LINGUAGGIO C

Breve Riassunto
Versione Completa

Indice.

Istruzioni C.

While.

Sintassi:

```
while ( <condition> ) <statement>;
```

Si usa per ripetere condizionatamente delle istruzioni.
<statement> è eseguito ripetutamente fino a quando <condition> è falso. Se non sono specificate condizioni, <condition> è assunto (true). Il test viene effettuato prima che <statement> sia eseguito. Così, se <condition> è (false) al primo passaggio, il ciclo non viene eseguito.

Esempio:

```
int i=0;

while (i<10)
{
    printf("\n%d",i);
    i++;
}
```

Conta da 0 a 9 stampando ongi numero. Se non fosse specificato "i++" il ciclo sarebbe infinito, perché (i<10) sarebbe sempre vera.

Do - While.

Sintassi:

```
do <statement> while ( <condition> );
```

Esegue <statement> ripetutamente finchè <condition> diventa (false).
Dal momento che il test viene fatto dopo che vengono eseguite le istruzioni, il ciclo viene eseguito almeno una volta.

Esempio:

```
do {
    printf ("Enter password: ");
    scanf("%s", password);
   } while (strcmp(password, checkword));
```

For.

Sintassi:

```
for ( [<initialization>] ; [<condition>] ; [<increment>] )  <statement>;
```

Implementa un ciclo iterativo.

<statement> viene eseguita ripetutamente finchè il valore di <condition> è (false).
Prima di entrare nel primo ciclo, <initialization> inizializza le variabili per il ciclo.
Dopo ogni iterazione del ciclo, <increments> incrementa il contatore del ciclo.
Tutte le parti sono opzionali. Se <condition> non è specificata, è assunta sempre (true).

Esempio:

```
int i;

for (i=0; i<10; i++)
{
    printf("\n%d",i);
}
```

Conta da 0 a 9 stampando ongi numero.

If.

Sintassi:

```
if ( <condition> )  <statement1>;

if ( <condition> )  <statement1>;
else  <statement2>;
```

Implementa istruzioni condizionali.
Si possono dichiarare variabili nell'espressione della condizione. Per esempio,

```
 if (int val = func(arg))
```

è una sintassi valida. La variabile val è nello scope dell'istruzione if e del blocco else se esiste.
La condizione deve essere di tipo booleano.
Qando <condition> è vera, <statement1> viene eseguito.
Se <condition> è falsa, viene eseguito <statement2>.
Il blocco else è opzionale, non ci possono essere istruzioni tra i blocchi if e else.

Esempio:

```
if (c==1)
{ blocco istruzioni 1 }
else
{ blocco istruzioni 2 }

oppure: if  (i==0) printf("Fine conteggio");
```

Switch.

Sintassi:

```
switch ( <switch variable> ) {
  case <constant expression> : <statement>; [break;]
    .
    .
    .
  default : <statement>;
}
```

L'istruzione switch passa il controllo a uno dei "case" determinati da <switch variable>.

Se nessun “case” corrisponde, viene eseguito quello di default.
Per terminare ogni blocco “case” si usa l’istruzione break;.

Esempio:

```
switch (s)
{
case 1 : /* se s==1 viene eseguito questo blocco */
   *** istruzioni ***
   break;
case 2 :
   *** istruzioni ***
   break;
...
default :
   *** istruzioni ***
}
```

Break, Continue, Goto.

Sintassi:

```
break;
```

L’istruzione break si usa all’interno di un ciclo per passare il controllo al ciclo più esterno e abbandonare il presente.

```
goto <identifier>;
```

L’istruzione goto trasferisce il controllo alla locazione del codice indicata dall’etichetta <identifier>.
Le etichette terminano sempre con i due punti “:”.

```
continue;
```

Passa il controllo alla fine di un ciclo; a questo punto viene eseguito la prossima iterazione del ciclo. Può essere utilizzato solo all’interno di cicli while, do e for.

Funzioni di libreria.

Standard I/O.

Funzioni fondamentali contenute nella libreria `stdio.h`:

printf.

Scrive una stringa formattata sullo schermo.
Dove Format string è una stringa generica oppure contenete *escape sequences* o *caratteri di conversione (*vedi).*

Sintassi:

```
printf("Format string", param);
```

Esempio:

```
s=5;
printf("\nIl risultato e': %d", s);
```

Scrive, andando prima a capo, il numero 5.

scanf.

Legge i dati forniti da tastiera (numeri, stringhe...).
Dove cstring è un *carattere di conversione* che deve essere specificato da solo, senza altri caratteri di tipo stringa, come avveniva nel BASIC con la funzione Input. La variabile v è quella in cui viene inserito il risultato dell'immissione, e deve essere preceduta dalla & (deve essere l'indirizzo di una variabile).

Sintassi:

```
scanf("cstring",&v);
```

Esempio:

```
printf("Inserisci numero: ");
scanf("%d",&a);
```

Richiede un numero e lo memorizza in a.

getchar.

Legge un carattere da tastiera. Se si immettono più caratteri in una linea, essi vengono memorizzati in un buffer e usati per le prossime chiamate di getchar. Il valore che viene restituito è il codice ASCII del carattere immesso, anche se questo è un numero. Viene anche retituito il carattere di invio '\n'.

Sintassi:

```
c=getchar();
```

Funzioni matamatiche.

Le funzioni matematiche più utili e comuni si trovano nella libreria `math.h`:

Nome	Sintassi	Funzione
`abs, labs, fabs`	`x=abs(y);`	valore assoluto di y int, long, float
`fmod`	`x=fmod(y, z);`	resto di divisione y e z float
`sqrt`	`x=sqrt(y);`	radice quadrata di y float
`pow`	`x=pow(y, z);`	y elevato a z float
`pow10`	`x=pow10(y);`	10 elevato a y float
`exp`	`x=exp(y);`	"e" elevato y float
`log`	`x=log(y);`	log base e di y float
`log10`	`x=log10(y);`	log base 10 di t float
`sin, cos, tan`	`x=sin(y);`	seno, coseno tangente di y float
`asin, acos, atan`	`x=asin(y);`	arcoseno... di y float
`atan2`	`x=atan2(y, z);`	arcotangente del rapporto y/z float

`sinh, cosh, tanh`	`x=sinh(y);`	seno iperbolico... di y float

Le funzioni che trattano float usano usano per massima compatibilità tipi double, la conversione è automatica.
Nella libreria sono anche definite delle costanti utili:

```
M_PI        Pi greco
M_PI_2      Pi greco / 2
M_PI_4      Pi greco /4
M_E         e (numero di Nepero)
M_SQRT2     radice quadrata di due
```

Si vedano inoltre le funzioni `atoi` e `atof` nella sezione "Stringhe".
ATTENZIONE: usando i float è molto pericoloso fare delle verifiche del tipo `(x==y)` poichè il risultato è "vero" solo se tutte le cifre decimali sono uguali, e questo è improbabile. E consigliabile scrivere invece:
`((x>=(y-d))&&(x<=(y+d)))` dove d è un piccolo margine di incertezza.

File.

Tutte le funzioni seguenti si trovano nella libreria `stdio.h` e, se non altrimenti specificato restituiscono EOF in caso di errore.

fopen.

Apre il file "nomefile" e restituisce un puntatore al file stesso.
La variabile "dati" è un puntatore al tipo FILE definito in stdio.h, e "mode" è una stringa che specifica la modalità di apertura del file: "r"=lettura, "w"=scrittura, "b"=binario, "a"=append, che si possono anche unire.

Sintassi:

```
dati=fopen("nomefile", mode);
```

Esempio:

```
FILE *in;
in=fopen("c:\suono.wav", "rb");
```

fclose.

Chiude un file precedentemente aperto restituisce EOF se l'operazione è fallita.

Sintassi:

```
int fclose(FILE *fp);
```

Esempio:

```
FILE *in;
...
flose(in);
```

feof.

Controlla se è stata raggiunta la fine del file nella precedente operazione di I/O. Restituisce un valore di verità.

Sintassi:

```
int feof(FILE *fp);
```

fprintf / fscanf.

Funzionano come printf e scanf ma scrivono su un file i dati formattati.

Sintassi:

```
int fprintf(FILE *fp, format string, param);
int fscanf(FILE *fp, format string, &param);
```

getc / putc.

Leggono e scrivono un carattere su un file. Il cursore nel file viene spostato in avanti ogni volta che si usano.

Sintassi:

```
int getc(int c, FILE *fp);
int putc(int c, FILE *fp);
```

fgets / fputs.

Leggono e scrivono stringhe all'interno di un file.
La prima legge n-1 caratteri dal file e li memorizza in un'area di memoria suntata sa s e inserisce il terminatore di stringa alla fine. La lettura viene interrotta in caso di incontro di \n (che viene restituita dalla funzione) o fine file.

Sintassi:

```
char *fgets(char *s, int n, FILE *fp);
int *fputs(char *s, FILE *fp);
```

fread / fwrite.

Leggono e strivono blocchi di dati (anche strutture) di dimensione "dimelemento", scrivendo o leggendone "numelementi" e immagazzinandoli nella variabile puntata da ptr.
Restituiscono il numero di byte letto o scritto.

Sintassi:

```
int fread(void *ptr, dimelemento, numelementi, FILE *fp);
int fwrite(void *ptr, dimelemento, numelementi, FILE *fp);
```

Esempio:

```
typedef struct
{
   int    campo1;
   float  campo2;
} dato;

dato rec;
FILE *fileout;

...
fwrite(&rec, sizeof(dato), 1, fileout);
```

fseek.

Sposta il cursore all'interno di un file ad un punto specificato, contato dall'inizio del file (ref=SEEK_SET), dalla posizione del cursore (SEEK_CUR) o dalla fine (SEEK_END).

Sintassi:

```
int fseek(FILE *fp, long offset, int ref);
```

ftell.

Restituisce la posizione del cursore all'interno di un file.

Sintassi:

```
long ftell(FILE *fp);
```

Memoria.

Funzioni della libreria `stdlib.h`

malloc.

La funzione malloc alloca un blocco di bytes lungo "size" dalla memoria heap. Serve ad un programma per richiedere memoria esattamente quando serve e della quantità necessitata.
Restituisce un puntatore al nuovo blocco, se esso è stato allocato, altrimenti (mancanza di memoria) restituisce un puntatore a NULL.

Sintassi:

```
void *malloc(size);
```

Esempio:

```
Typedef struct
{
   int dato1;
   float dato2;
   char stringa[40];
```

```
} mia_struttura;

mia_struttura *ptr; /* ptr è un puntatore ad una struttura mia_struttura */

ptr=malloc(sizeof(mia_struttura));
```

Alloca un blocco di memoria delle dimensioni di mia_struttura e restituisce un puntatore ad essa.

free.

Dealloca un blocco di memoria precedentemente allocato, rendendolo disponibile al sistema.
Richiede un puntatore di qualsiasi tipo ad un blocco, non restituisce niente.

Sintassi:

```
void free(void *block);
```

Stringhe.

Le stringhe sono degli array di caratteri che terminano con il carattere speciale '`\0`' detto terminatore di stringa.
Esse possono essere assegnate ad un array nel seguente modo:

```
char stringa[32]="Quasta e' una stringa";
```

Il terminatore viene inserito automaticamente. E' necessario che l'array abbia una dimensione adeguata.
Un altro tipo di assegnazione è quello con scanf:

```
char parola[20];

scanf("%s", parola);
```

Valgono le stesse specificazioni del caso precedente.
Alcune funzioni contenute nella libreria "`string.h`" sono:

strlen.

Restituisce la lunghezza (numero di caratteri) di una stringa, escluso il terminatore. Richiede il passaggio di un puntatore alla stringa stessa, nel caso di un array si usa il suo nome senza parentesi.

Sintassi:

```
unsigned int strlen(char *s);
```

strcmp.

Paragona due stringhe passate per indirizzo (*vedi* strlen) scandendo i caratteri di tutte e due, fino a quando non incontra una differenza o la fine della stringa. Restituisce un numero che è zero solo nel caso di stringhe identiche, può essere >0 o <0 secondo le differenze ASCII.

Sintassi:

```
int strcmp(char *s1, char *s2);
```

strcpy.

Copia una stringa in un'altra fino a quando non è stato copiato il terminatore della stringa sorgente. Restituisce un puntatore alla stringa di destinazione.

Sintassi:

```
char *strcpy(char *dest, char *sorg);
```

strcat.

Concatena due stringhe, copiando la stringa sorgente alla fine di quella di destinazione. Restituisce un puntatore a quella di destinazione.

Sintassi:

```
char *strcat(char *dest, const char *src);
```

atoi, atof.

Queste due funzioni convertono una stringa in un valore numerico, intero la prima e float la seconda, e possono risultare particolarmente utili in molte occasioni. La prima (atoi) si trova nella libreria `stdlib.h`, la seconda nella `math.h`:

Sintassi:

```
int atoi(chat *str);
double atof(char *str);
```

Le stringhe usate devono rispondere al seguente formato generico:

- per atoi: [spazi][segno][numeri]
- per atof: [spazi][segno][numeri][.][numeri][E][segno][numeri]

Ognuna delle parti tra parentesi è opzionale. La scansine della stringa termina al primo elemento non riconosciuto o alla fine della stringa, atof riconosce inoltre +INF e -INF per gli infiniti, e +NAN, -NAN per i non-numeri.

Dichiarazioni.

Tipi e strutture.

Tipi fondamentali.

16-bit data types, sizes, and ranges:

Type	Size (bits)	Range	Sample applications
unsigned char	8	0 to 255	Small numbers and full PC character set
char	8	-128 to 127	Very small numbers and ASCII characters
enum	16	-32,768 to 32,767	Ordered sets of values

unsigned int	16	0 to 65,535	Larger numbers and loops
short int	16	-32,768 to 32,767	Counting, small numbers, loop control
int	16	-32,768 to 32,767	Counting, small numbers, loop control
unsigned long	32	0 to 4,294,967,295	Astronomical distances
long	32	-2,147,483,648 to 2,147,483,647	Large numbers, populations
float	32	3.4 ^ 10-38 to 3.4 ^ 1038	Scientific (7-digit precision)
double	64	1.7 ^ 10-308 to 1.7 ^ 10308	Scientific (15-digit precision)
long double	80	3.4 ^ 10-4932 to 1.1 ^ 104932	Financial (18-digit precision)
near pointer	16	Not applicable	Manipulating memory addresses
far pointer	32	Not applicable	Manipulating addresses outside current segment

Sintassi della dichiarazione:

```
tipo nomevariabile;                 /* dichiarazione singola /*
tipo nomevar1, nomevar2, nomevarn;  /* diciarazione multipla */
tipo nomevar=volare;                /* dichiarazione con assegnazione */
```

Definizione di tipi.

Nuovi tipi di variabili possono essere costruiti usando l'istruzione "tipedef".

Sintassi:

```
typedef <type definition> <identifier>;
```

Assegna il nome <identifier> al tipo <type definition>.

Esempio:

```
typedef int tipo1; /* il tipo "tipo1" che non è una variabile ma un tipo */
tipo1 dato; /* dato è una variabile di tipo "tipo1" cioè int */
```

Tipi strutturati.

Costruttore array:

```
typedef char parola[32];
parola parola1; /* "parola1" è un array di 32 caratteri (da 0 a 31) */
```

Tipi enumerativi:

```
typedef enum {nome1, nome2, ...} nometipo;
nometipo nomevar;
```

I volari di "nomevar" anziché numeri potranno essere, solo nel codice, nome1, nome2 etc. Il compilatore gestice questi dati internamente come numeri int, quindi nome1 sarà uguale a 0 e così via.

Esempio:

```
typedef enum {false, true} boolean;
boolean flag;
```

Nel codice si potranno usare espressioni come questa:

```
if (flag=true)...
```

Dove comunque al posto di “true” si può usare indifferentemente 1 (0 per false), anche un printf di “flag” darebbe in uscita soltanto 0 o 1.

Strutture:

```
typedef struct <struct name> {
  <type> <variable-name>;
     .
     .
     .
} <structured variable>;
```

Costruisce un tipo strutturato, con:
<struct name> nome (opzionale) che identifica la struttura stessa;
<varible-name> identificatore dei singoli campi;
<structured varible> nome vero e proprio (opzionale) del tipo strutturato.
Se si usa soltanto <struct name> il tipo deve essere sempre indicato nel codice con <struct name> preceduto da “struct”.

Esempio:

```
typedef struct
{
   char nome[50];
   char cognome[60];
   int anni;
   float altezza;
} persona;

persona alunno; /* "alunno" è una variabile di tipo persona */

alunno.anni=20;
alunno.altezza=1.75;
```

Il punto “.” è utilizzato per selezionare il campo, sia in input che in output. Un campo di una struttura può essere a sua volta un’altra struttura (es.: esame.data.mese):

Macro e costanti.

Le macro servono a sostituire identificatori definiti con istruzioni, pezzi di codice o qualsiasi altra cosa. Il compilatore, nella fase iniziale di compilazione opera automaticamente la sostituzione. Questo può essere utile quando una determinata cosa viene utilizzata in molte parti del programma, e può sorgere la necessità di cambiarle, utilizzando le macro questo viene fatto automaticamente. E’ consigliabile inserire le definizioni delle macro all’inizio del codice. La sintassi è la seguente:

```
#define macro_identifier <token_sequence>
```

Esempio:

```
#define MAXLEN 1000
...
char nome[MAXLEN]; /* definizione di un'array */
...
if (n>MAXLEN)...
```

Stabilisce il nomero massimo di elementi di un'array.
Oppure:

Macro utili:

```
#define SWAP(a,b) tempr=(a); (a) = (b); (b) = tempr
#define max(a,b) ((a) >  (b) ? (a) : (b))
#define min(a,b) ((a) <= (b) ? (a) : (b))
```

Le costanti sono delle variabili definite in testa al programma e che non possono essere modificate in seguito. La sintassi è:

```
const <tipo> <nome>=<numero>;

const float gamma=2.1;
const int maxl=2332;
```

Operazioni / operatori.

Operatori unari.

Puntatori.

I puntatori sono delle variabili che contengono l'indirizzo di memoria di un'altra variabile, attraverso essi è possibile modificare fisicamente una variabile o leggerne il contenuto. In questo modo ogni procedura o funzione può accedere alle variabili interessate e leggerle o modificarle, senza che esse siano tuttavia visibili da esse, si evita così di utilizzare le variabili globali. Gli operatori interessati sono:

&	referenziamento;
*	dereferenziamento;
->	selezione diretta.

Facendo precedere il nome di una variabile dal simbolo "&" l'espressione avrà il valore dell'indirizzo della variabile e non del suo contenuto. Usando "*" davanti una variabile o un'espressione che indica un indirizzo, si otterrà come risultato la lettura del contenuto dell'indirizzo dato.
Quindi:

```
int dato; /* dato è un intero */
int *punt; /* punt è un puntatore ad un intero */

&dato       è un'indirizzo
punt=&dato  punt adesso contiene l'indirizzo di "dato"
*punt       è un intero, cioè "dato" stesso

dato=punt   è un errore, dato ora contiene il suo indirizzo!!
punt=dato   è un errore, punt ora punta ad un indirizzo errato.
```

L'operatore "->" si utilizza quando la variabile puntata è una struttura, e serve per selezionare in forma compatta il campo interessato:

```
typedef struct
```

```
{
   int dato1;
   long dato2;
} elem;

elem *punt; /* puntatore ad una struttura "elem" */
elem numero;

punt=&numero; /* assegna a "punt" l'indirizzo di "numero" */
(*punt).dato1 /* forma espansa */
punt->dato1 /* forma compatta, l'espressione è un valore */
&(punt->dato1) /* indirizzo del campo */
```

Gli **array** sono un particolare tipo di puntatori, il primo elemento infatti è un puntatore all'inizio del blocco di memoria occupato dai dati, quindi la sintassi corretta per il loro passaggio a funzioni è il seguente:

```
int q[32]; /* array che contiene 32 elementi (da 0 a 31) */
int *punt; /* puntatore ad interi */

q[3]=n; /* assegnazione normale */
q    /* è un puntatore al primo elemento dell'array */
punt=q; /* "punt" = l'indirizzo del primo elemento dell'array */

int func(int q[]) /* sintassi nella testata di una funzione */
a=func(q) /* sintassi nella chamata di una funzione */
```

Altri.

Sizeof.

```
int sizeof(nometipo);
```

Restituisce il numero di bytes che occupa ogni variabile di tipo "nometipo".

Autoincremento e autodecremento.

```
var++;
var--;
```

Incrementano o decrementano di una unità la variabile che li precede (o li segue). Equivalgono alla scrittura:
`var=var+1;`
`var=var-1;`
Particolarmente utili nei cicli for e while.

Operatori binari.

Operatori aritmetici e relazionali.

Operatore aritmetico	Sistassi	Funzione	Operatore relazionale	Sintassi	Funzione
+	c=a+b	somma algebrica	==	c=(a==b)	uguale
-	c=a-b	sottrazione	!=	c=(a!=b)	diverso
*	c=a*b	moltiplicazione	>	c=(a>b)	maggiore
/	c=a/b	divisione	<	c=(a<b)	minore

%	c=a%b	resto di divisione intera	>=	c=(a>=b)	maggiore o uguale
++	a++	incremento unitario	<=	c=(a<=b)	minore o uguale
--	a--	decremento unitario			

Gli operatori relazionali restituiscono un valore di verità.

Operatori logici e sui bit.

Operatore	Sintassi	Funzione
&	c=a&b	AND logico
|	c=a|b	OR logico
^	c=a^b	XOR logico
~	c=~a	NOT logico
<<	c=a<<b	shift sinistro
>>	c=a>>b	shift destro

Gli operatori >> e << scorrono i bit di "a" di "b" posti a destra o a sinistra .
Gli equivalenti logici sono:
`&&` per l'AND tra due condizioni logiche;
`||` per l'OR;
`!` per il NOT;

Esempio:

```
if ((a=5)&&(b=7))... /* esegue l'if se a=5 e b=7 */
while !(punt=NULL)... /* esegue finchè punt non è uguale NULL */
```

Appendice.

Escape sequences.

Consentono di formattare l'output, e si scrivere caratteri non stampabili.

Sequence	Value	Char	What it does
\a	0x07	BEL	Audible bell
\b	0x08	BS	Backspace
\f	0x0C	FF	Formfeed
\n	0x0A	LF	Newline (linefeed)
\r	0x0D	CR	Carriage return
\t	0x09	HT	Tab (horizontal)
\v	0x0B	VT	Vertical tab
\\	0x5c	\	Backslash
\'	0x27	'	Single quote (apostrophe)
\"	0x22	"	Double quote
\?	0x3F	?	Question mark
\O		any	O=a string of up to three octal digits
\xH		any	H=a string of hex digits
\XH		any	H=a string of hex digits

Caratteri di conversione.

Si usano nelle stringhe di formattazione (printf, scanf...) per indicare la rappresentazione dell'input o dell'output.

Type Char	Expected Input	Format of output
d	Integer	signed decimal integer
i	Integer	signed decimal integer
o	Integer	unsigned octal integer
u	Integer	unsigned decimal integer
x	Integer	unsigned hexadecimal int (with a, b, c, d, e, f)
X	Integer	unsigned hexadecimal int (with A, B, C, D, E, F)
f	Floating point	signed value of the form [-]dddd.dddd.
e	Floating point	signed value of the form [-]d.dddd or e[+/-]ddd
g	Floating point	signed value in either e or f form, based on given value and precision. Trailing zeros and the decimal point are printed if necessary.
E	Floating point	Same as e; with E for exponent.
G	Floating point	Same as g; with E for exponent if e format used
c	Character	Single character
s	String pointer	Prints characters until a null-terminator is pressed or precision is reached
%	None	Prints the % character
n	Pointer to int	Stores (in the location pointed to by the input argument) a count of the chars written so far.
P	Pointer	Prints the input argument as a pointer; format depends on which memory model was used. It will be either XXXX:YYYY or YYYY (offset only).

Infinite floating-point numbers are printed as +INF and -INF.

An IEEE ***Not-A-Number*** is printed as +NAN or -NAN.

Il formato complessivamente è il seguente:

```
% [.n] conv-char
```

Il segno % è obbligatorio e può essere seguito da uno specificatore di precisione, che specifica quante cifre decimali devono essere stampate, e dal carattere di conversione.

Esempio:

```
printf("Accelerazione= %.3f", a); /* stampa a come float con 3 cifre decimali */

printf("Valore esadecimale: %X", num); /* stampa num come valore esadecimale */
```

Appendice.