Linguaggio C

author: "Nicola Ferru"

Definizione

Il linguaggio C è un linguaggio compilato, fortemente tipizzato e case sensitive. Ovvimanete essendo un linguaggio compilato richiederà una ricompilazione nel caso in cui si cambi l'OS oppute l'architettura. È un linguaggio nato per i sistemi Unix negli anni 70' e quindi ha un ottica di programmazione orientata per quello che erano gli elaboratori all'epoca e anche quello che i programmatori usavano all'epoca, seguendo il loro schema mentale, che risultava leggermente diverso dal quello dei programmatori odierni, abbituati a linguaggi orientati alla programmazione ad Oggetti e poco abbituati a gestire manualmente le risorse. Il **C** è stato modificato nel tempo e attualmente lo standard è il **C11** del 2011, anche se maggiormente viene adottato il **C99**.

Architettura

I computer sono basati solitamente sulla architettura x86, più nello specifico x86_64, ma non è l'unica architettura esistente di altre un esempio è l'architettura arm e anche quella powerpc (**vecchi Machintosh e anche alcuni server della IBM**) che comunque risultano diffuse e in quel caso la gestione della memoria è diversa e non è certo che le variabili vengano gestite allo stesso modo. ma è sostanzialmente non è un problema perché è possibile verificare la dimensiome delle stesse.

Programma base "**hello world**"

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

int main(){
    printf("hello world");
    return 0;
}
```

Librerie

In C viengono utilizzate delle librerie per caricare e implementare nuove funzionalità, questo consente di ottimizzare al massimo il programma, per caricare una libreria è necessario utilizzare la direttiva al preprocessore `#include<nomeLibreria>` nel caso delle librerie di sistema, mentre, nel caso di librerie locali scritte ad'hoc per il programma bisogna utilizzare la direttiva `#include "nomeLibreria"`.

Quelle fondamentali

Nome	Descrizione

Nome	Descrizione
`<stdio.h>`	Contiene le funzioni per lo standard input\output
`<stdlib.h>`	Contiene funzioni di utilità generica
`<errno.h>`	Contiene tutto il necessario per la gestione degli errori
`<math.h>`	Contiene le funzioni matematiche avanzate utili per semplificare la vita al programmatore.
`<string.h>`	Una libreria che consente di gestire le stringhe "array di caratteri"
`<time.h>`	È una libreria che consente di sfruttare funzioni di tipo temporale "timer e letture dell'orario"
`<ctype.h>`	Dichiara funzioni per la classificazione dei caratteri.
`<float.h>`	Contiene le macro che vengono espanse ai vari limiti e parametri dei tipi in virgola mobile (floating-point) standard.

Il compilatore per sciendere li file da compilare da quelli libreria utilizza l'estensione del file per fare la verifica del compito del singolo file durante l'assemplaggio del file eseguibile.

Scelta dell'IDE

Per iniziare a programmare bisogan scegliere un IDE, un ambiente di sviluppo che ci consenta di lavorare ai potensiali progetti futuri.

discriminanti nella scelta

Un IDE moderno deve possedere determinati prerequisiti, uno tra tutti, è il riconoscimento della sintassi, che consente di verificare potenziali errori di pattitura e riduce l'errore. Un altro punto fondamentale è una buona integrazione con **GIT** che permette di tenere salvate le varie fasi di sviluppo rendendo più semplice la regressione in caso di problemi. I suggerimente sono ben accetti, quindi se l'editor possiede anche questa funzione tramete una shortcut è tutto un pro.

Commenti

Come in ogni buon linguaggio di programmazione anche il C supporta l'utilizzo dei commenti sia per singola linea sia multi-linea, i caratteri che vengono utilizzarti per indicare questo tipo di testo sono riportati qui sotto nella tabella.

Nome funzione	Descrizione
`// testo`	Commento a linea singola
`/* testo */`	Commento multi-linea

Il commento è un istruzione che non viene interpretata come codice eseguibile dal compilatore, che lo ignorerà e passerà alla prossima istruzione.

funzioni d'input/output basilari

Nome funzione	Descrizione
`printf()`	Funzione per stampare una stringa a schermo
`scanf()`	Funzione che consente di assegnare un valore ad una variabile

Queste due funzioni consentono di avere una interazione con il programma anche se non in modo persistente.

> Il termine persistenza in informatica, il concetto di persistenza si riferisce alla caratteristica dei dati di un programma di sopravvivere all'esecuzione del programma stesso che li ha creati: senza questa capacità questi infatti verrebbero salvati solo in memoria Ram venendo dunque persi allo spegnimento del computer.

By Wikipedia

Esempi

```c
// con testo statico
printf("testo");
// con testo dimanico
int c=4;
printf("%d",c);
// acquisizione di un valore
scanf("%d",&c);
```

Problemi con lo `scanf`? Sei di sicuro sotto Windows!

Microsoft Windows al contrario del resto del mondo usa due caratteri per gestire l'operazione "Vai a capo", quindi il buffer terrà in memoria il secondo carattere catturandolo nella `scanf` successiva il secondo carattere con tutti i problemi del caso, in pratica salterà un operaizione di input da parte dell'utente rendendo inutile l'operazione in questione. Questo problema ovviamente è ovviabile ed esistono due modi differenti per risolvere il problema:

1. Usando il comando `fflush(stdin)` che andrà a ripulire il buffer e risulta anche il modo più corretto di gestire il problema perché in questo modo viene eliminata una parte inutile;
2. Mettere una spazio perma del %tipoVariabile, ottenendo questo risultato `scanf(" %tipoVariabile",&x)`.

Variabili

Queste sono le variabili primitive presenti all'interno del C che compongono anche le variabili complesse e anche le strutture dati.

Nome	Descrizione

Nome	Descrizione
int	Numero intero
float	Numero reale
char	Carattere alfanumerico
double	Numero reale "esteso"
long	Numero intero "esteso"
short	Numero intero "ridotto"
void	Variabile nulla che viene utilizzata normamente per le funzioni che non devono rendere un valore.

- *esteso* - consente di inserire un valore più grande all'interno della variabile, ma occupa più spazio in memoria
- *ridotto* - permette di inserire un valore più piccolo rispetto alla variabile standard ma pesa meno memoria e quindi risulta più ottimizzato se non è necessario l'utilizzo di un **int** completo.

Operatore d'assegnamento

L'operatore d'assegnamento è un operatore che viene utilizzato per assegnare un dato valore all'interno di una variabile. viene espresso dal carattere `=`, non va confuso con l'operatore di confronto che viene espresso con `==` perché altrimenti sorgono dei problemi seri.

Esempio:

```
int i=0; // in questo caso il valore all'interno della variabile è stato sostituito con il valore 0
i=4; // adesso all'interno di i non è resente più 0 ma è presente il 4. Il valore 0 è stato eliminato definitivamente.
```

Ovviamente questo metodo funziona per le variabili primitive non per quelle composte.

Casting

Il casting è un pratica che consente di convertire il contenuto di una variabile di un determinato tipo in quello di un altra differente, questa pratica consente di svolgere delle operazioni che altrimenti non sarebbero possibili, come il poter avere il resto tra due interi trasformandolo in un numero relativo e quindi permette di avere un valore più preciso.
Esempio:

```
int a=42;
int b=3;
float ris=(float)a%b;
```

Il tipo lo si specifica tra parentesi tonde prima delle operazione di cui si vuole adattare il risultato.

Consigli

Dare sempre un nome valido alle variabili, perché così il programma risulta più leggibile dopo averlo scritto, cioè quando si scrive un programma non lo si fa mai di fila, molte volte è necessario prendere una pausa quindi se si danno dei nomi senza senso alle variabili diventarà più difficile interpretare l'utilizzo della stessa.

Contenuto delle variabili

Quando viene dichiarata una variabile senza averla inizializzata con un valore, il contenuto che si trova al suo interno è ""ignoto"", cioè il valore all'interno allo spazio di memoria è quello del programma che la stava utilizzando prima quindi non è gestibile e risulta totalmente random per il programmatore. Comunque per verificare la dimensione in memoria di una variabile bisogna utilizzare il `sizeof()`, non in tutti i computer è identico perché dipende dall'architettura.

Limite delle variabili

Le variabili avendo un valore fisso hanno un limite nella rapresentazione del dato, quindi per forza di cose è necessario conoscere questo limite per evitare troncature del risultato, perché ovviamente il sistema non ti avvisa del problema ma il risultato perde di precisione.

Rapresentazione nello standard input

rapresentazione	tipo variabile
`%d`	interi
`%f`	float / double
`%e`	decimali, in notazione esponenziale
`%c`	caratteri
`%s`	stringhe

Operatori logici e relazionali

Come tutti i linguaggi di programmazione, possiede una parte legata all'algebra booleana e anche agli aspetti logici come AND e l'OR e relazionali come maggiore, minore e uguale, anche le funzioni di comparazione.

Simboli	funzione
`==`	Comparazione
`&&`	AND logico
`||`	OR logico
`!=`	Differenza
`<`	minore

Simboli	funzione
>	maggiore
<=	minore\uguale
>=	maggiore\uguale

Operatori aritmetici

Simboli	funzione
-	sottrazione
+	somma
/	divisione
*	moltiplicazione
%	modulo (resto divisione)

Operatori composti

Simboli	funzione
+=	somma
-=	sottrazione
/=	divisione
*=	moltiplicazione
%=	modulo (resto divisione)

Esempio

```
cont+=5
// invece di
count=count+5
```

Occhio alle variabili non inizializzate perché non sono gestibili quindi non si può sapere che valore possa assumere il risultato.

Incremento di 1 di una variabile trammite ++ e decremento tramite il --

Il C ha una funzione che ti consente di incrementare un numero prima o dopo il richiamo della variabile, nel seguinte modo:

casi	funzione

casi	funzione
`NomeVariabile++`	incremento postumo
`++NomeVariabile`	incremento anticipato
`NomeVariabile--`	decremento postumo
`--NomeVariabile`	decermento anticipato

Nel caso della prima la variabile viene letta e poi dopo viene incrementata, mentre, nel secondo caso la variabile viene prima incrementata e poi letta. Stesso concetto per il decremento.

Visibilità delle variabili

Essettivamente una domanda che può essere posta è "Ma le variabile sono leggibili da tutte le funzioni?", la risposta a questa domanda è un sonoro no... Una variabile è disponibile solo dentro la funzione in cui viene dichiarata, ovviamente esistono anche le variabili globali ma effettivamente sono sconsigliate per una questione di sicurezza e di ordine. Infatti, per "sfruttare" (leggere) una variabile dichiareta altrove in un'altra funzione bisogna passarla tramite parametri e oltre tutto questa variabile non sarà modificabile se passata in questo modo, quindi qualunque modifica apportata dentro la funzione ospite non verrà realmente apportata. ovviamente un valore può essere reso e quindi salvato in questo modo, altrimente l'altro metodo per poter passare una variabile esterna è passandola tramite il puntatore della stessa. Onestamente questo implica una certa attenzioneda parte del programmatore per effettuare tali operazioni.

Dichiarare variabili globali

Per Dichiarare una variabile globale è sufficiente dichiarare una variabile al difuori di un corbo funzione.

```c
/* librerie importate */

static int nomeVariabile=3;

int main(){
  return 0;
}
```

Ovviamente il consiglio è quello di inizializzare sempre una variabile globale.

Le costanti

Le costanti come dice lo stesso nome sono dei valori che vengono assegnati staticamente, quindi non possono essere modificati durante l'esecuzione e neanche tramite altre operazioni, semplicemente sono dei valori assegnati e immutabili, ovviamente il programmatore può cambiare il valore quando la definisce ma non possono essere effettuate su di essa altre operazioni arbitrarie.

funzione	Descrizione
`#define nomeCostante valore`	Direttiva al preprogessore

funzione	Descrizione
`const tipoVariabile valore`	come variabile costrante

Il il risultato è lo stesso, ma gli utilizzi possibili sono differenti.

Contizioni "**i casi**"

In C è possibile verificare dei casi, con l'utilizzo della funzione `if` che consente di valutare una determinata valutare una condizione e nel caso sia prevista una condizione alternativa va utilizzato la funzione `else` e poi l'opzione alternativa.

esempio

```
if(x!=0)
  printf("il valore è maggiore di 0, perché il valore è %d",x);
else printf("il numero è 0");
```

Occhio, l'`else` non accetta parametri, per specificarli sarà necessario utilizzare un altro `if`.

switch case

Ovviamente esistendo casi con più possibilità esiste una funzione che consente aquisendo una variabile o una condizione di gestire diversi casi più uno "`default`" che viene scelto nel caso in cui il contenuto della variabile in questione non sia stato previsto.

```
int x;
scanf("%d",&x);
switch (x%3) {
  case 0:
  // istruzioni
  break;
  case 3:
  // istruzioni
  break;
  default:
  // istruzione predefinita
  break;

}
```

Cicli "iterazioni"

L'iterazione è l'atto di ripetere un processo con l'obiettivo di avvicinarsi a un risultato desiderato. Ogni ripetizione del processo è essa stessa definita un'iterazione, e i risultati di una sono utilizzati come punto di partenza per quella successiva. Diffuso è l'utilizzo negli algoritmi e nella programmazione in ambito

informatico, ma anche in campi come il project management.
By Wikipedia

Teorema di Jacopini-Bohm

Nella programmazione informatica, l'iterazione, chiamata anche ciclo o con il termine inglese loop, è una struttura di controllo, all'interno di un algoritmo risolutivo di un problema dato, che ordina all'elaboratore di eseguire ripetutamente una sequenza di istruzioni, solitamente fino al verificarsi di particolari condizioni logiche specificate.

Assieme alla sequenza o blocco e alla selezione è una delle tre strutture fondamentali per la risoluzione algoritmica di un dato problema secondo il Teorema di Böhm-Jacopini. Esistono varie forme di iterazione; le più conosciute sono il MENTRE (in inglese: while..do), il RIPETI (in inglese: repeat..until o do..while), ed il PER (comunemente detto ciclo for). Si può dire che l'iterazione è l'anello forte della programmazione che consente di automatizzare portando a termine un processo, al quale non basta la semplice esecuzione in sequenza di istruzioni.

Il cosiddetto "**ciclo infinito**", o "**loop infinito**", è un caso di iterazione dovuto solitamente ad un errore di programmazione che manda in stallo l'esecuzione del programma, mentre in alcune tecniche di programmazione soprattutto con microcontrollori è utilizzato in maniera voluta per iterare infinitamente all'interno del programma.

while & do/while

Il **while** e il **do/while** sono molto simili tra loro, nel caso del primo controlla la condizione all'inizio, mentre, il secondo controlla la conzione alla fine. questa funzione è molto utilizzata ed è l'unico modo per le più svariate operazioni, quindi bisogna avere una totale padronanza delle funzioni in questione conoscendo ogni minimo aspetto.

while

```
while(/* condizione */){
  // istruzioni da ripetere N volte
}
```

N = numero determinato di volte che dipendono esclusivamente da quello che si vuole fare.

do/while

```
do{
  // istruzioni da ripetere N volte
}while(/* condizione */);
```

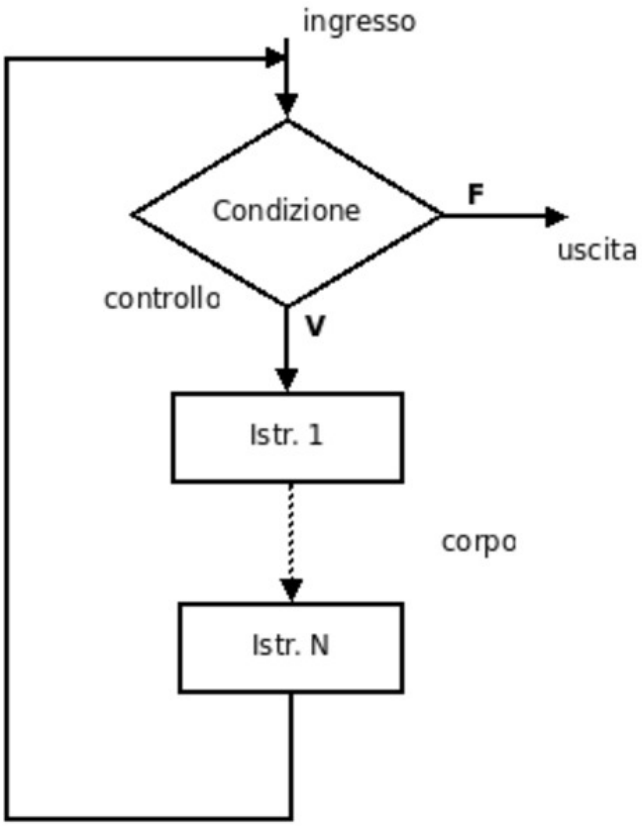
for

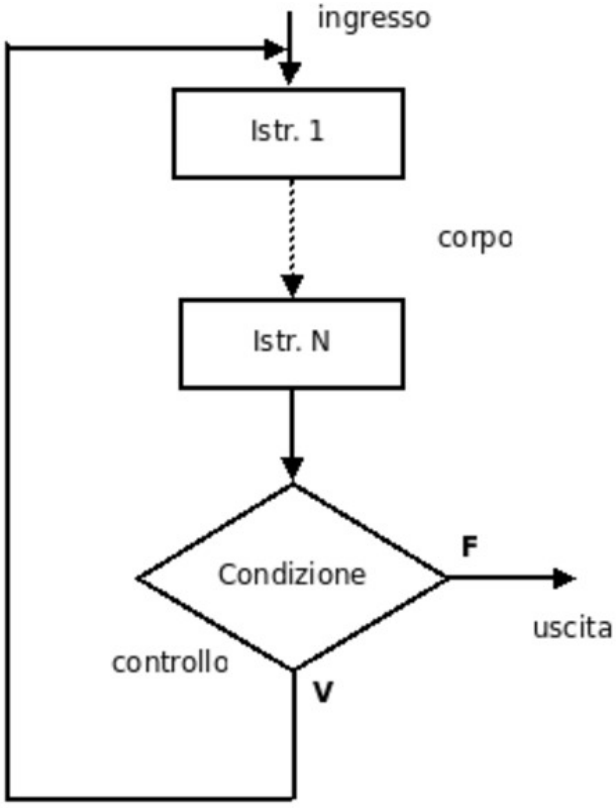
Il for è un'altra funzione che consente di effettuare un ciclo, controlla la condizione prima di effettuare un nuovo ciclo proprio come il while ma al contrario del while il for richiede di avere un numero fisso di volte, quindi è comodo per popolare gli array monodimensionali o bidimensionali "matrici".

```
int i;
for(i=0; i<N, i++){
  // istruzioni da ripetere.
}
```

Ovviamente in questo caso vendiamo le versioni iterative, ma tutto questo lo si può effettuare pure un modo ricorsivo, che risulta meno ottimizzato ma allo stesso tempo risulta molto più rapido da scrivere in molti casi se sono complessi da scrivere conviene sfruttare la ricorsione.

Iterativo





Attenzione!!

Bisogna prestare particolare attenzione al contatore del ciclo, perché deve essere sempre inizializzata perché altrimenti potrebbero creare dei problemi seri e sostanzialmete non funzionerà come da prassi.

Array

gli array sono un tipo di struttura dati che risulta composto da una serie di celle di memoria contigue, il contenuto delle singole celle deve essere dello stesso tipo, quindi quando si dichiara un array si deve avere un'idea chiara di quello che si vuole ottenere perché il tipo non lo si può cambiare inseguito dopo il run del programma. Di array ne esistono di due tipi statico e dinamico, il primo viene inizializzato con una

dimensione e non potra essere modificato in seguito. Mentre il secondo viene creato tramite i tool del C per gestire la memoria e poi verrà gestito tramite una variabile puntatore. Un ulteriore divisione che viene fatta per gli array è il fatto di essere monodimensionali (**vettore**) o bidimensionali (**matrice**).

Rapresentazione del array in memoria

Array mono dimensionale

v[0]	v[1]	v[2]	v[N]
1	2	0	35

Array bidimensionali

	m[0][0]	m[0][1]	m[0][2]	m[0][N]
m[1][0]	1	2	0	35
m[1][0]	5	6	7	8
m[2][0]	4	23	3	43
m[0][M]	6	4	5	9

Array statici

Gli arrai statici sono molto comodi quando sappiamo già quanti valori deve contenere, solitamente per gestire questo categoria viene utilizzata una costante N per gestire la dimensione.

```c
#define N 10 // ovviamente il numero lo si sceglie

int main(){
    int v[N]; // dimensione che va da 0 a N-1
    // ...
    return 0;
}
```

Per popolare un array sarà necessario utilizzare un ciclo. Ma si popolare anche per cella singola utilizzando l'indice dello stesso. Comunque non bisogna mai e poi mai sforare dalla array perché i risutati sono imprevedibili e soprattutto potrebbe portare ad un crush dell'applicativo per non dire del programma che sta utilizzando la memoria in questione.

Esempi di popolamento di un arrai statico

```c
    for (int i=0;i<N;i++){
        v[i]=N*i+4-random%10;
        // ...
    }
```

Un piccolo esempio

```c
#include <stdio.h>
#include <stdlib.h>
#define N 10

int main(){
    int vet[N];
    int i,ris,n=0;
    do {
        printf("inserisci il numero di elementi da sommare: ");
        scanf("%d", &n);
        if (n<0 || n>N){
            printf("Il valore inserito non è valido, ritenta\n");
        }
    }while (n<0 || n>N);
    for (i=0;i<n;i++){
        printf("inserisci il %d valore: ", i+1);
        scanf("%d",&vet[i]);
        ris+=vet[i];
    }
    printf("il risultato è %d\n",ris);
    return 0;
}
```

Le stringe

Una stringa è sempre un array di char ma un array di char non è detto che sia una stringa, cioè oltre al contenuto visibile è all'interno di un stringa è presente un carattere che indica la fine della stessa, che vale `NULL`, in questo caso viene rapresentato in questo modo `'\0'` che viene sfruttato dalle funzioni dedicate e effettivamente lo si frutta regolarmente per ridurre la complessità delle operazioni. In oltre bisogna sempre ricordare che questo non va mai eliminato, pena la trasformazione dello stesso in una semplice array di char rendendo inutilizzabili tutte le funzione scritte dai programmatori del C per rendere la gestione più semplice la gestione degli stessi. Infatti, l'intera libreria `<string.h>` è dedicata a questo scopo.

funzioni integrate della classe `<string.h>`

- strcpy(stringa_destinataria, stringa_originale) - permette di copiare una stringa in un'altra in modo automatizzato;
- strcpy(stringa_destinataria, stringa_originale, n_char) - permette di copiare una stringa in un'altra in modo automatizzato, poi scegliere il numero di caratteri da copiare;
- strcat(str1,str2) - contatena le stringhe;
- strlen(str) - rende la dimensione della stringa;
- strcmp - compara due stringhe e rende un intero che oscilla tra - infinito e + infinito, in base alle differenza di dimensione.

Logica dei puntatori

I puntattori vengono utilizzati in modo esplicito nel C, per gestire la memoria, infatti, esistono le variabili dedicate, che vengono dichiarate come le altre variabili ma con un * davanti al nome della stessa `int *i`. Sono fondamentali per l'utilizzo degli array dinamici e anche per la gestione dei file, proprio per la loro natura. Una delle funzioni che utilizza un puntatore come la scanf(), ogni variabile possiede il suo puntatore, per accedere al puntatore invece che al contenuto della stessa è necesario mettere davanti al nome il carattere `&`. Esistono anche i famigerati puntatori universali di tipo `void` che possono puntare a qualunque spazio di memoria a prescendere dal tipo della variabile. Ovviamente la dimensione occupata dal puntatore è sempre la stessa ma viene in suddivi in base alla rapresentazione fisica in memoria, perché ovviamente bisogna ricordare che i tipi di variabili hanno dimensioni defferenti tra loro proprio per il fatto che devono contenere delle rappresentazioni differenti.

Perché si utilizzano i puntatori

I puntatori sostanzialmente vengono utilizzati in due casi:

1. È indispensabile per modificare direttamente una variabile all'interno di una funzione, perché il C altrimenti va a creare una copia della stessa e quindi se si effettua una modifica alla stessa nel corso della funzione il valore contenuto nella variabile non verrà modificato;
2. la gestione della memoria in modo diretto consentendo la creazione di un array dinamico e anche altre strutture dati con dimensione variabili ma non continue.

Questo tipo di operazioni saranno molto frequenti all'interno della scrittura dei programmi. Si arriverà proprio ad un punto in cui non ci saranno più passaggi diretti ed espiciti.

Esempio

```
int main (){
    int *p;
    int a=3;
    p=&a; // assegno a p l'indirizzo della variabile a
    printf("%d",*p); // stampo il valore contenuto nel indirizzo a cui
punta p.

}
```

Come dovreste aver notato quando si vuole accedere al puntatore si utilizzano due modalità, la prima quella senza l'asterisco * consente di utilizzare il puntatore come quello che è una variabile che può contenere solo indirizzi, mentre, il secondo quando si mette davanti l'asterisco consente di accedere al contenuto della varabile puntata.

Gestione dei file

Come in tutti i linguaggi strutturati, il `C` ha la possibilità di gestire dei file, ovviamente è sempre il C, quindi le operazioni vanno svolte manualmente. E soprattutto richiedono una certa attenzione, perché il rischio è quello di sovrascrivere qualche documento in portante che una volta perso non lo si recupera, è perso definitivamente.

Variabili dedicate

Nome variabile	Descrizione
FILE	questo tipo di variabile indice un file fisico presente sul device d'archiviazione in questione

Effettivamente la cosa importate in questi casi è fornire un percorso valido in cui salvare lo stesso.

Funzioni dedicate

Nome funzione	Descrizione
`fprintf()`	Funzione per stampare una stringa in un file
`fscanf()`	Funzione che consente di assegnare un valore da file
`fopen()`	Funzione che consente di aprire un file
`fclose()`	Funzione che consente di chiudere un file

Esempi

Operazione di scrittura su file

```
/* librerie standard */

int main() {
  FILE *fd;
  int x=-32;

        /* apre il file in scrittura */
  fd=fopen("scrivi.txt", "w");
  if( fd==NULL ) {
    perror("Errore in apertura del file");
    exit(1);
  }


        /* scrive il numero */
  fprintf(fd, "%d\n", x);


        /* chiude il file */
  fclose(fd);

  return 0;
}
```

Strutture

In C è possibile accorpare una serie di variabili a sieme, permettendo di sopperire al limite dell'array di una variabile convinzionele, quindi si crea una variabile composta per ottenere il risultato tanto desiderato, il suo nome è struttura e va dichiarata prima del corpo **main()** e sostanzialmente funzionano da contenitore.

```
struct nomeStruttura {
  int a;
  char b;
  //...
};
```

Esempio

Una data è composta da tre numeri interi quindi il metodo migliore per rapresentare questo tipo di elemento è proprio la struttura e lo fa anche in modo eccellente visto che di strutture possono essere anche fatti degli array.

```
struct data{
  int giorno;
  int mese;
  int anno;
};
```

E così si risolve un problema gestionale che normalmente fa pardere diverso tempo. Per fruttare la veriabile contenuta all'interno della struttura basta utilizzare la sintassi `nomeStruttura.nomeVariabile=Valore`.

```
  data.giorno=31;
  data.mese=10;
  data.anno=3002;
```

Ovviamenre i campi presenti all'interno di una struttura possono essere anche degli array, proprio per il fatto che le strutture possono contenere delle variabili e quindi va bene così, ti semplifica la vita e riduce lo sfrorzo durante la programmazione.

Operazione di recupero dei dati da un file

```
int main(){
  FILE *pf;
  char nome_file[128];
  printf("Inserisci il nome di un file (percorso completo): ");
  scanf ("%s" , nome_file );
  pf = fopen(nome_file , "r"); if (pf) {
      fseek(pf, SEEK_END);
      printf ("%s è lungo %ld bytes" , nome_file , ftell (pf ));
      fclose( pf );
```

```
  }else{
    printf("%s non esiste !", nome_file );
  }
  return 0;
}
```

Consiglio

Il consiglio è di scrivere una funzione esterna al `main` per gestire queste operazioni, perché così si possono gestire al meglio le eccezioni e quindi si può impedire all'utente finale di fare potenziali danni al suo stesso sistema delimitando le operazioni possibili guidandolo nel giusto modo. Perché l'utente non sa cosa sta facendo.

ATTENZIONE!!!!

Quando si apre un file bisogna sempre e comunque chiuderlo quando si finisce di esequire una determinata operazione, altrementi si rischia creare degli effettivi problemi, uno dei tanti è che il file risulta in utilizzo finché il programma è aperto anche se non è più necessario, per di più rischi di danneggiare lo stesso scrivendoci per errore e tanti altri problemi logici che non possono essere espressi in due righe.

Le funzioni

Le funzioni vengono utilizzate in programmazione per poter scomporre un problema complesso in tanti più piccoli più semplici da gestire e da scrivere. La funzione può essere di diversi tipi, infatti, proprio come una variabile condivide gli stessi tipi proprio per il fatto che la stessa deve rendere qualcosa indietro che sia un intero o un numero reale, l'importante è che tutto venga gestito nel migliore dei modi e venga scritto modo più ordinato possibile.

Esempio

Funzione che rende una media

```
float media (int v[]) {
  int i,
  float ris=0;
  for (i=0;i<N;i++)
    ris+=(float)v[i];
    ris/=N;
  return ris;
}
```

Funzione che svolge le tabelline

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

void tabelline(int n, int nv){
```

```
    int ris=1,i;
    printf("-- tabelline --");
    for (i=0;i<nv;i++){
        ris=n*i;
        printf("\n%d * %d = %d",n,i,ris);
    }
    printf("\n%d * %d = %d\n",n,i,ris);

}

int main (){
    int base, numeroVolte;
    printf("inserisci il tabellina: ");
    scanf("%d", &base);
    printf("inserisci numero di moltiplicazione: ");
    scanf("%d", &numeroVolte);
    tabelline(base,numeroVolte);
}
```

Stampa di un array monodimensionale

Il problema più elementare è la dupplicazione del codice, quindi la solunzione migliore nella gestione degli array è la seguente.

```
  void stampaVet(int V[], int dim /* La dimensione dell'array */){
    if(dim<0){
      printf("Errore: dimensione errata");
      return ;
    } else{
      for(int i=0;i<dim,i++)
        printf("%3d ",v[i]);
    }
  }
```