Manipolo il linguaggio

Le strutture....1
I File....2
L' allocazione dinamica....3
Le stringhe....4
Gli Ordinamenti....5
La Ricorsione....6

Le strutture

Una struttura è un area di memoria che definiamo noi precedentemente che può contenere al suo interno una serie di altri tipi.

Ad esempio la struttura *studenti* può contenere al suo interno i *campi*:

- *matricola (intero)*
- *nome (stringa)*
- *cognome (stringa)*
- *anni (intero)*

questi campi vengono gestiti da un vettore di strutture lungo quanto il numero degli studenti da gestire, ecco qui di seguito un programma che gestisce l'inserimento relativo a 100 studenti.

```
#include<stdio.h>
#define N 3

struct studenti{
	int matricola;
	char nome[10];
	char cognome[10];
	int anni;
}

int main(){
	int i;
	struct studenti allievo[N];

	printf("Inserisci: Matricola, Nome, Cognome, Età");
	for(i=0;i<N;i++){
		printf("\nAllievo [%3d]: ",i+1);
		scanf("%d %s %s %d", \
			&allievo[i].matricola, allievo[i].nome , \
			allievo[i].cognome, &allievo[i].anni);
	}

	for(i=0;i<N;i++)
		printf("\n%5d %10s %10s %3d", \
			allievo[i].matricola,allievo[i].nome, \
			allievo[i].cognome,allievo[i].anni);

	printf("\nfine.\n");
	return 0;
}
```

```
samba@sadeo:~$ gcc -o studenti studenti.c
samba@sadeo:~$ ./studenti
Inserisci: Matricola, Nome, Cognome, Età
Allievo [  1]: 12345 Pinco Pallino 15
Allievo [  2]: 12435 Pippo Pluto 16
Allievo [  3]: 14324 Minni Topa 14
...

12345     Pinco    Pallino   15
12435     Pippo     Pluto    16
14324     Minni    Topola   14
...
...

fine.
```

Le strutture sono quindi utili per manipolare aree di memoria che contengono vari tipi al suo interno come in questo caso dei tipi di interi (anni) e dei tipi stringa (nome).

I File

Supponiamo adesso di voler leggere la stessa struttura da un file e non da tastiera, poiché può sembrare lungo e noioso dover inserire 100 volte i dati degli studenti.

Le poche differenze che ci sono nella lettura da FILE e nella lettura da TASTIERA sono le funzioni per i file e un nuovo tipo *FILE* *, che identifica un puntatore al file.

Ecco di seguito, l'esempio precedente modificato per la lettura da un file di questo tipo:

file.txt
12345 Pinco Pallino 15 12435 Pippo Pluto 16 14324 Minnie Topola 14 ... ecc ....

```
#include<stdio.h>
#define N 3

struct studenti{
	int matricola;
	char nome[10];
	char cognome[10];
	int anni;
};

int main(int argc, char **argv){
	int i;
	struct studenti allievo[N];
	FILE *pf;
	char filename[10];

	printf("Inserisci il nome del file: ");
	scanf("%s",filename);
	pf = fopen(filename,"r");
	if(pf == NULL){
		printf("Errore nell'apertura del file.\n");
		return (1);
	}
	for(i=0;i<N;i++){
		fscanf(pf,"%d %s %s %d",
		&allievo[i].matricola , allievo[i].nome,  \
		allievo[i].cognome,&allievo[i].anni);
	}

	for(i=0;i<N;i++)
		printf("\n%5d %10s %10s %3d",
		 allievo[i].matricola,allievo[i].nome,  \
		 allievo[i].cognome,allievo[i].anni);

	fclose(pf);
	printf("\nfine.\n");
	return (0);
}
```

```
Inserisci il nome del file: file.txt

12345      Pinco    Pallino  15
14232      Pippo      Pluto  16
14534     Minnie     Topola  14
fine.
```

Come vedete l'input è immediato e non c'è più bisogno di scrivere i dati di ogni studente, quindi decisamente più veloce e ottimizzante.

Ho usato il tipo *FILE* per dichiarare un puntatore a file: pf, successivamente ho letto il nome del file da tastiera e aperto il file tramite la funzione *fopen* specificando il tipo *r* per la sola lettura. La funzione *fscanf* è uguale alla scanf se non che specificando il puntatore pf la funzione preleva i dati dal file precedentemente aperto. per poi chiuderlo con la funzione *fclose.*

L' allocazione dinamica

Supponiamo ora che il numero di studenti non sia noto a priori ma sia specificato nella prima riga del file impedendo in questo modo l'allocazione statica, ovvero *struct studenti allievo[**100**]*, essendo che il numero *100* non è più noto.

In quseto modo dovremmo prima leggere il file e successivamente allocare il vettore di studenti.

Supponiamo quindi che il file sia questo:

file.txt
3 *12345 Pinco Pallino 15* *14232 Pippo Pluto 16* *14534 Minnie Topola 14*

e il programma quindi questo:

```
#include<stdio.h>
#include<stdlib.h>
#define N 3

struct studenti{
	int matricola;
	char nome[10];
	char cognome[10];
	int anni;
};

int main(int argc, char **argv){
	int i,n;
	struct studenti *allievo;
	FILE *pf;
	char filename[10];


	printf("Inserisci il nome del file: ");
	scanf("%s",filename);
	pf = fopen(filename,"r");
	if(pf == NULL){
		printf("Errore nell'apertura del file.\n");
		exit(EXIT_FAILURE);
	}
	fscanf(pf,"%d",&n);
	allievo = malloc(n * sizeof(struct studenti));
	if (allievo == NULL ){
		printf("errore nell'allocare\n");
		exit(EXIT_FAILURE);
	}

	for(i=0;i<N;i++){
		fscanf(pf,"%d %s %s %d",
		&allievo[i].matricola,allievo[i].nome,	\
		allievo[i].cognome,&allievo[i].anni);
	}

	for(i=0;i<N;i++)
		printf("\n%5d %10s %10s %3d",
		allievo[i].matricola,allievo[i].nome,	\
		allievo[i].cognome,allievo[i].anni);
	fclose(pf);
	free(allievi);
	printf("\nfine.\n");
	return (0);
}
```

```
Inserisci il nome del file: file.txt

12345      Pinco    Pallino  15
14232      Pippo      Pluto  16
14534     Minnie     Topola  14
fine.
```

Tre sono le cose che cambiano nel nostro programma:

1. *struct studente *allievo*: alloco la mia struttura successivamente, quindi all'inizio avrò solo un puntatore che successivamente apparterrà ad una struttura *studenti*.

2. Eseguo la prima *fscanf* per ottenere il numero di studenti e poi alloco *allievi* con la funzione *malloc* al cui interno uso la funzione *sizeof* per sapere quanto è la grandezza della struttura *studenti.*
3. Controllo che *allievi* non sia uguale a *NULL* ,altrimenti sarebbe errore. Al termine del programma libero la memoria precedentemente allocata usando la funzione *free*.

Le stringhe

Faccio un piccolo approfondimento sulle stringhe poiché ne abbiamo parlato in modo molto rapido e superficiale. Le stringhe non sono altro che dei vettori di caratteri e in quanto vettori ad ogni cella contengono la singola informazione, quindi il singolo carattere. Quindi una stringa come questa:

stringa =

C	I	A	O	\0

sarà quindi :

*string[0] = '**C**' , string[1]='**I** ',string[2] = '**A**', string[3]= '**O**', string[4] = '**0**' .*

L'ultimo carattere *\0* è un carattere speciale che segna il termine della stringa.

ATTENZIONE: Chiamare *stringa* unicamente con il nome della variabile significa chiamare i suo puntatore: *stringa = 0x12345* ; ci rendiamo conto che quindi eseguire normali operazioni con le stringe come il confronto, la copia o la lunghezza richiedono l'uso di altre funzioni, come in quiesto esempio:

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

int main(int argc, char **argv){
        int i;
        char s1[100],s2[100], *s;
   do{
        printf("Inserisci il primo nome: ");
        scanf("%s",s1);
        printf("Inserisci il secondo nome: ");
        scanf("%s",s2);
    while(strlen(s1)>10 || strlen(s2)>10);

        if( strcmp(s1,s2) > 0)
                printf("%s \n%s",s2,s1);
        if (strcmp(s1,s2)<0)
                printf("%s\n%s",s1,s2);
        if(strcmp(s1,s2)==0){
                s = strdup(s1);
                printf("stringhe uguali: %s",s);
        }
        printf("\nfine.\n");
        return (1);
}
```

```
samba@sadeo:~$ ./stringhe
Inserisci il primo nome: Anna
Inserisci il secondo nome: Zucchero
Anna
Zucchero
fine.

samba@sadeo:~$ ./stringhe
Inserisci il primo nome: Zucchero
Inserisci il secondo nome: Anna
Anna
Zucchero
fine.

samba@sadeo:~$ ./stringhe
Inserisci il primo nome: Anna
Inserisci il secondo nome: Anna
stringhe uguali: Anna
fine.
```

1. utilizziamo una nuova libreria che si chiama *string.h* per utilizzare le funzioni relative alle stringhe
2. la funzione *scanf* non richiede una & poiché chiamare il nome della stringa significa chiamarne il suo puntatore
3. La funzione *strlen* conta quanti caratteri ha una stringa e ne restituisce il numero, in questo caso se il numero è maggiore di 10 chiediamo il reinserimento.
4. la funzione *strcmp* confronta due stringhe e restituisce un numero minore uguale o maggiore di zero a seconda del risultato.
5. la funzione *strdup* prende una stringa completa e la copia in una ottimizzata e allocata

esattamente quanti sono i caratteri e restituendone il puntatore, in questo modo oltre che essere più facile l'utilizzo, si risparmia memoria.

Gli Ordinamenti

Esistono diversi tipi di ordinamenti tra questi quelli più comuni e quelli meno comuni, complicati e quasi impossibili da capire immediatamente che possono poi rendersi però molto veloci. Tra questi:

- Selection Sort: prendo una carta se è più piccola la porto all'inizio.
- Insertion Sort: parto dai primi 2 e il più piccolo lo metto ordinato e vado avanti
- Bubble Sort: il più piccolo viene confrontato a due a due con tutti finché non galleggia all'inizio
- Counting Sort: Conto le occorrenze e tramite conteggi e indici posiziono ordinatamente i valori
- Merge Sort(ricorsivo): spacco il vettore a metà finchè non ne ho solo 2 che ordino e risalgo
- Quick Sort(ricorsivo): Scelgo un Pivot e ordino da inizio a fine spostando poche volte i dati.

ed ecco qui di seguito i codici di ogni ordinamento:

Selection Sort

```
void SelectionSort(int v[]){
	int i ,j, small;
	printf("\n\t\t\tSelection Sort\n");

	for(i=0;i<N-1;i++){
		small=i;
		for(j=i+1;j<N;j++)
			if(v[j]<v[small]){
				small = j ;
			}
	scambia(v, small , i );
	}
}
```

Insertion Sort

```
void InsertionSort(int v[]){
	int i , j , key ;
	printf("\n\t\t\tInsertion Sort\n");
	for(i=0;i<N ;i++){
		key = v[i];
		j=i-1;
		while(j>=0 && v[j]>key){
			v[j+1]=v[j];
			j--;
		}
		v[j+1]=key;
	}

}
```

Bubble Sort

```
void BubbleSort(int v[]){
	int i,j;
	printf("\n\t\t\tBubble Sort\n");
	for(i=N-1;i>1;i--){
		for(j=0;j<i;j++)
			if(v[j]>v[j+1])
				scambia(v , j , j+1);
		stampa(v);
	}
}
```

Counting Sort

```
#define k 15
void CountingSort(int A[], int n){
	int i, j, *B, C[k];
	B = malloc(n*sizeof(int));
	for (i=0; i<k; i++)	C[i] = 0;
	for (i=0; i<n; i++)	C[A[i]]++;
	for (i=1; i<k; i++)	C[i] += C[i-1];
	for (i=n-1; i>=0; i--)
		B[C[A[i]]-1] = A[i];C[A[i]]--;
	for (i=0; i<n; i++)
		A[i] = B[i];

}
```

Ordinamenti Ricorsivi:

Quick Sort	Merge Sort

void quicksort (int A[], int p, int r)
{
int q;
  if (p < r)
  {
  q = partition(A, p, r);
  quicksort(A, p, q);
  quicksort(A, q+1, r);
  }
  return;
}
int partition (int A[], int p, int r)
{
int x, i, j, temp;
  x = A[p]; i=p-1; j=r+1;
  while (i < j)
  {
  while(A[--j] > x);
  while(A[++i] < x);
  if (i < j)
  {
  temp = A[i];
  A[i] = A[j];
  A[j] = temp;
  }
  }
  return(j);
}
	void MergeSort(int A[], int p, int r)
	{
	int q;
	  if (p < r) {
	  q = (p + r)/2;
	  MergeSort(A, p, q);
	  MergeSort(A, q+1, r);
	  Merge(A, p, q, r);
	  }
	  return;
	}
	void Merge(int A[], int p, int q, int r)
	{
	int i, j, k, B[MAX];
	  for (i = p, j = q+1, k = p; i <= q && j <= r;)
	  if (A[i] < A[j]) B[k++] = A[i++];
	  else B[k++] = A[j++];
	  for (; i <= q;) B[k++] = A[i++];
	  for (; j <= r;) B[k++] = A[j++];
	  for (k = p; k <= r; k++) A[k] = B[k];
	  return;
	}

La Ricorsione

Mi rendo conto che presentarvi degli algoritmi ricorsivi può essere un po' strano, ma basta pensare semplicemente che una funzione ricorsiva è una funzione che richiama se stessa.

La ricorsione si usa spesso, ma non sempre è il metodo migliore. Tra i vari tipi di algoritmi che ho affrontato posso dire che esistono principalmente 2 tipi di algoritmi che richiedono la ricorsione:

- Divid et impera : divido un problema fino a quando non diventa semplice e lo risolvo.
- Provale tutte una sarà la soluzione ottima e vai avanti, se non va torna indietro e riprova.

La prima tipologia è quella usata anche nel merge sort, ovvero abbiamo diviso un vettore a metà fino a quando non abbiamo ottenuto un vettore di un elemento. Essendo un vettore di un elemento sicuramente ordinato abbiamo risolto un problema semplice e torniamo su ad un problema più complesso, avendo già risolto un pezzo.

Altro caso è il calcolo di un fattoriale per cui per calcolare il fattoriale di un numero molto grande è difficile ma sottraendo il valore da calcolare fino a 1 oppure 0 (sapendo che è uguale a 1) tornare su sapendo già il risultato divide il problema.

fattoriale.c

```
int fact(int n){
	int sub;
	if(n==0)
		return 1;
	else{
		sub = fact(n-1);
		return(n*sub)
	}
}
```

La seconda tipologia è quella usata per quei problemi per cui non si trova un altro modo per agire se non quelle di provarle tutte fino a quando non si trova il risultato ottimo(localmente o globalmente). Ad esempio se dovessimo decidere come posizionare dei mobili di diverse dimensioni dentro dei camion di diversa dimensione e bilanciare tutti i camion per non appesantirne troppo nessuno. in questo caso ci è necessario provarle tutte, poiché non esiste una formula matematica che ci garantisce il bilanciamento e il posizionamento ottimale di tutti i mobili.

(Il file relativo ai camion ve lo allego nell'archivio **algoritmi-ricorsivi.tgz**)

Seconda lezione finita.

sono le 3 di notte del 3 marzo.

cazzo, ho fatto tardi....

"L'uomo e' ancora il piu' straordinario dei computer."

-- John F. Kennedy, "Discorso, 21 maggio 1963"

Autore: samba@autistici.org