	2 byte	$\{0, 2^{16}-1\}$
`signed int`	2 byte	$\{-2^{15}, 2^{15}-1\}$
`unsigned char`	1 byte	{0, 255}*
`signed char`	1 byte	{-128, 127}*

*** codice numerico dell'insieme dei caratteri ASCII**

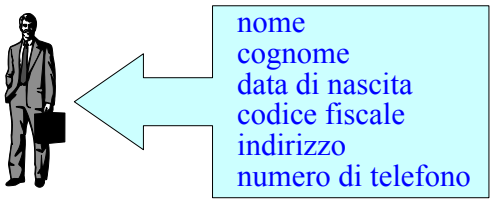
N.B. i valori nella tabella sono indicativi e possono variare da macchina a macchina

STRUTTURE DATI

- Il costruttore `struct`

LE STRUTTURE

- Collezione di una o più variabili, **eventualmente di tipi diversi**, raggruppate da un nome comune
  - aiutano ad organizzare dati complessi
  - consentono di trattare come un unico oggetto un insieme di variabili correlate
- utili per descrivere un oggetto attraverso i suoi attributi



DICHIARAZIONE DI UNA struct

- Il "C" permette di definire strutture mediante un'apposita dichiarazione che consiste in:
  - la parola chiave **`struct`**
  - l'**elenco** degli elementi (**campi**) della struttura racchiuso tra **`{ }`**
    - ogni elemento specifica il **tipo del campo** e l'**identificatore del campo**
    - gli elementi dell'elenco sono separati da **`;`**
  - l'**identificatore** (gli identificatori) di variabile (se ve ne sono più di uno sono separati da **`,`**)
  - il **terminatore** di dichiarazione **`;`**

IL COSTRUTTORE struct

```
struct  { char  Nome[30];
          char  Cognome[30];
          int   Telefono;
        } Stud1, Stud2;
```

Identificatori della struttura

- Una dichiarazione **`struct` definisce un tipo**
  - Le variabili `Stud1` e `Stud2` sono definite come strutture di tipo **`struct`**

N.B. come gli array, le variabili `struct` sono variabili strutturate, **ma possono aggregare dati di tipo diverso**

ACCESSO AI CAMPI DELLA struct

- Per accedere ai singoli campi di una struttura si scrive **l'identificatore della struttura** seguito da un *punto* e dall'**identificatore del campo** desiderato

☞*"dot notation"*

```
Stud1.Nome
Stud1.Cognome
Stud1.Telefono
```

Possono essere usate come normali variabili con le restrizioni imposte dal loro tipo specificato nella dichiarazione di struttura

IL COSTRUTTORE DI TIPO struct

- La dichiarazione esplicita di un tipo ottenuto mediante il costruttore **`struct`** consiste in:

```
typedef struct  { char  Nome[30];
                  char  Cognome[30];
                  int   Telefono;
                } Studenti;
```

Identificatore del nuovo tipo struttura

- Una o più variabili di tipo `Studenti` possono essere dichiarate semplicemente scrivendo:

```
Studenti  Stud1, Stud2;
```


IL COSTRUTTORE DI TIPO struct

- I costruttori di tipo **`struct`** si possono combinare tra loro

```
typedef struct  { int   giorno;
                  int   mese;
                  int   anno;
                } Data;

typedef struct  { char  Nome[30];
                  char  Cognome[30];
                  int   Telefono;
                  Data  DataIscrizione;
                } Studenti;
```


ACCESSO AI CAMPI DELLA struct

- I meccanismi di accesso a elementi di variabili strutturate si possono combinare tra loro

Alcuni esempi:

```
Stud1.DataIscrizione.giorno = 3
Stud1.DataIscrizione.mese = 7
Stud1.DataIscrizione.anno = 2000
```

```
if (Stud2.Cognome[0] == 'A')
```

ACCESSO AI CAMPI DELLA struct

```
typedef struct  { char  Destinatario[30];
                  int   Importo;
                  Data  DataEmissione;
                } DescrizioneFatture;

DescrizioneFatture  ArchivioFatture[1000];
```

```
if (ArchivioFatture[500].DataEmissione.Anno <= 1991)
   printf("%d", ArchivioFatture[500].Importo);
else
   printf("La fattura in questione e' stata emessa
           dopo il 1991\n";
```

- Usando identificatori appropriati, il C permette di scrivere un codice molto simile al linguaggio naturale

OSSERVAZIONI SULLE struct

- Gli elementi di una struttura possono essere coinvolti singolarmente in tutte le operazioni proprie del tipo che li caratterizza
- Nel suo complesso, la struttura puo' anche essere coinvolta - tramite il suo identificatore - in operazioni di confronto (==, !=) o assegnamento

```
Studenti Stud1, Stud2;

Stud1 = Stud2;
```

Assegna tutti i campi della variabile Stud2 a quelli di Stud1

STRUTTURE DI CONTROLLO

- `switch`

L'ISTRUZIONE DI SELEZIONE switch

- Si presta a indicare **scelte plurime** sulla base del valore di una particolare variabile o espressione
- Consiste in:
  - la parola chiave **`switch`**
  - un'espressione racchiusa tra **`( )`**
  - una sequenza di istruzioni **`case`** racchiuse tra **`{ }`**
    Ogni istruzione **`case`**, a sua volta consiste in:
    - la parola chiave **`case`**
    - un'espressione **costante** seguita dal separatore **`:`**
    - una o più istruzioni separate da **`;`**
    - l'istruzione **`break`** che causa l'uscita da **`switch`**

L'ISTRUZIONE DI SELEZIONE switch

```
switch (espressione)
{
     case espr-cost1 : Istruzioni A;
                       break;
     case espr-cost2 : Istruzioni B;
                       Istruzioni C;
                       break;
}
```

Avvia un processo di selezione sulla base del valore dell'espressione

Specifica il valore della espressione per cui devono essere eseguite le istruzioni associate

Provoca l'uscita immediata dallo **switch**

L'ISTRUZIONE DI SELEZIONE switch

```
switch (espressione)
{
     case espr-cost1 : Istruzioni A;
     case espr-cost2 : Istruzioni B;
                       Istruzioni C;
}
```

Cosa accade senza l'istruzione **break**?

Poiché i casi servono soltanto come etichette, l'esecuzione delle istruzioni associate ad uno di essi è seguita dall'*esecuzione sequenziale dei casi successivi*.

L'ISTRUZIONE DI SELEZIONE switch

Modalità di esecuzione dell'istruzione **switch**:

- **Valuta** l'espressione tra **( )**
  - l'espressione deve essere di tipo *integral* (**int** oppure **char**)
- **Confronta** il valore dell'espressione con l'insieme dei valori specificati nelle "clausole" **case**
- Se il valore appartiene ad uno degli insiemi specificati **esegue** le istruzioni ad esso associate
- Se il valore non appartiene a nessuno degli insiemi non viene eseguito nulla all'interno dell'istruzione **switch** e l'esecuzione del programma continua con la prima istruzione che segue lo **switch**

ESEMPIO 13

```
. . .
int classe, ClasseScelta;
float x;
. . .
switch (classe) {
  case 1:   x = (x*15)/12.;
            ClasseScelta = 1;
            break;
  case 2:   x = (x*14)/12.;
            ClasseScelta = 2;
            break;
  }
. . .
```

L'ISTRUZIONE DI SELEZIONE switch

- Se la stessa sequenza di istruzioni deve essere eseguita per più valori dell'espressione costante
  - E' possibile costruire una sequenza di clausole **`case`** separate da **`:`** e concluse da **`:`**

```
case 'A': case 'G': case 'H':
    Istruzioni_Associate;
```

- Per indicare istruzioni da eseguire **comunque** anche se nessuno dei casi precedenti è stato soddisfatto si usa il caso identificato dalla parola chiave **`default`** (è opzionale)

```
default : Istruzioni_di_Default;
```


ESEMPIO 14

```
. . .
char CarattereLetto;
. . .
switch (CarattereLetto) {
  case 'A': case 'G': case 'H':
  printf("Il carattere letto e` A o G o
  H\n");
            break;
  default : printf("Il carattere letto e`
  sbagliato\n");
            break;
}
. . .
```


L'ISTRUZIONE DI SELEZIONE switch

Per un corretto uso del costrutto **`switch`** vanno tenute presenti le seguenti considerazioni:

- rispettare *l'esclusività dei casi*
  - per evitare ambiguità è necessario che i valori delle espressioni nei **`case`** siano tutti diversi fra loro
- Il trattamento *completo dei diversi casi*
  - può essere assicurato dalla presenza dell'istruzione **`default`**

ESERCIZIO

- *Scrivere un programma che legge i caratteri del nome dell'utente (il carattere # termina la sequenza di immissione) e trasforma ciascuno di essi in una nota musicale.*

Nota: Per semplicita` si considerino solo sette note (do, re, mi, fa, sol, la si).

RISOLUZIONE

```
#include <stdio.h>        /* programma melodia */
main() {
  char C;
  int resto;
  printf("Inserisci il primo carattere del tuo
  nome\n");
  scanf("%c", &C);
  while (C != '#')  {
    resto = C % 7;
    switch (resto) { }
    printf("Inserisci il prossimo carattere del
  tuo nome (per terminare digita #)\n");
  scanf("%c", &C);
  }
}
```


RISOLUZIONE (cont.)

```
#include <stdio.h>        /* programma melodia */
main() {
  char C;
  int resto;
  printf("Inserisci il primo carattere del tuo
  nome\n");
  scanf("%c", &C);
  while (C != '#')  {
    resto = C % 7;
    switch (resto) { }
    printf("Inserisci il prossimo carattere del
  tuo nome (per terminare digita #)\n");
  scanf("%c", &C);
  }
}
```

L'operatore % produce il resto della divisione intera dei suoi operandi

RISOLUZIONE (cont.)

```
#include <stdio.h>
main() {
  char C;
  int resto;
  printf("Inserisci il primo carattere del tuo
         nome\n");
  scanf("%c", &C);
  while (C != '#')  {
    resto = C % 7;
    switch (resto) { }
    printf("Inserisci il prossimo carattere del
  tuo nome (per terminare digita #)\n");
  scanf("%c", &C);
  }
}
```

Blocco di selezione multipla "switch"

RISOLUZIONE (cont.)

```
switch (resto)
{
  case 0: printf("Il carattere %c corrisponde
              alla nota 'do'\n", C);
          break;
  case 1: printf("Il carattere %c corrisponde
              alla nota 're'\n", C);
          break;
  case 2: printf("Il carattere %c corrisponde
              alla nota 'mi'\n", C);
          break;
  . . .   . . .
  case 6: printf("Il carattere %c corrisponde
              alla nota 'si'\n", C);
          break;
}
```