Verso l'utilizzo di un Microcontrollore

- Oggi iniziamo una serie di "lezioni" ed esperienze di laboratorio che ci porteranno alla programmazione e utilizzo di un microcontrollore cioe` di un computer in miniatura. Il cuore di questo piccolo computer e' il microprocessore.

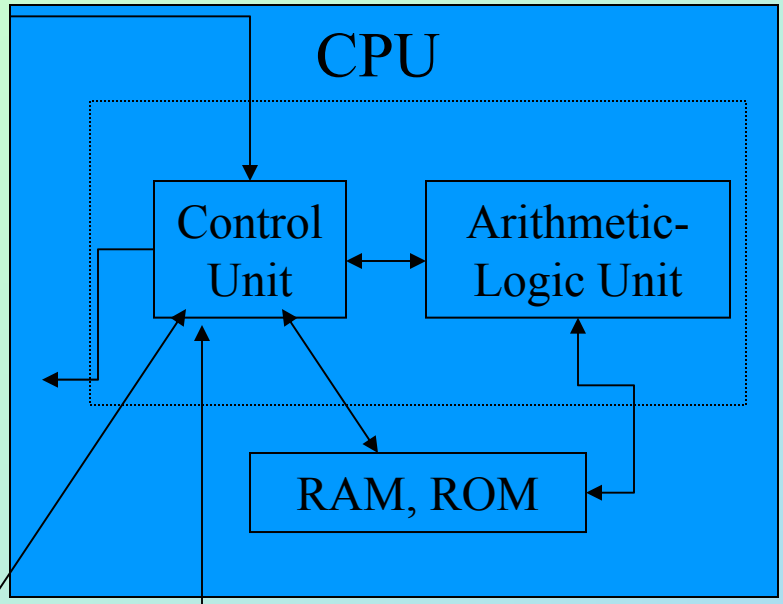
- Per poter arrivare a comprendere come utilizzare un microcontrollore abbiamo bisogno di qualche lezione ed esercitazione introduttiva per avere almeno un'idea di massima sulla struttura di un computer sia dal punto di vista Hardware (HW) che Software (SW).

  HW : "la plastica, gli integrati, l'alluminio..." tutto cio` che si puo` toccare (CPU, ROM, RAM, tastiera, mouse...).

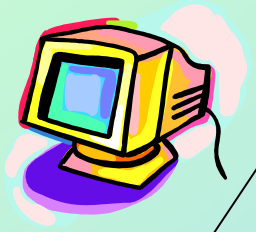
  SW : i programmi che fanno fare qualcosa al computer (sistema operativo, drivers per le periferiche, compilatori, programmi applicativi...).

Schema generale

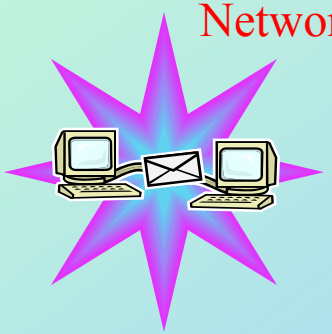
Input Devices



Output Devices



Network

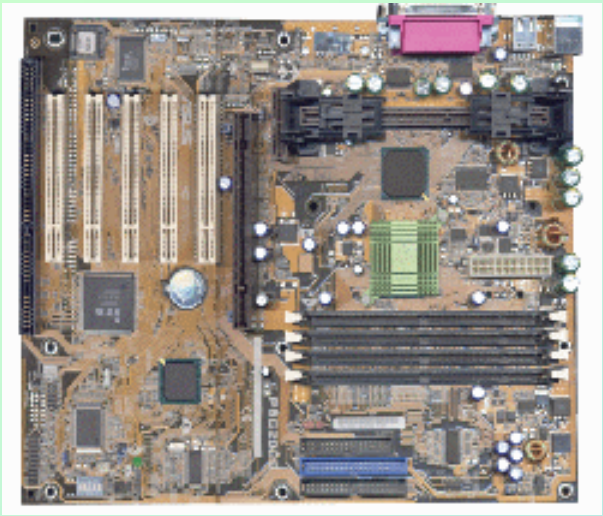


External Memory

- Componenti principali :
  - CPU (Central Processing Unit) e' il cervello del computer, esegue molto velocemente semplici istruzioni (bit shift, somme, copie di memoria) ed utilizza la RAM (Random Acess Memory) come memoria di lavoro.
  - Le comunicazioni con il computer avvengono con periferiche di input (tastiera, mouse) e periferiche di output (stampante e schermo).
  - I dati vengono immagazzinati in memoria primaria volatile e non (ROM, RAM) e secondaria non volatile (Hard disk, floppy, CD ...).
  - Il "mondo esterno" viene visto attraverso la rete.

- Parametri importanti : Tipo di CPU e velocita` (Pentium 3 - 800 MHz), dimensione di RAM (256, 512 Mbytes) e Hard disk (40 Gbyte) e periferiche ....

Motherboard e Bus



- La mother board e` la base sulla quale vengono collegate : CPU, RAM, ROM, Real Time Clock, la batteria tampone e i bus delle perferiche;
- i BUS permettono lo scambio di informazioni tra le varie parti del computer (sono rappresentati con le frecce nere nel disegno precedente);

• Il bus e' composto da piu` linee di dati che permettono di trasferire contemporanemente piu` bit (es.: 8, 16 o 32 bit); la velocita` con cui le informazioni vengono scambiate sui bus e la loro larghezza sono due dei parametri che definiscono la velocita` di esecuzione di un programma;
• Esistono bus e standard di comunicazioni stabiliti dai vari produttori (IBM XT 8 bit, IBM AT 16 bit, PCI 32 bit 33 o 66 MHz).

Collegamenti delle periferiche : le porte e i connettori

- Porta seriale (maschio a 9 o 25 pin) per collegare mouse, modem ... ;
- Porta parallela (femmina a 25 pin) per collegare ad esempio la stampante;
- Porta video (femmina a 15 pin su tre file) per il collegamento dello schermo;
- Porta PS/2 (femmina circolare) per il collegamento del mouse;
- Porta USB (Universal Serial Bus) nuovo tipo di connessione ad alta velocita` permette di collegare periferiche esterne in cascata (telecamera, mouse ...);
- Porte audio (connettori jack) casse audio.

- Per ogni porta e` definito il protocollo di comunicazione. Il protocollo di comunicazione deve specificare : velocita` di trasmissione, livelli di 0 e 1, logica di traduzione dei livelli, quali pin fanno cosa, comunicazione sincrona o asincrona ...

Il Software

• Sistema Operativo (SO) : e` un insieme di semplici istruzioni che servono per il funzionamento base del computer, e` memorizzato sul disco rigido e viene copiato sulla RAM al momento dell'accensione. Esistono vari tipi di SO : Windows, Unix, Linux, MacOs, DOS ... alcuni di questi liberi alcuni a pagamento ...

• Programmi applicativi : Word, Excell, StarOffice, Access, Telnet , compilatori ... programmi di livello alto che servono per le piu` svariate applicazioni;

• Programmi scritti da noi : sono una serie di istruzioni scritte in un linguaggio di programmazione :
- assembler cioe` il linguaggio della macchina;
- linguaggi di programmazione piu` "umani" sono fortran, C, C++, Java, LISP...

Introduzione al linguaggio di programmazione **C**

Caratteristiche generali

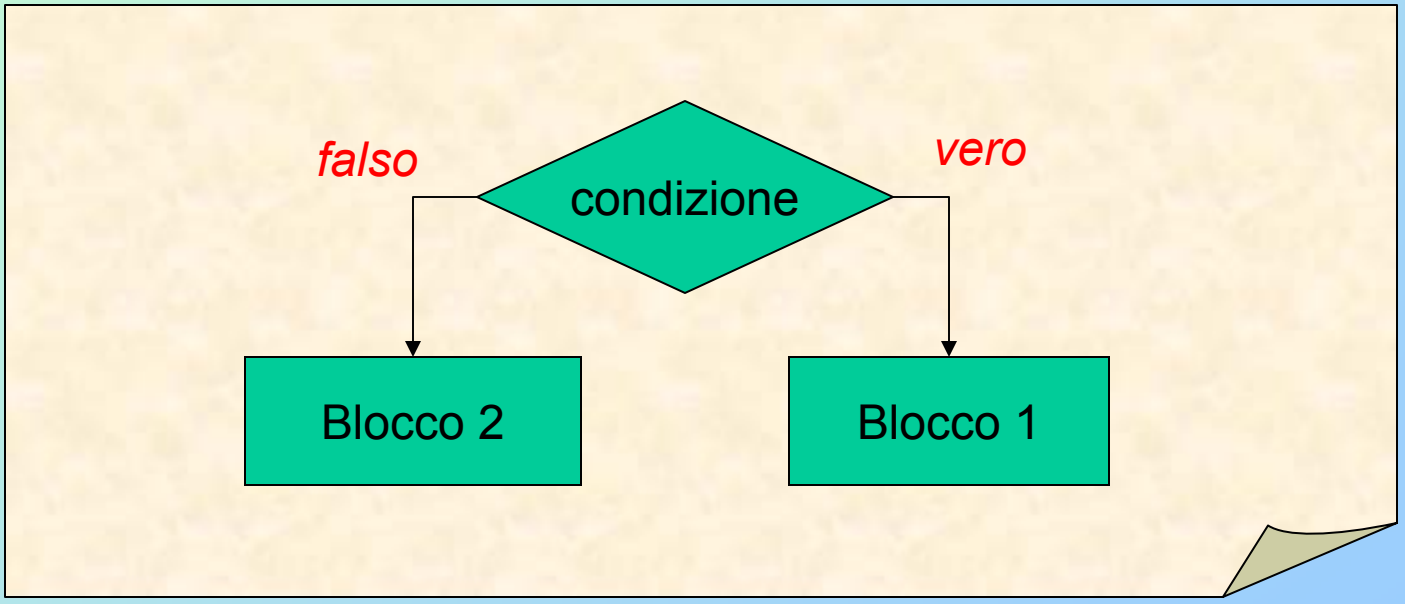
- Il C è un linguaggio: semplice (~30 keywords)
  strutturato →
  procedurale →
  tipizzato →

- Basso livello (vicino all'architettura del sistema, il cuore (kernel) del sistema operativo linux e` scritto in C)

- Grande flessibilità (adatto a piccoli e grandi progetti, adatto per implementare il SW per online ed offline ...)

...un linguaggio strutturato

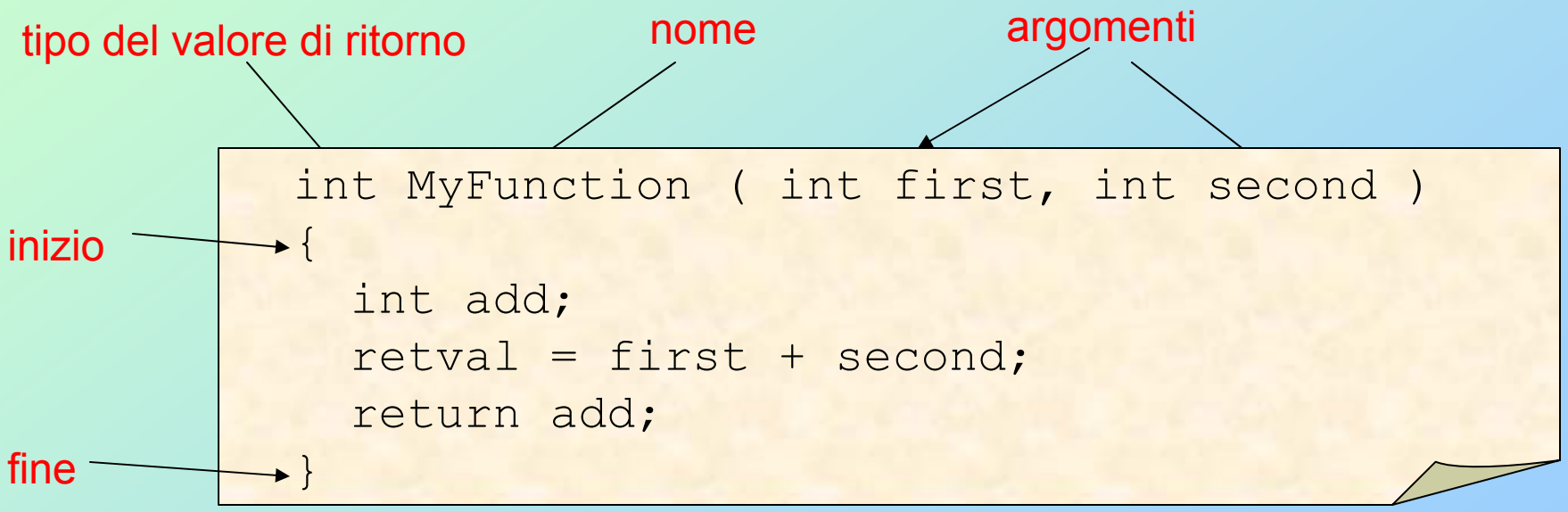
- Esistono una serie di istruzioni di controllo che permettono di realizzare *loops* ed esecuzioni condizionate di parte del programma



...un linguaggio procedurale

- Unità fondamentale: *funzione*

Mediante le funzioni si possono definire operazioni complesse a partire da istruzioni elementari



Una funzione speciale: main

main: e` la funzione da cui inizia l'esecuzione del programma. Il codice interno del C all'inizio del programma invoca questa funzione e riprende il controllo per finire il programma quando main finisce (ritorna).

```
int main()
{
  int ierr;
  .....
  return ierr;
}
```

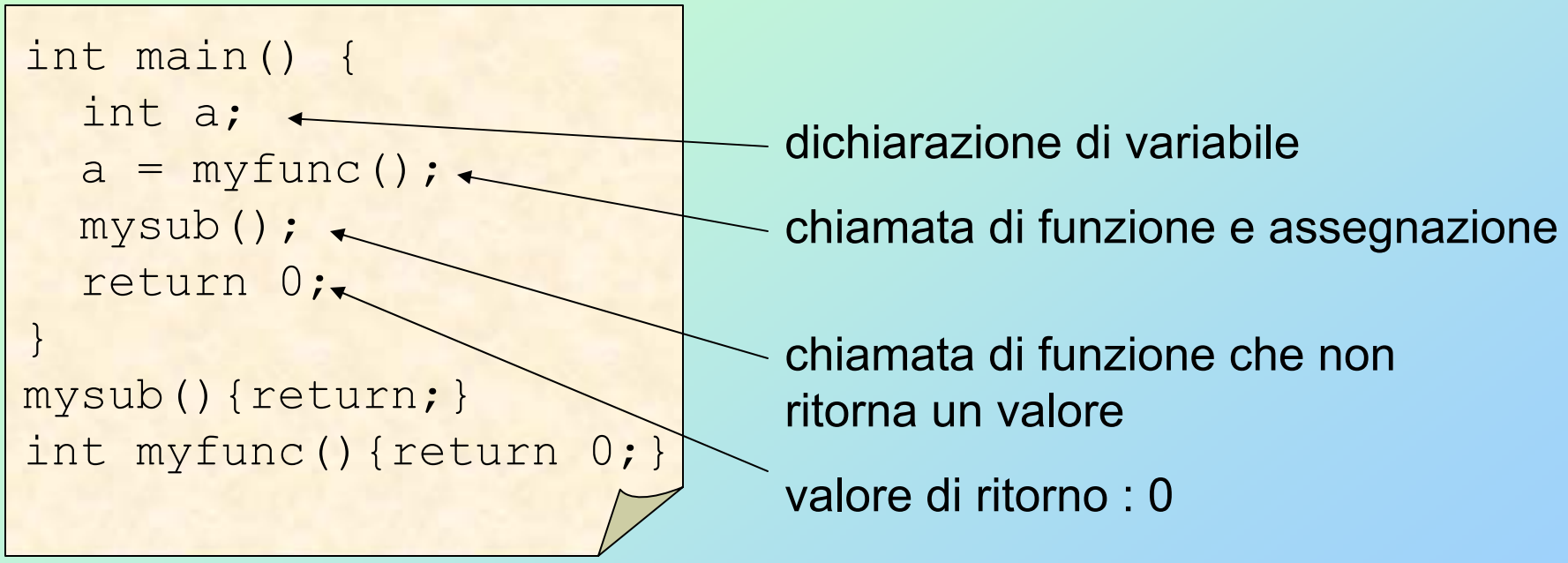
Formattazione del listato

- Il C è *"case sensitive"*: `MyFunction` ≠ `myfunction`
- La formattazione è libera MA
  - gli spazi possono essere inseriti SOLO tra un *"token"* e l'altro!
  - è necessario almeno uno spazio tra due tokens che non siano delimitatori (parentesi, " , ' , ….)

`main ( ) {}`	OK	`ma in(){}`	NO!
`int main(){` `return 0;}`	OK	`int main(){` `return0; }`	NO!

Un programma compilabile

```
int main() {
  int a;
  a = myfunc();
  mysub();
  return 0;
}
mysub(){return;}
int myfunc(){return 0;}
```



- Le parentesi graffe { … } racchiudono un gruppo di istruzioni da trattare come una singola azione:
  es.: corpo di una funzione, corpo di un ciclo, ecc..
- Il punto e virgola chiude tutte le istruzioni

L'esempio classico: "Hello World !"

Direttiva per il **preprocessore**: serve ad includere il file stdio.h che contiene (tra l'altro) la descrizione (prototipo) della funzione **printf** che appartiene alla libreria standard.

hallo.c:

```
#include <stdio.h>

int main() {
  printf("Hello World!\n");
  return 0;
}
```

Con lo stesso comando possiamo includere files scritti da noi, in questo caso si usa " .." anziche` < .. >

es.:

```
#include "MyFile.h"
```

L'esempio classico: "Hello World !"

Definizione della funzione speciale **main**: ritorna un intero (int) e non prende argomenti in input: ()

Tutto le linee di codice contenute tra { .....} sono le istruzione eseguite dalla funzione main.

hallo.c:

```
#include <stdio.h>

int main() {
  printf("Hello World!\n");
  return 0;
}
```

L'esempio classico: "Hello World !"

Chiamata alla funzione **printf** che stampa una stringa sul dispositivo di standard output: lo schermo.

La chiamata ad una funzione e` eseguita scrivendo:

nome-funzione(argomento1, argomento2,...);

nel nostro caso l'argomento di printf e` una serie di caratteri che sono racchiusi tra " ".

hallo.c:

```
#include <stdio.h>

int main() {
  printf("Hello World!\n");
  return 0;
}
```

\n e` un carattere speciale che indica alla funzione printf di andare a capo.

L'esempio classico: "Hello World !"

`return 0`: indica due cose, la prima e` che il controllo deve tornare al codice che ha chiamato main e la seconda che il valore che assume main e` 0.

hallo.c:

```
#include <stdio.h>

int main() {
  printf("Hello World!\n");
  return 0;
}
```

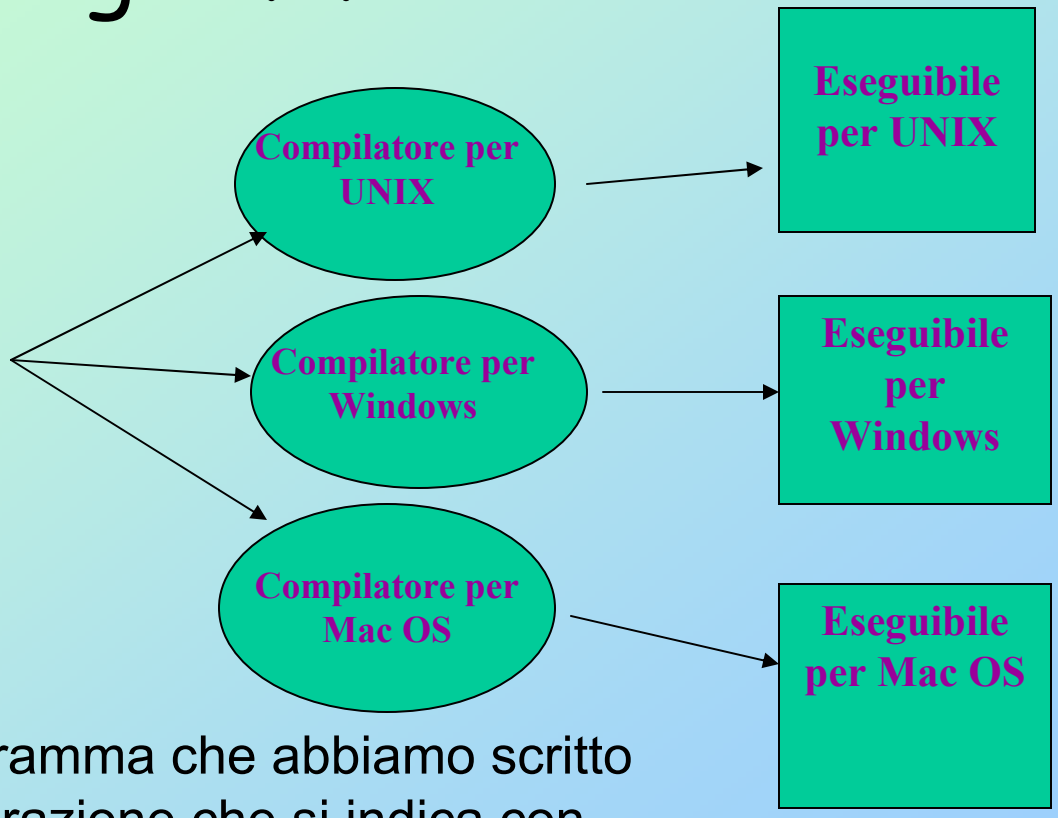
Esce dal codice di main (lascia lo scope di main)

Come fare funzionare un programma …

hallo.c:

```
#include <stdio.h>

int main() {
  printf("Hello World!\n");
  return 0;
}
```



- Per poter eseguire il programma che abbiamo scritto dobbiamo compiere un'operazione che si indica con **compilazione :** cioe' dobbiamo tradurlo in linguaggio macchina. Il compilatore dipende quindi dalla macchina su cui lavoriamo.

hallo.c:

```
#include <stdio.h>

int main() {
 printf("Hello World!\n");
 return 0;
}
```

Compilazione e altro...

- In effetti con “compilazione” vengono indicate una serie di operazioni, tutte automaticamente ed in modo invisibile all’utente, di cui pero` e` importante avere almeno un’idea di massima:

• il programma viene **preprocessato,** si eseguono cioe` i comandi che iniziano per #...), nel nostro caso viene incluso nel file hallo.c il file stdio.h

• il file di testo cosi` prodotto viene **compilato** e tradotto in linguaggio macchina (assembly) e viene prodotto un file che ha normalmente estensione “.obj”, nel nostro caso hallo.obj

• i file objects cosi` prodotti vengono infine passati al **linker**: si cercano e trasfericono nel programma le funzioni che si trovano nelle librerie es.: printf, o in altri file scritti da noi e viene prodotto l’eseguibile (hallo.exe)

Compilatore lcc-win32

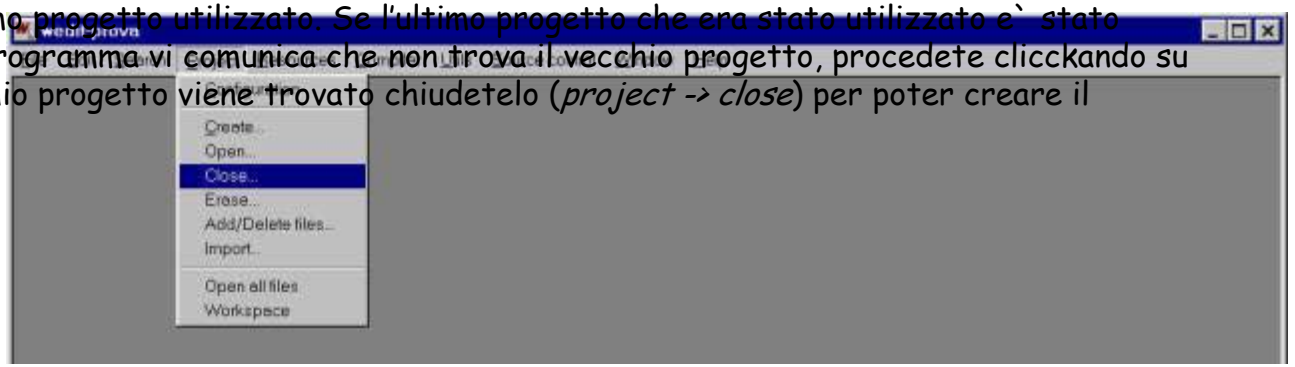
- Il compilatore che utilizzeremo e` “lcc-win32” : un compilatore per Linguaggio C per sistemi operativi Windows.
- Il compilatore richiede in input il codice C nomedelfile.c e produce in output un programma eseguibile nomedelfile
- il programma che utilizziamo in effetti contiene piu` del compilatore, offre anche un editor (programma per scrivere il testo del nostro programma), un modo per organizzare il nostro codice e varie altre cose che non utilizzeremo

Istruzioni per l'uso del compilatore

reazione di un progetto con il compilatore lcc-win32

Il primo esercizio in laboratorio oggi sara' di imparare ad utilizzare il compilatore. Seguite le istruzioni che trovate nelle fotocopie allegate in fondo al plico.

) Far partire il compilatore *start -> programmi -> lcc-win32 -> lcc-win32*. Il compilatore parte aprendo l'ultimo progetto utilizzato. Se l'ultimo progetto che era stato utilizzato e` stato cancellato il programma vi comunica che non trova il vecchio progetto, procedete clicckando su ok. Se il vecchio progetto viene trovato chiudetelo (*project -> close*) per poter creare il vostro.



Convenzioni

Una serie di convenzioni per non perdere tempo ...

- il nome di un file e` composto da due parte separate da un punto: hallo.c , sebbene ci sia completa liberta` nella definizione delle due parti (anzi possono in generale essere anche una sola o piu` di due) e` meglio seguire delle convenzioni che aiutano a velocizzare il lavoro: la prima parte del file deve avere un nome che ci ricordi il contenuto del file stesso, evitate gli spazi o caratteri strani nel nome; la seconda parte segue convenzioni che ormai sono diventate di dominio comune:

  .c per un file sorgente in C che contiene comandi (.cxx, .F, .f ....)

  .h per un file sorgente in C che contiene dichiarazioni

  .obj per un file compilato oggetto

  .exe per un file eseguibile (anche nulla come estensione indica un spesso un file eseguibile)

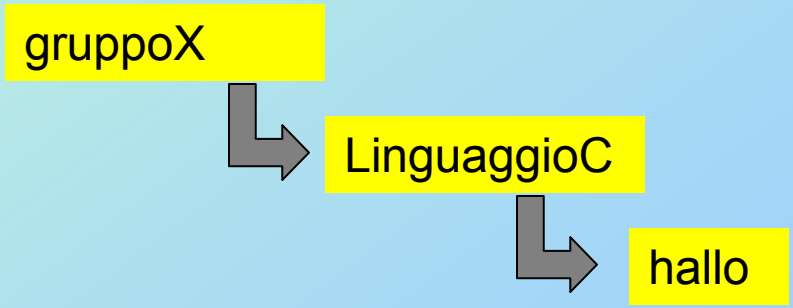
- quando utilizzate il programma lcc-win32 si crea un progetto per ogni esempio che facciamo. Il progetto conterra` il file .c e tutti gli altri file che il compilatore vi produce.

Convenzioni

•Create prima di tutto un folder sul desktop con il nome: gruppoX dove X e` la lettera che identifica il vostro gruppo

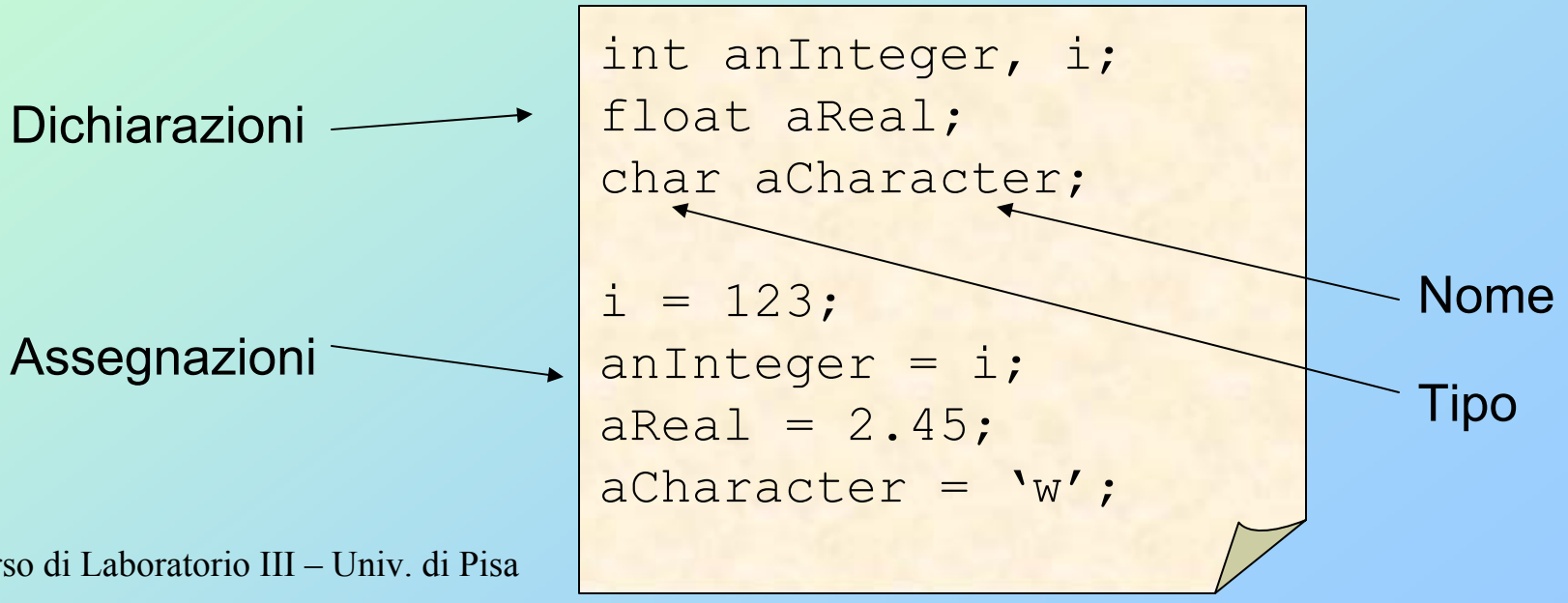
•all'interno di gruppoX create un ulteriore folder per ogni esperienza, ad esempio per l'esperienza di oggi create un folder LinguaggioC

•all'interno di questo folder create un folder per ogni nuovo progetto che fate ad esempio hallo



Le variabili

- Le variabili sono dei “contenitori” in memoria capaci di immagazzinare *tipi* diversi di dati (numeri interi, reali, caratteri, …)
- Costituiscono la struttura di dati piu’ semplice del linguaggio su cui agiscono operatori e funzioni



I tipi numerici del linguaggio C

Tipi interi:

	char	8
signed	**short** int	16
unsigned	**int**	32
	long int	32

Tipi in virgola mobile:

float	32
double	64
long double	80

- Tutti i tipi interi sono *signed* per default tranne char che è *unsigned*
- Non esistono i numeri complessi (ma esistono nelle librerie)
- Non esiste un tipo specifico per le stringhe di caratteri
- Quale e' il max numero rappresentabile con *short int* ?
- Un float sulla maggior parte delle macchine ha sei cifre significative

Definizioni e "scopo" delle variabili

- Tutte le variabili vanno dichiarate.

```
double globale;
int main() {
  float automatica;
  int i = 0;
  /* bla bla */
  return 0;
}
void f( double z ) {
  static int i;
  /* bla bla */
}
```

Variabile *globale*: è visibile da tutte le funzioni. Deve essere dichiarata **`extern`** dalle funzioni che la usano e che sono definite in altri files

Variabili *automatiche*: sono private, cioè visibili solo all'interno della funzione. Non sono inizializzate automaticamente. Nascono e muoiono ad ogni chiamata

Variabile *statica*: è analoga alla variabile automatica, ma il suo valore permane tra una chiamata e un'altra

Esempio sull'uso delle variabili

ese1.c

```
#include <stdio.h>

int globale = 0;
void a();

int main () {
  int locale;
  locale = 11;
  globale = 20;
  printf ("Variabile globale in main = %d\n", globale );
  printf ("Variabile locale in main = %d\n", locale );
  a();
  printf ("Variabile globale in main = %d\n", globale );
  printf ("Variabile locale in main = %d\n", locale );
  a();
  return 0;
}

void a () {
  int locale;
  int static localestatica = 100;
  localestatica = localestatica + 1;
  locale = 10;
  globale = 21;
  printf ("\nChiamata della funzione \n");
  printf ("\tVariabile globale nella funzione = %d\n", globale );
  printf ("\tVariabile locale nella funzione = %d\n", locale );
  printf ("\tVariabile locale statica nella funzione = %d\n\n", localestatica );
}
```

Le istruzioni di controllo (1/3)

- Cicli (Loop):

```
for(espr1;espr2;espr3) {
   ...
}
```

- Ciclo *for*:
  espr1 è valutata prima della 1ª iterazione
  espr2 è valutata prima di ogni iterazione; se è vera viene eseguito il blocco {…}
  espr3 è valutata al termine di ogni iterazione

Esempio :

```
for(i=0;i<10;i = i + 1)
   {
   printf("Hello !\n");
   printf("Lab III !\n");
   }
```

Le istruzioni di controllo (2/3)

- Cicli (Loop):

```
while ( espr ) {
   ...
}
```

- Ciclo *while*:
  espr è valutata prima di ogni iterazione;
  se è vera viene eseguito il blocco {…}

Esempio :

```
int i = 0;
  while(i < 10)
  {
  printf("Hello !\n");
  printf("Lab III !\n");
  i = i +1;
  }
```

Le istruzioni di controllo (3/3)

- Cicli (Loop):

```
do {
   ...
} while ( espr );
```

- variante *do ... while*:
  espr è valutata al terminie di ogni iterazione; se è vera si riesegue il blocco {...}

Esempio :

```
int i = 0;
 do {
   printf("Hello !\n");
   printf("Lab III !\n");
   i = i +1;
    }  while(i <10);
```

Esempio ciclo for

ese2.c

```
#include <stdio.h>

int main() {
   int i;
   for(i=0;i<10;i++)
   {
   printf("Hello !\n");
   printf("Lab III !\n");
   }
   return 0;
}
```

Esempio ciclo while

ese3.c

```
#include <stdio.h>

int main() {
   int i = 0;
   while(i <10)
   {
   printf("Hello !\n");
   printf("Lab III !\n");
   i = i +1;
   }
   return 0;
}
```

i++;

Esempio ciclo do-while

ese4.c

```
#include <stdio.h>

int main() {
  int i = 0;
  do {
    printf("Hello !\n");
    printf("Lab III !\n");
    i = i +1;
     }  while(i <10);

  return 0;
}
```

Esempio di uso di for (II)

ese5.c

```
#include <stdio.h>

/* Conversione da centimetri a pollici con una funzione */

float Convert ( float cm );

main() {

  float Pollici;
  float Centimetri = 0;
  float MaxCentimetri = 10;

  printf ( " \n  Centimetri    Pollici \n \n");

  for ( Centimetri=0; Centimetri <= MaxCentimetri; Centimetri+=0.5 ) {
    Pollici = Convert ( Centimetri );
    printf ("  %6f      %6f\n", Centimetri, Pollici );
  }
}

float Convert ( float cm ) {

  float inch = cm*2.54;
  return ( inch );

}
```

sintassi di printf()

```
#include <stdio.h>

/* Conversione da centimetri a pollici con una funzione */

float Convert ( float cm );

main() {

  float Pollici;
  float Centimetri = 0;
  float MaxCentimetri = 10;

  printf ( " \n  Centimetri    Pollici \n \n");

  for ( Centimetri=0; Centimetri <= MaxCentimetri; Centimetri+=0.5 ) {
    Pollici = Convert ( Centimetri );
    printf ("   %6f      %6f\n", Centimetri, Pollici );
  }
}

float Convert ( float cm ) {

  float inch = cm*2.54;
  return ( inch );

}
```

- Specifica il formato con cui stampare il primo dato che gli passiamo (Centimetri):
- % specifica l'inizio delle specifiche del formato
- 6 (opzionale) specifica il minimo nro di spazi da attribuire alla scrittura del numero
- f indica di stampare il nro come un float

altri formati: %c carattere, %d intero, %u unsigned integer, %x unsigned exadecimal ...

Le istruzioni di controllo (2/3)

- Maggior controllo sui cicli: istruzioni di salto
  - istruzione **`break;`**:
    causa l'immediata uscita dal più interno dei cicli in cui si trova
  - istruzione **`continue;`**:
    causa l'esecuzione immediata dell'istruzione di chiusura del ciclo, cui segue l'iterazione successiva del ciclo
  - istruzione **`goto`** *label*`;`:
    salto incondizionato all'etichetta *label*`:`; consente l'uscita da molti cicli annidati
  - istruzione **`return`** *espr_opz*`;`
    esce dalla funzione corrente ritornando opzionalmente il valore di *espr_opz*

Le istruzioni di controllo (3/3)

- Istruzioni di scelta:

```
if (espr1) { ... }
else { ... }
```

- *if*:
  se *espr1* è vera viene eseguito il blocco if. La clausola *else* è opzionale e si riferisce sempre alla condizione if immediatamente precedente.
  `if … else if … else if … else`

```
switch (espr) {
    case espr_cost1 : …
    break;
    case espr_cost2 : …
    default : …
}
```

- *switch*:
  l'esecuzione comincia al primo "case" che corrisponde ad *espr,* e continua fino al break o alla fine del blocco {…}

Esempio if-else if-else

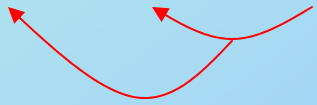
```
#include <stdio.h>

void main()  {
            int i;
            for(i=0;i<10;i++)

            {
                        if (i==0) printf("inizio \n");
                        else if (i < 3) printf("ora i vale %d\n",i);
                        else if (i==4) continue;
                        else if (i<7) printf("invece adesso i vale %d\n",i);
                        else break;
            }
}
```

Operatori (1/5)

- Assegnamento: operatore **=**
  - *lvalue* **=** *espr*;
  - significato: valuta l'espressione *espr*
    assegna *espr* ad *lvalue*
  - è associativo da destra a sinistra
  - per assegnare il valore 10 alle variabili a e b, si può scrivere: `a = b = 10;`



Operatori (2/5)

- Tra tipi numerici:
  - moltiplicazione e divisione: `*`, `/`
  - resto della divisione : `%` (es.: `20 % 9` fa `2`)
  - addizione sottrazione: `+`, `-`
  - Non esiste un operatore di elevamento a potenza (si puo` utilizzare una funzione di libreria)
- Booleiani
  - Equivalenza (e non) di espressioni logiche: `==`, `!=`
  - AND tra espressioni logiche: `&&`
  - OR tra espressioni logiche: `||`
  - NOT di espressione logica: `!`

Operatori (3/5)

- Operatori di manipolazione dei bit:
  - AND bit a bit: **`&`** **`es.: 0110 & 0100 = 0100`**
  - OR ed XOR bit a bit: **`|`**, **`^`** **`es.: 0110|1100 = 1110`**
  - spostamento a sinistra e a destra: **`<<`**, **`>>`**

    es.: a: 1000000010101101
    a << 2: 10 0000001010110100
  - NOT bit a bit (complemento a 1): **`~`**
- Operatori relazionali:
  - minore e maggiore: **`<`**, **`>`**
  - minore o uguale, maggiore o uguale : **`<=`**, **`>=`**

Esempio uso operatori bit a bit

```
#include <stdio.h>

void PrintBin ( int );

int main () {

  int i,j,k,l,m;

  i = 210;
  j = 1108;

  PrintBin ( i );
  printf ( "   %d\n",i );
  PrintBin ( j );
  printf ( "   %d\n",j );

  printf ( "AND\n" );
  k = i & j;
  PrintBin ( k );
  printf ( "   %d\n",k );

  printf ( "OR\n" );
  l = i | j;
  PrintBin ( l );
  printf ( "   %d\n",l );

  printf ( "\n" );
  PrintBin ( l >> 3 );
  printf ( "   >> 3  " );
  printf ( "\n" );
  printf ( "\n" );

  PrintBin ( l << 3 );
  printf ( "   << 3  " );
  printf ( "\n" );
  printf ( "\n" );

  PrintBin ( ~l );
  printf ( "   ~   " );
  printf ( "\n" );
  printf ( "\n" );


  return 0;
}

void PrintBin ( int val ) {
  int i;
  for ( i=16; i>=0; i-- ) {
    if ( val & ( 1 << i ) ) printf ("1");
    else printf ( "0" );
  }
}
```

ese6.c

Operatori (4/5)

- Espressioni di assegnamento in forma compatta:

  `a = (a + b);` si può anche scrivere

  `a += b;`

  - L'operatore `+=` si comporta come un operatore di assegnamento, e quindi *ritorna* il valore assegnato, es.:
    `c += a += b + 1;` significa
    `a = a + (b + 1); c = (c + a);`

- Analogamente abbiamo:

  `-=, *=, /=, %=, &=, ^=, |=, <<=, >>=`

Operatori (5/5)

- Operatori di incremento e decremento:
  - notazione prefissa:

  `++a;` incrementa `a` e ritorna il nuovo valore

  `--a;` decrementa `a` e ritorna il nuovo valore

  - notazione postfissa:

  `a++;` ritorna `a` e, dopo, lo incrementa

  `a--;` ritorna `a` e, dopo, lo decrementa
- Possono essere applicati a tutti i tipi interi (e ai puntatori)

Due operatori speciali

- casting:
  - **`( type )`** *espr;*
    forza l'espressione *espr* ad essere interpretata come fosse di tipo *type* es.:

```
double f;
f = ( double ) 10;
```

- sizeof:
  - ritorna la dimensione in bytes di un tipo

```
unsigned int size;
size = sizeof( float );
```

Vettori

- Rappresentano zone contigue di memoria accessibili attraverso un indice

- Dichiarazione

```
int array_of_int[100];
double vec[2] = {1.1, 2}
```

Dichiarazione di vettore di 100 interi

Dichiarazione con inizializzazione

Espressione costante

- Uso

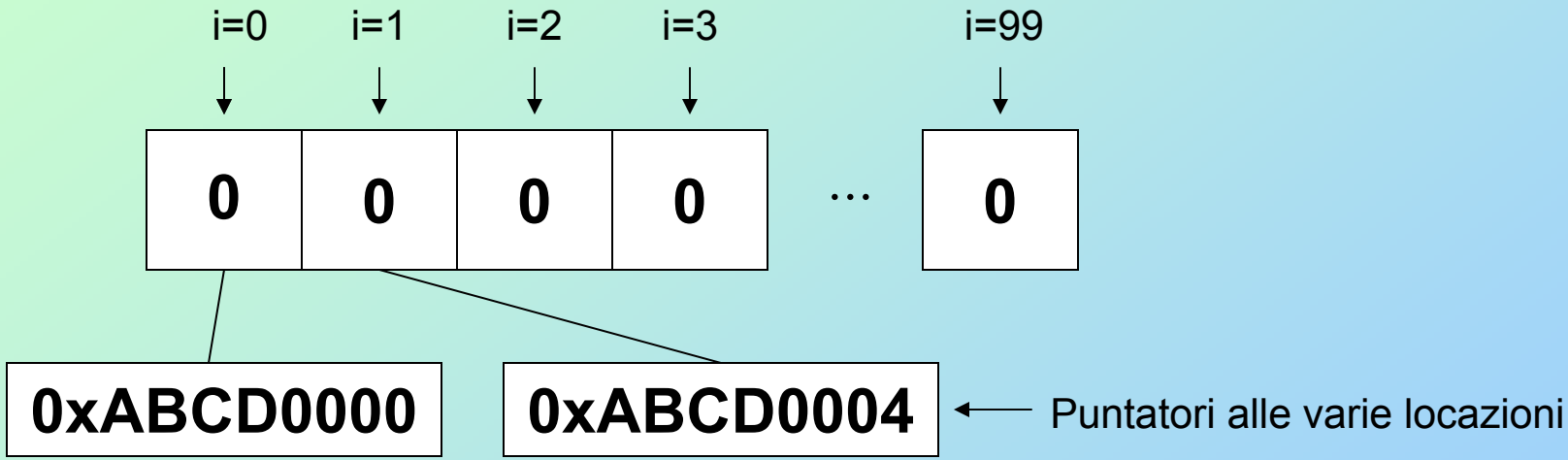
```
int i;
int array_of_int[100];
i = array_of_int[12];
```

Attenzione!

Nessun controllo sui limiti.

In questo caso l'indice del vettore deve essere < 100 altrimenti…. segmentation fault (core dumped)

Vettori: immagazzinamento in memoria



- I vettori sono memorizzati in locazioni di memoria consecutive
- L'indice del primo elemento è 0

Esempio uso di vettori

ese7.c

```
#include <stdio.h>

int main () {

  int i,j,imin, xmin, tmp;
  int v1[] = { 2,5,3,8,5,1,9,6,0,5 };

  for ( i=0; i<10; i++ ) {
    xmin = v1[i];
    for ( j=i; j<10; j++ ) {
      if ( v1[j] <= xmin ) {
                    imin = j;
                    xmin = v1[j];
      }
    }
    tmp = v1[i];
    v1[i] = v1[imin];
    v1[imin] = tmp;
  }

  for ( i=0; i<10; i++ ) {
    printf ( "v[%d] = %d\n", i, v1[i] );
  }

  return 0;
}
```

Le librerie standard

- Tutte le istruzioni del linguaggio C sono di basso livello.
- Le operazioni complesse (compreso l'I/O !) sono effettuate dalle funzioni delle "librerie standard" (abbiamo visto ad es.: **`printf(...)`**)
- Le librerie più comunemente usate sono:
  - Input e output: <stdio.h>
  - Manipolazione stringhe: <string.h>
  - Funzioni matematiche: <math.h>
  - Gestione della memoria ed altre utilità: <stdlib.h>

Per saperne di più

- Per la parte di HW e struttura di Microprocessore :
  - P. Horowitz W. Hill, The Art of electronics.
- Testo sacro:
  - *B.W. Kernighan, D.M. Ritchie*:
    The C programming language
- Altri libri:
  - *I. Pohl:* A Book on C
  - *M. Waite, S. Prata*: New C Primer Plus (2nd. Ed.)
- Materiale su www:
  - *T. Love*:
    ANSI C for Programmers on UNIX Systems
    (http://www.nd.edu/~cchen/teaching_C/node3.html)