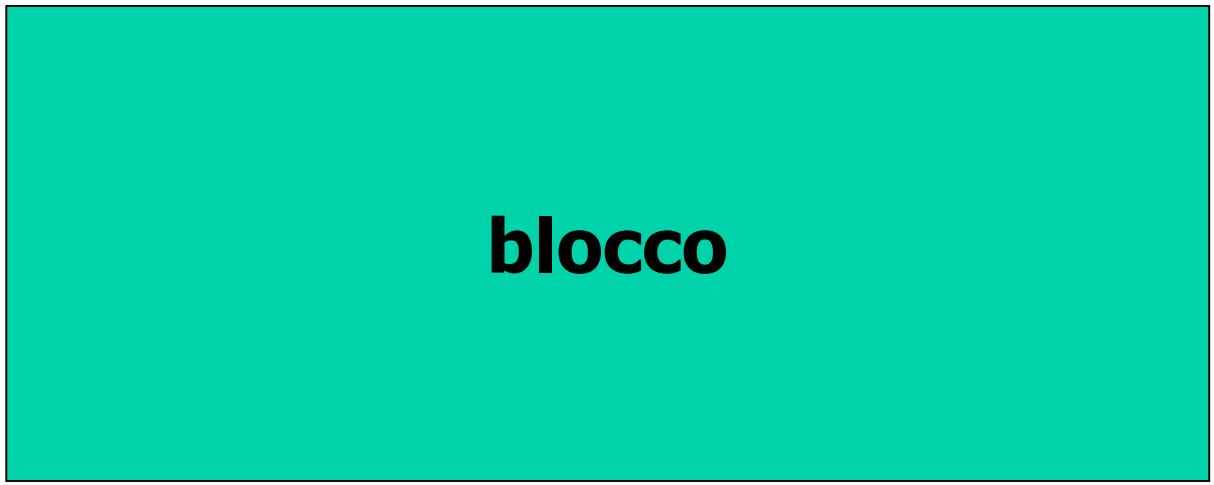
Il linguaggio C

Notate che ...

Il C è un linguaggio a blocchi

```
int main (void) {
```




```
}
```

Il C è un linguaggio a blocchi (2)

Non è possibile mischiare dichiarazioni e comandi !

```
int main (void) {
```

Dichiarazione di variabili locali

Comandi

```
}
```

Variabili non inizializzate

Le variabili non inizializzate non producono nessun errore!!!!

```
int main (void) {
   /* ed esempio …. */
   int min, max, out;
   min = max * out;
   …


}
```

Controllate sempre di aver inizializzato le variabili prima del loro uso!

Il linguaggio C

Tipi di dato, costanti, operatori

I tipi di dato primitivi

- Interi : `short, int, long`
  - 2,4,8,16 byte (dipende dalla implementazione)
  - segnaposto
    - `%d` rappresentazione decimale
    - `%x, %o` rappresentazione esadecimale o ottale
  - costanti intere : `1, -2, 4565`
  - La funzione predefinita `sizeof( )` fornisce la lunghezza in byte di un qualsiasi tipo o variabile C
    - es. `sizeof(int)`

I tipi di dato primitivi (2)

- Caratteri :
  - `char`, 1 byte, cod. ASCII, segnaposto `%c`
  - costanti carattere : `` `a`, `b`, `c` ``
  - i caratteri sono visti da C come interi un po' speciali : è possibile operare sui caratteri con le normali operazioni su interi
  - es:

```
char c = `a`;
c++;
putchar(c);   /* ???? */
```

I tipi di dato primitivi (3)

- Reali :
  - `float, double, long double`
  - 4,8,16 byte
  - rappresentazione in virgola mobile
  - segnaposto :
    - stampa (es. printf ( )) `%f, %lf`
    - lettura (es. scanf( )) `%f` (float) , `%lf` (double) , `%Lf` (long double)

I tipi di dato primitivi (4)

- Libreria matematica (`libm`)
  - fornisce tutte le più comuni operazioni matematiche per operare sui reali
    - es. `sqrt( )`
  - per utilizzarla includere il file
    - `#include <math.h>`
  - compilare con

    `gcc … … … -lm`

    effettua il link esplicito della libreria

I tipi di dato primitivi (5)

- Non esistono i tipi `byte` e `bool` !!!
- Le stringhe sono rappresentate come array di caratteri
- **`signed/unsigned`**
  - modificatore per i tipi interi e caratteri
  - permette di scegliere fra la rappresentazione di soli interi positivi (es. 0,255 su un byte) o quella in complemento a due (es. -128,+127)
    - es. `unsigned int`, `signed char`

I tipi di dato primitivi (6)

- **`limits.h`**
  - contiene macro che definiscono i massimi ed i minimi numeri rappresentabili per tutti i tipi
  - es. `INT_MIN, INT_MAX`
  - per utilizzarli includere il file `limits.h`

I tipi di dato primitivi (7)

- Conversioni fra tipi diversi :
  - sono automatiche in C
  - posso scrivere espressioni con operandi di tipo diverso

```
int a = 0; float b, r = 3.1;
b = a + r;
```

  - di solito viene convertito tutto nel tipo più accurato (le regole a p.115 nel testo KP)
  - conversioni esplicite

```
r = (float) a / b
```

I valori booleani

- Rappresentazione dei valori booleani in C :
  - non esiste il tipo `bool`!
  - sono rappresentati con `int`
  - 0 rappresenta FALSO, qualsiasi valore != 0 rappresenta VERO
    - es. 456 viene interpretato come VERO
    - es.

    ```
    if (1) printf("VERO");
    ```

    è un condizionale con condizione sempre vera, che quindi stampa sempre VERO sullo schermo

Gli operatori booleani

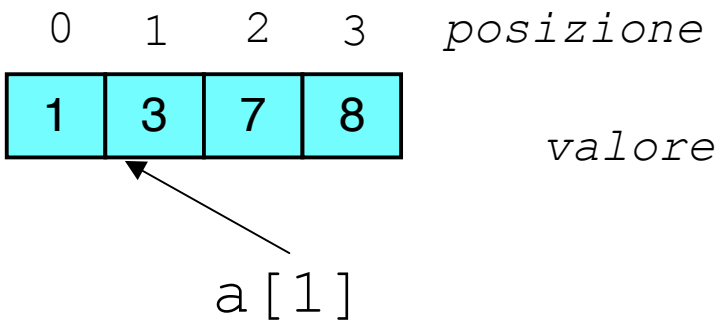
- && (and), || (or), ! (not)
  - valore : 0 se sono falsi e 1 se sono veri
  - sono rappresentati con `int`
- operatori bit a bit : & (and), | (or), ^ (xor), << (lshift), >> (rshift), ~ (complemento)
  - es.

```
1 << 3    // maschera con 1 in terza posizione
a & ~(1 << 3 )  // azzera il terzo bit
```

Gli array

- Aggregati di variabili dello stesso tipo :
  - dichiarazione (array statici):

```
int a[4];
```



  - gli indici partono da 0
  - non viene effettuato nessun controllo sugli indici nell'accesso : se cerco di accedere a `a[15]` il compilatore non dà errore ed in fase di esecuzione il risultato dipende dal contenuto della memoria adiacente ad `a`

Le stringhe

- Sono array di caratteri terminati da ´\0´
  - dichiarazione :

```
char parola[8]="mango";
```

  - permette di memorizzare una qualsiasi parola di al più 7 caratteri (l'ottavo serve per il terminatore)
  - è inizializzata con la parola "mango"
  - `string.h` fornisce tutte le più comuni funzioni per manipolare stringhe

Le strutture (record)

- Aggregati di variabili di tipo diverso
  - dichiarazione di una nuova struttura:

```
struct studente {
   char nome[40];
   char cognome[40];
   double voto;
};
```

  - dichiarazione di variabili di tipo struttura :

```
struct studente x;
struct studente classe[140];
```

Le strutture (record) (2)

- Ridenominazione tipi :

```
typedef tipo nome;
```

  - es :

```
typedef int Intero;
Intero nome[40];
```

  - es :

```
typedef char stringa40[41];
stringa40 nome;
```

Le strutture (record) (3)

- Ridenominazione tipi :

```
typedef struct {
        char nome[40];
        char cognome[40];
        double voto;
    } studente ;
```

  - così si può eliminare struct dalle dichiarazioni

```
studente x;
studente classe[140];
```

Le strutture (record) (4)

- Ridenominazione tipi :

```
typedef struct studente {
        char nome[40];
        char cognome[40];
        double voto;
    } studente ;
```

Opzionale (serve per i tipi ricorsivi)

  - così si può eliminare struct dalle dichiarazioni

```
studente x;
studente classe[140];
```

Le strutture (record) (5)

- Accesso ai campi :
  - tramite l'operatore punto (.)
  - es :

```
studente x;
studente classe[140];

classe[i].nome = x.nome;
```

I tipi enumerati

- Rappresentano insiemi finiti:
  - giorni: lunedi, martedi, …
  - mesi: gennaio, febbraio, …
  - bool: true, false
- Associano un identificatore (costante) ad ogni elemento dell'insieme
- Sono rappresentati con degli interi
  - ... ma il compilatore controlla che le funzioni siano chiamate con il tipo giusto (type checking)

I tipi enumerati (2)

- Esempio:

```
enum giorni {lun,mar,mer,gio,ven,sab,dom};

enum giorni x;        /* dichiarazione */
x = lun;
…
```

  - Si può evitare di ripetere `enum` con un `typedef`
  - I valori interi assegnati partono da 0 :
    - `lun == 0, mar == 1, mer == 2,…`

I tipi enumerati (3)

- Esempio:

```
typedef enum {true = 1, false = 0} bool;

bool trovato;        /* dichiarazione */
trovato = false;
...
```

  - Si possono scegliere i valori interi assegnati ad ogni elemento

Il linguaggio C

Funzioni

Funzioni C

- Le funzioni C sono l'unico modo per strutturare i programmi e non si possono annidare
- Non esistono le procedure
- Non esistono i metodi
- Dichiarazione di funzioni :
  - descrive il *prototipo* della funzione ovvero: nome della funzione, il tipo del valore restituito ed il tipo di tutti gli argomenti utilizzati

Funzioni C (2)

- Dichiarazione di funzioni :
  - es: prototipo della funzione di somma di due interi

```
int somma (int, int);
```

oppure

```
int somma (int x, int y);
```

x, y sono inutili, ma vengono convenzionalmente specificati per documentazione

Funzioni C (3)

- Definizione di funzioni :
  - la definizione di una funzione è divisa fra :
    - una intestazione (head) che fornisce il prototipo della funzione ai nomi per i parametri formali, ed
    - un corpo (body) che specifica le azioni da compiere
  - es:

```
int somma (int x, int y) {
   return (x + y);
}
```

intestazione

Funzioni C (4)

- Definizione di funzioni :
  - la definizione di una funzione è divisa fra
    - una intestazione (head) che fornisce il prototipo della funzione ai nomi per i parametri formali, ed
    - un corpo (body) che specifica le azioni da compiere
  - es:

```
int somma (int x, int y) {
   return (x + y);
}
```

corpo

Funzioni C (5)

- La dichiarazione o la definizione di una funzione deve sempre precedere il suo uso!
  - Tipica struttura di un programma C (unico file):

```
#direttive al preprocessore
dichiarazioni di variabili globali
(es. int k;)
dichiarazione di funzioni (prototipi)
(es. int somma (int x, int y);)

int main (…) {…}

definizione di funzioni
(es. int somma (int x, int y) {…})
```

Funzioni C (6)

- Scope delle variabili:
  - Il corpo di una funzione contiene le variabili locali
    - sono allocate sullo stack
    - sono accessibili solo a quella funzione
    - perdono il valore fra un'invocazione e l'altra

Funzioni C (7)

- Scope delle variabili (cont.) :
  - Le variabili globali sono variabili dichiarate al di fuori delle funzioni
    - sono accessibili all'interno di tutte le funzioni
    - sono allocate nell'area dati e vengono deallocate solo alla terminazione del programma
  - Le globali sono sconsigliate a meno di casi motivati!
    - devono essere sempre adeguatamente documentate

Funzioni C (8)

- Scope delle variabili (cont.) :
  - Le variabili statiche locali (`static`)
    - sono accessibili all'interno della funzione che le dichiara
    - mantengono il valore fra una invocazione e l'altra
- Tutti i parametri sono passati *per valore*
  - una copia viene messa sullo stack all'atto della chiamata

Esempio di funzione

```
#include <stdio.h>

int max (int a, int b);    /* dichiarazione */

int main (void){
  printf("Il massimo è %d \n", max(10,2));
  return 0;
}

int max (int a, int b) {
   return (a < b)? b : a ; /* definizione */
}
```

Esempio di funzione (2)

```
/* un esempio di var globale */
#include <stdio.h>

int max (int a, int b);    /* dichiarazione */
int k = 0;                 /* var globale */
int main (void){
  printf("Il massimo è %d \n", max(10,2));
  printf("Il valore di k è %d \n", k);
  return 0;
}

int max (int a, int b) {
   k = k + 1;                /* side effect */
   return (a < b)? b : a ; /* definizione */
}
```

Esempio di funzione (3)

```
/* un esempio di var statica */
#include <stdio.h>

int max (int a, int b);   /* dichiarazione */

int main (void){
  printf("Il massimo è %d \n", max(10,2));
  /* k non è più accessibile fuori da max */
  return 0;
}

int max (int a, int b) {
   static int k = 0;
   k++;
   return (a < b)? b : a ; /* definizione */
}
```