Lezione 3 e 4

- Debugging e documentazione
- Regole di buona programmazione
- Schemi a blocchi e gestione del flusso
  (sequenza iterazione e scelta)
- Operatori ed input/output elementare
  (sulla dispensa *operatori.pdf*)

Fabio Scotti (2004-2009)

Laboratorio di programmazione
per la sicurezza

Valentina Ciriani (2005-2009)

Laboratorio di programmazione

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 3 e 4

Documentazione e debugging

Obiettivi :

- Conoscere dove reperire e come leggere la documentazione
- Imparare ad usare le funzionalità del programma debugger per controllare e correggere i programmi

Manuali online

- **Librerie del C Standard:**
  http://www.infosys.utas.edu.au/info/documentation/C/CStdLib.html.
- **Un buon libro di programmazione C o dispense del corso.**
- **www.google.com**

3

libc-2.3.2.chm

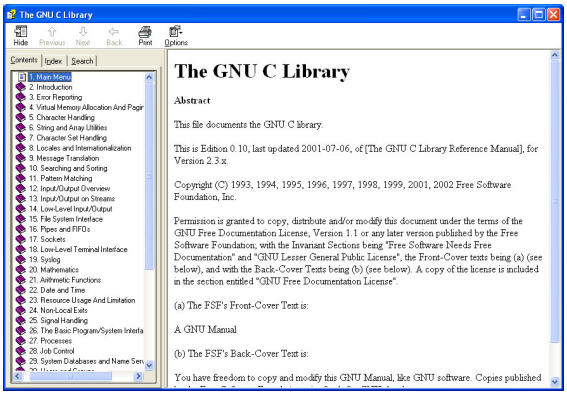
The GNU C Library

Libreria e commenti

http://www.infosys.utas.edu.au/info/documentation/C/

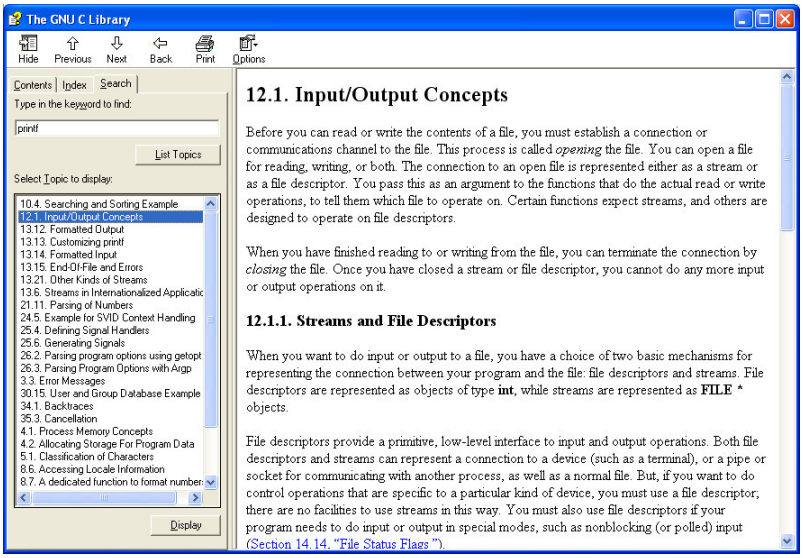
Help

http://www.dti.unimi.it/~fscotti/md_labsicurezza/allegati/libc-2.3.2.chm



4

Esempio di ricerca



5

Debugging (1)

- **Strumento importantissimo per:**
  - controllare il funzionamento del programma;
  - correggere eventuali errori.
- **Il compilatore lavora in modo diverso durante la fase di debug.**

  Tiene traccia di:
  - tutte le chiamate a funzione;
  - valore delle variabili;
  - istruzioni.

6

Debugging (2)

Le possibilità che offre un normale debugger sono:

- inserire dei **breakpoint** nel codice;
- procedere una istruzione alla volta (**stepping**);
- mostrare il **valore delle variabili**;
- **backtraking**
  - elenco delle funzioni che sono state chiamate prima di un breakpoint o una interruzione.
- **finestra della CPU**
  - monitorare i registri della CPU.

7

Abilitare il programma di debug

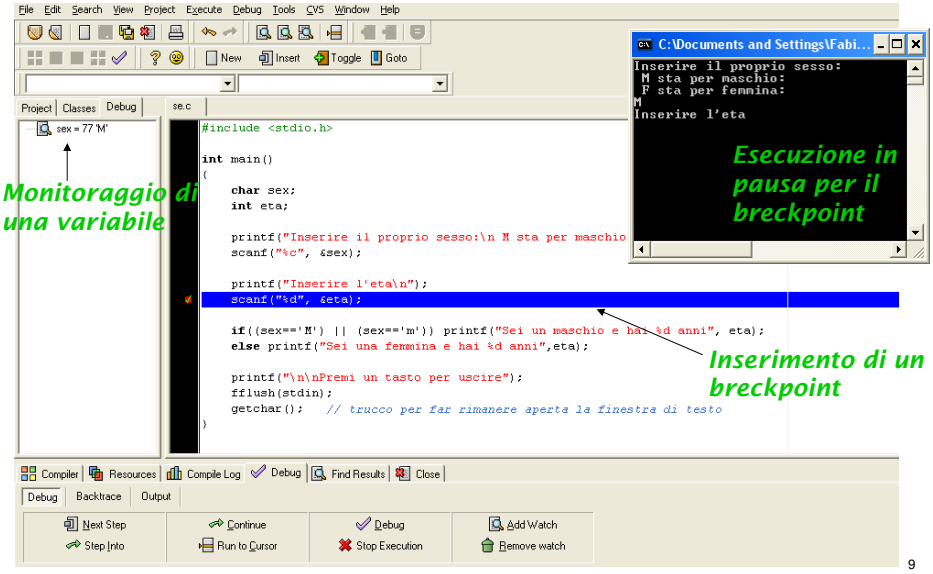
- **Nella nuova versione del software (dalla 4.9.9.1) il debugger si è già abilitato. Controllalo su**
  → DevC++ → Tools → Compiler Option → riquadro Settings → Linker: **`Generate debugging information:YES`**
- **Nelle versioni vecchie (e nel dubbio) meglio abilitare il debugger a mano per evitare noie.**
  → *DevC++ → Tools → Compiler Option*

  → Inserire il check
  `Add the following commands when calling compiler`

  → Aggiungere *esattamente* questi parametri
  -g3 -Wall

8

Esempio di debug



9

Fabio Scotti e Valentina Ciriani - Università degli Studi di Milano

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 3 e 4

Regole di buona programmazione

Obiettivi :

- Conoscere la differenza fra un programma scritto bene e male;
- Migliorare le stesura dei propri programmi per renderli più comprensibili, rivendibili e manuntentibili.

Regole di buona programmazione

KEEP IT SIMPLE !!!

La soluzione migliore è la più semplice!

Keep it simple !!

- **Scrivere codice facilmente comprensibile.**
- **Non usare costrutti complessi, troppo compatti, arzigogolati (lasciare la programmazione "offuscata" a chi ha tempo da perdere).**
  - The International Obfuscated C Code Contest
  - Radice quadrata: http://www.ioccc.org/2001/cheong.c
- **Un programma scritto in modo semplice è più portabile.**
- **E' anche più facilmente mantenibile (anche dopo anni può essere facilmente capito).**

Leggi e testa

- **L'uso di una funzione di una libreria che non si conosce richiede:**
  - un minuto di lettura del manuale;
  - la creazione di piccoli programmi per testarne il funzionamento sulla propria macchina.
- **Questo consente di risparmiare molto tempo e rende affidabili i propri "blocchetti di codice/funzioni di libreria".**

13

Nomi autoesplicativi

- **Dai nomi FURBI alle variabili ed alle funzioni.**
- **Se tutti i programmi sono pieni di variabili che si chiamano** i t temp ii iii a b c **la qualità di programmazione dell'autore sarebbe giustamente sottovalutata.**

Esempio di nome per una fuzione:

`CALcolologNaturale()` → **KO**

`calcolo_Logaritmo_Naturale().` → **OK**

NOMI AUTOESPLICATIVI !!

14

Indentazione e stile di formattazione

- **Indentare il proprio codice!**
- **Consente di trovare velocemente gli errori.**
- **Non cambiare continuamente lo stile di formattazione dei propri programmi, ma usare sempre lo stesso: ordinato e noto.**

Esempio :

```
somma=Primo+s;
```
→ **KO**

```
somma = addendo1 + addendo2;
```
→ **OK**

15

Dal "complesso" al "semplice"

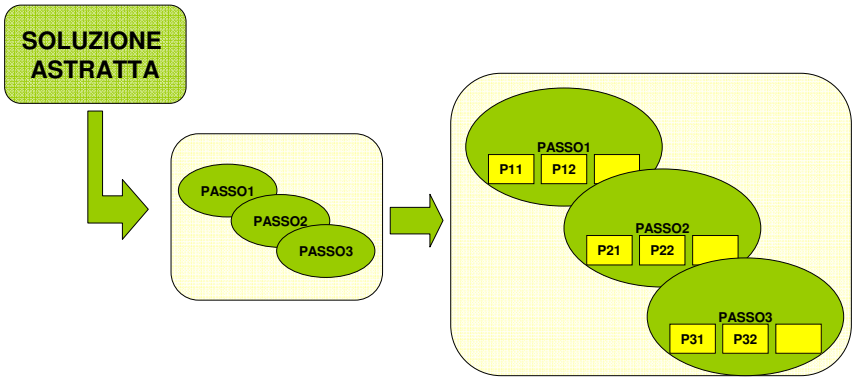
- **Se la regola più importante è "keep it simple!!", allora come è possibile applicarla se il problema da risolvere è complesso?**
- **Il segreto è imparare a SCOMPORRE IN PASSI ELEMENTARI la soluzione del problema in oggetto.**
- **La soluzione: una sequenza di operazioni che produce gli effetti desiderati lavorando sui dati a disposizione.**

PROGETTAZIONE TOP-DOWN

16

Progettazione TOP-DOWN

Una soluzione inizialmente deve essere pensata ad ALTO LIVELLO DI ASTRAZIONE, poi via via dettagliata e scomposta fino ad una sequenza di passi elementari:



17

Esempio di scomposizione (1)

PROBLEMA

- **Descrivere la sequenza di passi che un robot deve eseguire per lavare le finestre di una stanza.**
- **Il passo elementare che abbiamo a disposizione è il seguente: il robot posto davanti ad una finestra la lava.**
- **Il robot è dotato di sensori per localizzare una finestra in sua prossimità.**

18

Esempio di scomposizione (2)

Scomposizione di basso dettaglio.

1. **Il robot lava le finestre della parete che ha davanti.**

2. **Il robot si muove verso la parete successiva fino a che o ritorna davanti alla parete iniziale oppure ha un problema.**

19

Esempio di scomposizione (3)

Scomposizione ulteriore del punto 1.

1. **Il robot lava le finestre della parete che ha davanti.**
   - 1.1 Cerca la prima finestra.
   - 1.2 Lava la finestra.
   - 1.3 Si muove verso la successiva finestra; se non la trova passa al punto 2 altrimenti ritorna al punto 1.2.

2. **Il robot si muove verso la parete successiva fino a che o ritorna davanti alla parete iniziale oppure ha un problema.**

20

Esempio di scomposizione (4)

Scomposizione ulteriore del punto 1.1

1. Il robot lava le finestre della parete che ha davanti.

1.1 Cerca la prima finestra.

1.1.1 Si posiziona in alto a sinistra.

1.1.2 Scende lungo la colonna usando i sensori: se trova una finestra passa al punto 1.2.

1.1.3 Arrivato in fondo alla colonna si riporta in alto sulla seconda e reitera dal punto 1.1.2.

1.2 Lava la finestra.

….

21

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 3 e 4

Schemi a blocchi

Obiettivi :

- Imparare a passare da un problema complesso ad un insieme di problemi semplici attraverso decomposizioni;
- Tutta la programmazione sta in 3 parole: sequenza, scelta ed iterazione.

Sequenza, scelta e iterazione

Ogni algoritmo può essere formalizzato usando queste tre semplici strutture:

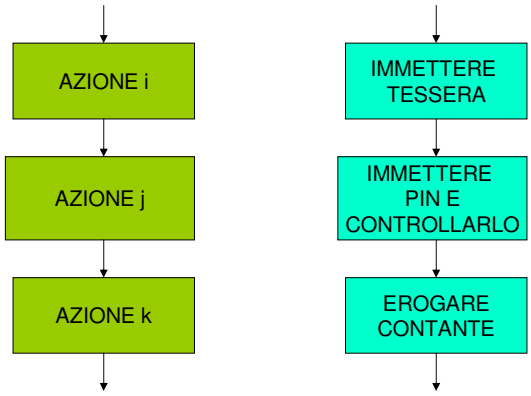
1. Sequenza
2. Scelta
3. Iterazione

Sequenza

- **La soluzione è corretta se le azioni vengono eseguite in una sequenza prefissata.**
- **Un esempio chiarificatore: IL BANCOMAT.**
  1. Inserire la tessera.
  2. Digitare il PIN e controllarlo.
  3. Erogare il contante.
- **Appare evidente che se invertiamo alcuni passi la soluzione non funziona come previsto.**

Sequenza in blocchi

L'idea della sequenza può essere efficacemente rappresentata con dei blocchi e delle frecce:



25

Scelta (1)

- **Una o più azioni che devono essere eseguite, in alternativa ad altre, in base al valore di verità di una specifica Proposizione Logica (o espressione logica).**
- **Esempio**

  Se PIN immesso dall'utente è uguale al PIN registrato in banca:

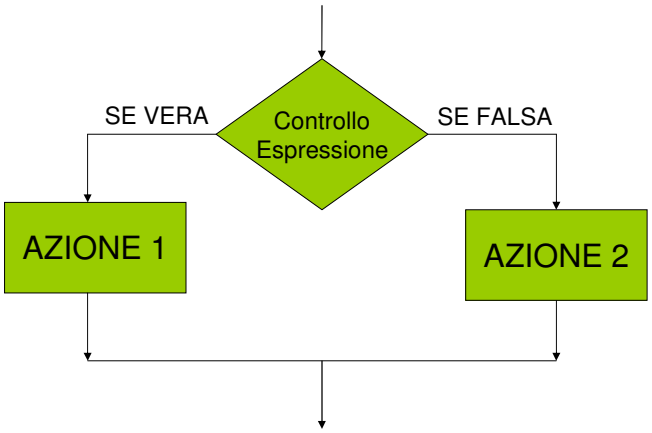
  - allora il BANCOMAT eroga i soldi;
  - altrimenti all'utente è richiesto di digitare un nuovo PIN.

- **La proposizione logica è: "il PIN immesso dall'utente è uguale al PIN registrato in banca". Essa può essere solo o vera (1) o falsa (2).**

26

Scelta (2)

L'idea della scelta può essere rappresentata con dei blocchi nel modo seguente:



27

Iterazione (1)

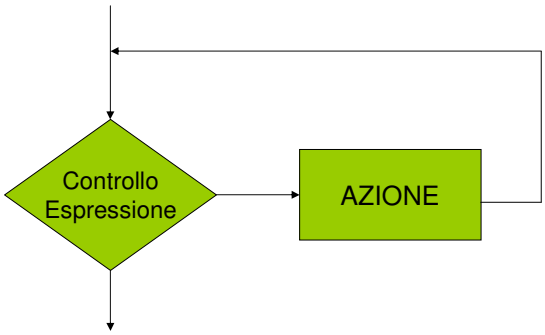
- **Una o più azioni che devono essere eseguite, zero o più volte, sulla base del valore di verità assunto, ogni volta, da una Proposizione Logica.**
- **Esempio**

  Finché il ragazzo inserisce il gettone il videogioco riparte.
- **La proposizione logica è : "*un gettone è stato inserito*".**

28

Iterazione (2)

L' idea della iterazione può essere rappresentata con dei blocchi nel modo seguente (non è il solo modo!).



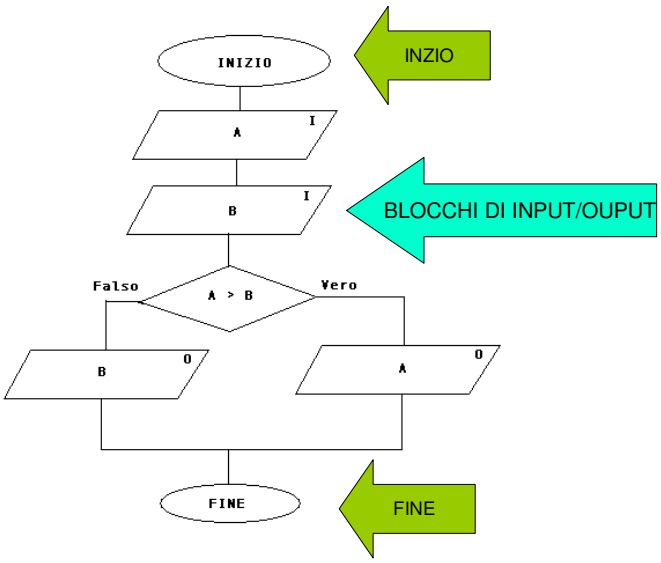
29

Composizione delle strutture

- **Si usano delle variabili e viene loro attribuito un nome simbolico per farvi riferimento (es: la variabile "conteggio").**
- **Le strutture si possono collegare fra loro.**
- **Una azione può essere:**
  - un assegnamento;
  - una operazione di input-output;
  - oppure essere sostituita da una struttura.

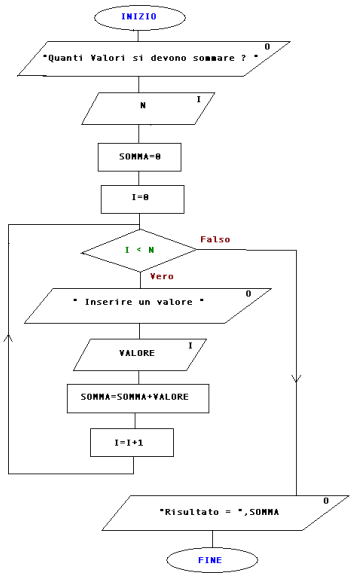
30

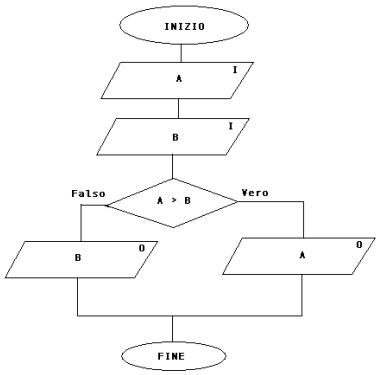
Altri dettagli



31

Esempi di connessioni





32

Esercizio di verifica

- **PROBLEMA**

  Disegnare lo schema a blocchi che descriva la seguente sequenza:

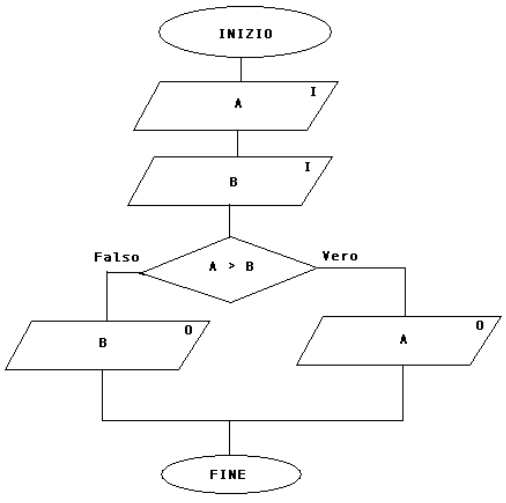
  1. vengano chiesti due numeri l'utente;
  2. venga stampato il numero maggiore.

- **COMMENTO ALLA SOLUZIONE**

  Impiegare due variabili A e B per descrivere i due numeri ed una struttura di scelta.

33

Soluzione dell'esercizio



34

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 3 e 4

Input/Output elementare

Obiettivi :

- Imparare ad usare comandi elementari per stampare a video ;
- Imparare leggere i valori immessi dall'utente dalla tastiera .

Esempio: `printf()` (1)

- **Problema**: ***Dichiarare una variabile `i` intera, assegnarvi un valore e stamparla a video.***

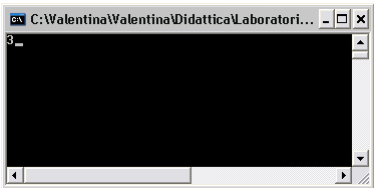
```c
#include <stdio.h>
int main()
{
    int i;
    i = 3;
    printf("%d", i);
    getchar();
}
```

1. **Dichiarazione della variabile `i` come variabile intera (`int`).**
2. **Assegnamento del valore `3` alla variabile `i`.**

Esempio: `printf()` (2)

3. **Chiamata della funzione `printf()` che stampa la variabile `i` a terminale.**
   1. Alla sinistra della virgola è presente una stringa chiamata stringa di controllo.
   2. La stringa di controllo definisce in che modo il contenuto della variabile `i` debba essere stampato.
   3. Ad esempio "`%d`" significa che `i` verrà stampato come un intero.

```
#include <stdio.h>
int main()
{
    int i;
    i = 3;
    printf("%d", i);
    getchar();
}
```



37

Output: `printf()`

- **Come già sappiamo, se scriviamo :**
  `printf("Hello WORD ");`
  compare a monitor `Hello WORD`
- **Se scriviamo:**
  `i=3;`
  `printf("Il valore contenuto e': %d", i)`
  compare a monitor `Il valore contenuto e': 3`

- **La funzione printf quindi:**
  - stampa la stringa di controllo,
  - sostituisce alle parti della stringa che iniziano con % i valori delle variabili passati alla destra della virgola.

38

Esempio: `scanf()` (1)

- **Problema**: ***Dichiarare una variabile `i` intera, leggere un valore da video e copiarlo in `i`.***

```
#include <stdio.h>
int main()
{
    int i;
    printf("immetti un numero intero :");
    scanf("%d", &i);
    printf("il valore memorizzato e' %d", i); // lo stampiamo per controllo
    fflush(stdin);
    getchar();
}
```

1. **Dichiarazione della variabile `i` come variabile intera (`int`).**
2. **Stampa di un messaggio per far capire all'utente che deve scrivere un intero (l'immissione si conclude con il tasto invio).**

39

Esempio: `scanf()` (2)

3. **Chiamata della funzione `scanf()` che copia il numero letto da tastiera nella variabile `i`.**
   1. Alla sinistra della virgola è scritto come deve essere letto il dato immesso dalla tastiera .
   2. Nel nostro caso l'indicazione "**`%d`**" indica che deve essere letto come un numero intero .
4. **Stampa il valore immesso per poterlo controllare usando una `printf ()`.**

```
#include <stdio.h>
int main()
{
    int i;
    printf("immetti un numero intero :");
    scanf("%d", &i);
    printf("il valore memorizzato e' %d", i);
    fflush(stdin);
    getchar();
}
```

40

Input: `scanf()` (1)

- **Le parti della stringa che iniziano con `%` indicano come deve essere letto l'input.**
- **Se l'utente digita 1024, il sistema potrebbe interpretare questo dato**
  - come il numero intero 1024
  - come la stringa composta dai caratteri '1', '0', '2' e '4'.
- **Indicando `%d` si impone che il dato debba essere letto un numero intero.**

41

Input: `scanf()` (2)

La funzione scanf quindi:

- legge il valore scritto dall'utente nel formato descritto dalla stringa che inizia con `%`,
- inserisce il valore letto nell'indirizzo indicato a destra della virgola (`&i` significa "all'indirizzo di `i`").

- **Attenzione: non è corretto scrivere**
  `scanf("%d", i);`
  senza &.

42

Fabio Scotti (2004-2009)
Laboratorio di programmazione per la sicurezza

Valentina Ciriani (2005-2009)
Laboratorio di programmazione

Lezione 3 e 4

Operatori, espressioni e istruzioni

Obiettivi :

- Conoscere i principali operatori del linguaggio C ;
- Imparare a distinguere espressioni e istruzioni.

Concetti che analizzeremo

OPERATORI, ESPRESSIONI E ISTRUZIONI

- **Definizione di**
  - Programma;
  - Istruzione;
  - Espressione;
  - Operatore e Operando.
- **Espressione di assegnamento.**
- **Espressioni ed operatori aritmetici.**
- **Operatori di incremento e decremento.**
- **Espressioni e operatori logici.**

Programma C

- **Il linguaggio C manipola i dati attraverso un insieme predefinito di**
  - operatori;
  - istruzioni.
- **Un programma C è una sequenza di istruzioni.**
  - Ad esempio `i=3;` è una istruzione.
  - Le istruzioni terminano sempre con un punto e virgola.

45

Istruzioni e espressioni

- **Una istruzione è una *espressione* terminata da un *punto e virgola*.**
  (Istruzione) ←→ (espressione;)
- **Una espressione è un insieme di**
  - variabili,
  - costanti e
  - richiami di funzione
  connessi da operatori aritmetici.
- **Ad esempio `a+b` oppure `x>3` sono espressioni.**

46

Operatori e operandi

- **L'operatore è un simbolo che indica al linguaggio di eseguire un'operazione, o un'azione, su uno o più operandi.**
- **In C gli operandi sono espressioni.**
- **Ad esempio nell'espressione `a+b` troviamo il segno `+` che è l'operatore, mentre le variabili `a` e `b` sono gli operandi.**

47

Variabili

- **La vita di una variabile:**
  - dichiarazione,
  - memorizzazione di un valore,
  - manipolazione.
- **Se si usa una variabile senza dichiararla il compilatore segnala un errore.**
- **La variabile è un'espressione.**

48

Assegnamento (1)

- **Memorizzazione di un valore in una variabile:**

```
nome_variabile = valore; //istruzione

nome_variabile = valore //espressione
```

- **A sinistra di = c'è la destinazione che può essere solo una variabile.**
- **A destra c'è la sorgente che può essere qualsiasi espressione che dia un valore.**

Assegnamento (2)

- **L'espressione assegnamento:**
  1. memorizza nella variabile a sinistra il valore dell'espressione a destra.
  2. restituisce il valore della variabile a sinistra.
- **Esempi**
  - `a = 5` è una espressione di assegnamento corretta (con valore **5**)
  - `a == 5` è una espressione non di assegnamento (con valore **diverso da 0** se `a` è uguale a 5, e valore **0** se `a` è diversa da 5)
  - `a = 5;` è un'istruzione corretta
  - `10 = a;` è un errore sintattico segnalato dal compilatore.

Espressioni ed operatori aritmetici

- **Una espressione aritmetica è**
  - un insieme di variabili, costanti e richiami di funzione connessi da operatori aritmetici.
- **Il risultato di un'espressione aritmetica è sempre un valore numerico.**
- **Operatori aritmetici (alto priorità massima):**

Negazione (- unario)		
Moltiplicazione (*)	**Divisione (/)**	**Modulo (%)**
Somma (+)	**Sottrazione (-)**	
Assegnamento (=)		

- **L'operatore % è il resto della divisione intera**
- **Anche l'assegnamento è un operatore.**

51

Parentesi

- **E' sempre meglio usare le parentesi tonde per evidenziare le priorità nell'espressione (per chi legge, non per il compilatore).**
- **Esempi:**

```
a = b = c = d + 1;  // NOOO.
a = ( b = (c = d + 1));  // si !!
a[x = n*2, y = n++] = (b = 27)+(c =(f =e-3));
        // ok hai usato le parentesi...
        // magari però è meglio usare più righe....
```

52

Operatori di incremento e decremento (1)

- **Operatori di incremento e decremento**
  - ++ è l'operatore di incremento (**`++i`** è equivalente all'espressione **`i = i + 1`**)
  - -- è l'operatore di decremento (**`i = i – 1`**)
- **Possono essere**
  - **Prefissi**: l'operazione viene eseguita prima di restituire il valore dell'espressione (es: **`++i`**).
  - **Postfissi**: l'operazione viene eseguita dopo aver restituito il valore dell'espressione (es: **`i++`**).

```
#include <stdio.h>
int main()
{
    int i,a,b;
    i = 15;
    a= ++i;
    b= i++;
    printf("%d, %d, %d", a, b, i);
    getchar();
}
```

53

Operatori di incremento e decremento (2)

- **Li useremo solo all'interno dei cicli `for`**
- **L'uso non razionale di questi operatori**
  - rende il codice illeggibile per le altre persone.
  - può dare comportamenti inaspettati dell'eseguibile.

```
#include <stdio.h>
int main()
{
    int i,a,b;
    i = 15;
    a= ++i;
    b= i++;
    printf("%d, %d, %d", a, b, i);
    getchar();
}
```


```
#include <stdio.h>
int main()
{
    int i;
    i = 15;
    printf("%d, %d, %d", ++i, i++, i);
    getchar();
}
```

54

Espressioni logiche

- **Un'espressione logica è un'espressione che genera come risultato un valore vero o falso**
- **Viene utilizzata dalle istruzioni di controllo del flusso di esecuzione (if, while, ecc.).**
- **La valutazione delle espressioni logiche può avere:**
  - un valore diverso da zero (interpretato come vero)
  - oppure un valore pari a zero (interpretato come falso).
- **Una *variabile* è un'*espressione logica***
  - se il suo contenuto è diverso da zero, allora l'espressione è vera,
  - altrimenti l'espressione è falsa .
  - Es: if (a) … while(casiDaAnalizzare) ….

55

Operatori logici

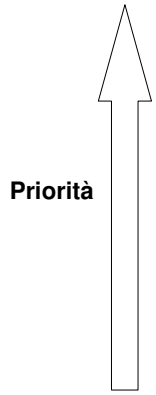
- **Le espressioni logiche possono contenere gli operatori relazionali usate per confrontare tra loro dei valori.**
- **Operatori relazionali (alto priorità massima):**

maggiore(>)	maggiore-uguale(>=)	minore(<)	minore-uguale (<=)
uguale (==)	diverso (!=)		

- **Operatori logici (alto priorità massima):**
  - NOT logico (!)
  - AND logico (&&)
  - OR logico (||)

56

Priorità tra operatori



! ++ -- - (unario)
* / %
+ - (binario)
> >= < <=
== !=
&&
||
=

57

Attenzione! Errori tipici!!

- **Una direttiva del precompilatore NON è un'istruzione (ad esempio `#include`).**
- **Le direttive non terminano con un punto e virgola come invece devono terminare tutte le istruzioni.**
- **Non confondere una uguaglianza con un assegnamento!**
  - `x == 3` NON E' `x = 3;`
  - `x == 3` è un espressione di uguaglianza che ritorna il valore 1 se x è uguale a 3, 0 viceversa. x non prende il valore 3!

58