Introduzione corso Arduino – Array, Puntatori, Funzioni e Struct

Dato un generico array possiamo svolgere le seguenti azioni su di esso:

- Inserimento
- Cancellazione
- Ricerca
- Ordinamento

Inserimento

Dato un generico array di dimensione dim e già caricato con n elementi, con n < dim, può essere necessario dover inserire un nuovo elemento in una posizione x < n, ovvero in cui è già presente un valore, senza voler perdere il valore già presente.

Esempio:

array = | 4 | 8 | 2 | 5 | | | dim = 6 n = 4

si vuole inserire il numero 7 alla posizione x = 1 in cui è già presente il numero 8.

Se n < dim, quindi ci sono ancora posti vuoti all'interno dell'array, allora possiamo procedere copiando a destra di un posto tutti gli elementi dall'indice n-1 fino all'indice x.

array = | 4 | 8 | 2 | 5 | | |

array = | 4 | 8 | 2 | 5 | 5 | |

array = | 4 | 8 | 2 | 2 | 5 | |

array = | 4 | 8 | 8 | 2 | 5 | |

Partendo dall'ultimo indice contenente un valore, copio il suo valore nell'indice successivo.

Arrivato a questo punto posso inserire il nuovo valore nell'indice x e aumentare n di uno.

array = | 4 | 7 | 8 | 2 | 5 | | dim = 6 n = 5

Algoritmo:

```
if (n + 1 <= dim){                  //Controllo se sono presenti posti vuoti nell'array
    for(int i = n - 1; i >= x; i--){  //Scorro l'array da n-1 a x
        array[i + 1]  = array[i];     //Copio di un posto a destra tutti i valori da n-1 a x
    }
    array[x] = 7;                   //Inserisco il nuovo valore all'indice x
    n++;                            //Aumento di uno il numero di elementi presenti nell'array
}
```

Cancellazione

Dato un generico array di dimensione dim e già caricato con n elementi può essere necessario dover cancellare un elemento in una posizione x.

Esempio:

array = | 4 | 8 | 2 | 5 | | | dim = 6 n = 4

si vuole eliminare l'elemento di indice x = 1.

Si procede dunque copiando a sinistra di un posto tutti gli elementi da x + 1 a n – 1.

array = | 4 | 8 | 2 | 5 | | |

array = | 4 | 2 | 2 | 5 | | |

array = | 4 | 2 | 5 | 5 | | |

Si conclude diminuendo n di uno

Realizzato da Davide Malvezzi

array = | 4 | 2 | 5 | 5 | | | dim = 6 n = 3

Si nota che l'ultimo e il penultimo elemento sono uguali. Questo non è un problema perchè avendo diminuito n di uno fa si che l'ultimo elemento non sia mai raggiunto.

Algoritmo:

```
for(int i = x; i < n - 1; i++){      //Scorro tutti gli elementi da x a n - 1
    array[i] = array[i + 1];         //Copio sull'indice corrente il valore successivo
}
n--;                                 //Diminuisco il numero di elementi presenti nell'array di uno
```

Ricerca

Dato un generico array di dimensione dim e già caricato con n elementi può essere necessario dover trovare la posizione di un elemento x. L'idea più banale è quella di scorrere tutto l'array e controllare se il valore all'indice corrente è uguale al valore che si sta cercando. Nel caso sia così allora l'elemento si trova all'indice corrente, altrimenti, se l'array è stato scorso completamente, l'elemento non è presente nell'array.

Algoritmo:

```
i = 0;                               //Inizializzazione dell'indice
while(i < n && array[i] != x) i++;   //Si può leggere come "finchè ci sono ancora elementi nell'array (i < n) e
                                     l'elemento corrente è diverso da quello che si sta cercando (array[i] != x) passa a
                                     controllare l'elemento successivo (i++).

if(i < n){                           //Il while si interrompe solamente se i > n o se array[i] == x.
    …                                Se dopo il while i è ancora minore di n vuol dire che la condizione di uscita è stata
}                                    array[i] == x, quindi l'elemento x è stato trovato in posizione i.
else{                                Se dopo il while i è >= di n allora l'elemento non è stato trovato.
    ...
}
```

Ordinamento

Dato un generico array di dimensione dim e già caricato con n elementi può essere necessario dover ordinare gli elementi in ordine crescente (o decrescente). Uno degli algoritmi di ordinamento più semplici da implementare e facile da ricordare è il selection sort. L'idea di base del selection sort è quella di scorrere tutto l'array fino n – 1 e per ogni elemento i scorrere nuovamento l'array da i + 1 ad n per cercare un elemento minore del corrente. Nel caso sia presente, i due elementi vengono scambiati di posto.

Esempio:

Si vuole ordinare in ordine crescente il seguente array:

array = | 4 | 8 | 2 | 5 | | | dim = 6 n = 4

Per ogni elemento dell'array cerchiamo nelle posizione successive un elemento che sia minore. Per esempio partendo con i = 0, array[i] vale 4. Se guardiamo negli indici successivi ad i, l'elemento più piccolo è 2 in posizione 3. Trovato l'elemento minore possiamo procedere a scambiare i due di posto.

array = | 4 | 8 | 2 | 5 | | | i = 0

array = | 2 | 8 | 4 | 5 | | | i = 1

Aumentiamo i di uno e ripetiamo le operazioni precedenti. Ora i = 1 e array[i] vale 8. Cerchiamo un elemento minore di 8 e di posizione successiva ad i. In caso ci siano più di due elementi minori dell'elemento corrente, come in questo caso (4 e 5 < 8), si considera solo l'elemento più piccolo. Procediamo a scambiare 8 con 4.

array =	2	4	8	5			i = 2

Aumentiamo nuovamente i di uno e procediamo nuovamente come prima, fino a quando i < n – 1.

array =	2	4	5	8			i = 3

Algoritmo:

```
int temp;                          //Variabile di supporto per effettuare lo scambio
int min;                           //Variabile che conterrà l'indice dell'elemento minore
for(int i = 0; i < n – 1; i++){    //Scorro tutto l'array fino ad n – 1
    min = i;                       //Inizializzo la posizione del minore con l'indice corrente
    for(int j = i + 1; j < n; j++){ //Scorro l'array dalla posizione corrente fino ad n
        if(array[j] < array[min]){ //Se l'elemento di indice j è minore dell'elemento di indice min
          min = j;                 //Assegno a min j (ho trovato un elemento minore del precedente)
        }
    }
    temp = array[i];               //Effettuo lo scambio.
    array[i] = array[min];
    array[min] = temp;
}
```

Nel caso si volesse ordinare in ordine decrescente basta mettere > nella condizione dell'if.

Questi 4 algoritmi, bene o male, si ritrovano sempre in programmi dove vengono utilizzati degli array. È bene quindi cercare di ricordarseli. Impararli a memoria non è una soluzione per ricordarli. Un buon modo è cercare di capire cosa faccia ogni algoritmo e da quello cercare di implementarlo da zero.
Per esempio la frase "L'algoritmo di inserimento sposta a destra tutti gli elementi dopo la posizione in cui si vuole inserire un elemento", potrebbe aiutare a ricordare il principio di funzionamento dell'inserimento. Questa frase da per sottinteso che l'array deve avere almeno un indice libero per inserire un elemento e che il contatore del numero di elementi dell'array dovrà essere aumentato di uno una volta terminato l'inserimento, ma è comunque una buona base di partenza. Esercitatevi a fare dei "giochini" di questo tipo!

Puntatori

Ogni variabile utilizzata all'interno di un programma è un'area di memoria centrale che contiene un valore e ogni area di memoria ha un indirizzo univoco. Un puntatore è una variabile che contiene un indirizzo di memoria di un'altra variabile. I puntatori si dichiano nel seguente modo:

```
tipo_variabile *nome_variabile;
```

per esempio:

```
char   *ch          // puntatore ad una variabile char
int    *ip;         // puntatore ad una variabile int
float  *fp;         // puntatore ad una variabile float
```

Per ottenere il puntatore di una variabile si usa l'operatore &:

```
int var = 20;
int *ip;            //in questo caso la variabile ip conterrà l'indirizzo di memoria di var.
ip = &var;
```

Nel caso avessimo un puntatore ad una variabile possiamo accedere e modificare il valore della variabile con l'operatore *:

```
*ip = 32;
printf("%d", *ip);      //stampa 32
```

Operazioni matematiche sui puntatori

Le variabili puntatori possono essere considerate come variabili contenenti un numero e quindi è possibile applicare operazioni matematiche su di esse. Le principali operazioni sono quelle di addizione e sottrazione, effettuate con gli operatori +, ++, - e --. Per ogni unità sommata (o sottratta) ad un puntatore, esso avanzerà (o tornerà indietro) di n * dimensione della variabile puntata byte.
Per esempio:

```
char c, *pc;
int i, *pi;
float f, *pf;

pc = &c;
pi = &i;
pf = &f;

pc = pc + 1;        //pc avanza di un solo byte perchè la dimensione in byte di char è 1
pi = pi + 1;        //pi avanza di 4 byte perchè la dimensione in byte di int è 4
pf = pf + 2;        //pf avanza di 8 byte perchè la dimensione in byte di float è 4
```

Array e puntatori

Quando viene allocato un vettore, la variabile-vettore non è altro che un puntatore al primo elemento dell'array quindi possiamo dire che:

```
int a[8];

for(int i = 0; i < 8; i++){
        printf("puntatore %p\n", &v[i]);
        printf("valore %d\n", v[i]);
}
```

← Equivale a →

```
int a[8];

for(int i = 0; i < 8; i++){
        printf("puntatore %p\n", a + i);
        printf("valore %d\n", *(a + i));
}
```

Funzioni e procedure

Le funzioni sono blocchi di codice che a partire da certi dati in input producono un risultato. Per dichiarare una funzione dobbiamo prima di tutto definire il prototipo prima del main(). Un prototipo è così strutturato

Tipo_dato_output Nome_Funzione([Tipo_paramentro_1, Nome_paramentro_1], ...);

Per esempio un prototipo di una funzione che effettui la somma di due numeri interi è:

int somma(int a, int b);

La funzione deve essere poi dichiarata dopo il main:

```
int somma(int a, int b){
    int s = a + b;
    return s;
}
```

L'istruzione return indica il valore che la funzione deve "ritornare" una volta che viene chiamata. Ogni funzione può ritornare al massimo un valore. La funzione può essere poi richiamata da main:

```
int main(){
    int n1 = 5, n2 = 4;
    int s = somma(n1, n2);
}
```

La funzione viene richiamata attraverso il suo nome e con fra () i paramentri necessari. I parametri di una funzione devono essere delle stesso tipo e in stesso numero di quelli del prototipo (valgono ovviamente le regole di casting). Nei parametri a e b della funzione somma vengono **copiati** i valori passati (a = n1, b = n2), quindi a e b sono due variabili indipendenti da n1 e n2.
La variabile s adesso conterrà il valore ritornato dalla funzione somma, in questo caso 9;

Esempi di funzioni:

```
int max(int a, int b){
    if(a > b) return a;
    return b;
}

long int pow(int base, int esp){
    long int tot = 1;
    for(int i = 0; i < esp; i++){
          Tot = tot * base;
    }
    return tot;
}

bool pari(int n){
    if(n%2 == 0)return true;
    return false;
}
```

Nel caso una funzione contenga più istruzione return, essa ritornerà solo il valore presente nel primo return che viene eseguito, in quanto una chiamata a return termina l'esecuzione di una funzione.
Esistono poi funzioni dette procedure che non ritornano nessun dato. In questo caso il loro valore di ritorno è void. Esempio:

```
void 1toN(int n);

int main(){
   1toN(8);          //In questo caso la funzione non è assegnata a nessuna variabile perchè non ha valore di ritorno
}

void 1toN(int n){                          //stampa i primi n numeri naturali
    for(int i=0; i<n; i++){
        printf("%d   ", i);
    }
}
```

A volte è poi necessario che una funzione ritorni più di un valore. In questo caso possiamo passare come parametri i puntatori alle variabile in cui salvare i valori. Prendiamo come esempio la funzione che calcoli le soluzione di una equazione di 2° grado:

```
void eq2(int a, int b, int c, float* x1, float* x2);

int main(){
  float x1, x2;
  eq2(1, 4, 6, &x1, &x2);
}

void eq2(int a, int b, int c, float* x1, float* x2){
    float delta = b*b – 4 * a * c;
    *x1 = (-b + sqrt(delta)) / 2 * a;
    *x2 = (-b - sqrt(delta)) / 2 * a;
}
```

Nel main, utilizzando l'operatore &, otteniamo i puntatori delle variabile x1 e x2 e le passiamo come parametri alla funzione.
Nella funzione, utilizzando l'operatore * sui puntatori passati, riusciamo a modificare i valori presenti agli indirizzi. Facendo così le variabili vengono modificate anche nel main. Più in generale è utile passare variabili attraverso il loro puntatore quando il loro valore deve essere modificato all'interno della funzione e questa modifica deve essere mantenuta anche nel main.

Come detto precedentemente, una variabile-vettore non è altro che un puntatore al primo elemento dell'array, quindi per passare un vettore ad una funzione basta mettere come paramente un puntatore dello stesso tipo dell'array.
Esempi:

```
int cerca(int* arr, int dim, int x);
void elimina(int* arr, int* dim, int pos);
void scambio(int* a, int* b);      //essendo scambio usato in ordina deve essere dichiarato prima di ordina
void ordina(int* arr, int dim);

int main(){
  int a[10];
  int dim = 6;
  int i = cerca(a, dim, 3);          //non bisogna mettere & davanti ad a perchè è già un puntatore
  elimina(a, &dim, 4);               //dovendo modificare dim passo il suo puntatore
  ordina(a, dim);
}

int cerca(int* arr, int dim, int x){
  int i = 0;
  while(i<dim && arr[i] != x) i++;
  if(i<dim) return i;
  return -1;
}

void elimina(int* arr, int* dim, int pos){
  for(int i = pos; i<*dim; i++){
```

```
    arr[i] = arr[i+1];
  }
  (*dim)--;
}

void scambio(int* a, int* b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

void ordina(int* arr, int dim){
  int min, temp;
  for(int i = 0; i<dim-1; i++){
    min = i;
    for(int j = i+1; j<dim; j++){
      if(arr[j]<arr[min]) min = j;
    }
    scambio(&arr[i], &arr[min]);
  }
}
```

Struct

Le struct permettono di definire nuovi tipi di dato a partire dai tipi di dato più semplici (bool, char, int, float e derivati). Per esempio la struct che rappresenta i dati di una persona potrebbe essere così definita:

```
typedef struct Persona{
        char nome[16], cognome[16];
        int età;
}Persona;
```

Persona è un nuovo tipo di dato formato da due array di 16 elementi e un intero. Una volta definito possiamo dichiarare variabili di tipo Persona e accedere ai singoli campi della struttura. Per accedere ai singoli campi si utilizza nome_variabile.nome_campo come segue:

```
int main(){
  Persona p;
  scanf("%d", &p.età);
  puts(p.nome);
  puts(p.cognome);
}
```

Anche per le struct è possibile passare le variabili attraverso il puntatore per poterne modificare i valori all'interno delle funzioni

```
void carica(Persona* p);

int main(){
  Persona p;
  carica(&p);
}
```

```
void carica(Persona* p){
   scanf("%d", &(*p).età);
   puts((*p).nome);
   puts((*p).cognome);
}
```

Per accedere ai campi di un puntatore ad una struct bisogna prima ottenere il valore della struct all'indirizzo (*p) e poi accedere al campo con l'operatore . seguito dal nome del campo.
Per semplificare le cose è stato creato l'operatore -> che permette di accedere direttamente ad un campo di una struct a partire dal suo puntatore. La funzione sopra può essere riscritta nel seguente modo:

```
void carica(Persona* p){
   scanf("%d", &(p->età));
   puts(p->nome);
   puts(p->cognome);
}
```

L'operatore -> può essere solo usato per accedere ai campi di una struct a partire da un suo puntatore.

Come ogni tipo di dato è possibile definire anche array di struct. Ad ogni indice dell'array si avrà un'istanza della struct con la quale si può accedere a tutti i campi presenti.

Esercizio:
Realizzare un programma che permetta la gestione di libri all'interno di uno scaffale. Ogni libro è caratterizzato da un id numerico, titolo, autore, data di pubblicazione, numero di pagine. L'utente potrà eseguire le seguenti operazioni:

- inserimento di un nuovo libro (max 16)
- rimozione di un libro dato il suo id
- ricerca di un libro per id
- stampa di tutti i libri presenti
- ordinamento dei libri per id

Soluzione nella pagina seguente.

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

#define MAX_DIM		32

typedef struct Data{
	int dd, mm, yy;
}Data;

typedef struct Libro{
	int id, pags;
	char titolo[32], autore[32];
	Data pubblicazione;
}Libro;


int search(Libro* l, int dim, int id);
void insert(Libro* l, int* dim);
void remove(Libro* l, int* dim, int pos);
void print(Libro* l, int dim);
void sort(Libro* l, int dim);


int main(){
	Libro l[MAX_DIM];
	int dim = 0;
	char scelta;

	int id, pos;
	do{
		printf("1. Inserisci\n");
		printf("2. Rimuovi\n");
		printf("3. Cerca\n");
		printf("4. Stampa\n");
		printf("5. Ordina\n");
		printf("6. Esci\n");
		printf("Cosa vuoi fare?  ");
		fflush(stdin);
		scelta = getchar();

		switch(scelta){
			case '1':
				insert(l, &dim);
				break;
			case '2':
				printf("Quale id vuoi cancellare?  ");
				scanf("%d", &id);
				pos = search(l, dim, id);
				if(pos != -1){
					remove(l, &dim, pos);
				}
				else{
					printf("Libro non trovato con id = %d\n", id);
				}
```

```
                        break;
                case '3':
                        printf("Quale id vuoi vedere?  ");
                        scanf("%d", &id);
                        pos = search(l, dim, id);
                        if(pos != -1){
                                print(&l[pos], 1);
                        }
                        else{
                                printf("Libro non trovato con id = %d\n", id);
                        }
                        break;
                case '4':
                        print(l, dim);
                        break;
                case '5':
                        sort(l, dim);
                        break;
        }
        system("pause");
        system("cls");

    }while(scelta != '6');


    return 0;
}

int search(Libro* l, int dim, int id){
    int i = 0;
    while(i < dim && l[i].id != id)i++;
    if(i < dim)return i;
    return -1;
}

void insert(Libro* l, int* dim){
    if((*dim) + 1 <= MAX_DIM){
        printf("Inserisci id  ");
        scanf("%d", &l[*dim].id);
        printf("Inserisci titolo  ");
        fflush(stdin);
        gets(l[*dim].titolo);
        printf("Inserisci autore  ");
        fflush(stdin);
        gets(l[*dim].autore);
        printf("Inserisci pagine  ");
        scanf("%d", &l[*dim].pags);
        printf("Inserisci data (gg/mm/yyyy)  ");
        scanf("%d%d%d",
            &l[*dim].pubblicazione.dd,
            &l[*dim].pubblicazione.mm,
            &l[*dim].pubblicazione.yy
        );
        (*dim)++;
    }
```

```
	else{
		printf("Array pieno\n");
	}
}

void remove(Libro* l, int* dim, int pos){
	for(int i = pos; i < *dim; i++){
		l[i] = l[i+1];
	}
	(*dim)--;
}

void print(Libro* l, int dim){
	for(int i = 0; i < dim; i++){
		printf("%d %s %s %d %d/%d/%d\n",
			l[i].id,
			l[i].autore,
			l[i].titolo,
			l[i].pags,
			l[i].pubblicazione.dd, l[i].pubblicazione.mm, l[i].pubblicazione.yy
		);
	}
}

void sort(Libro* l, int dim){
	int min;
	Libro temp;
	for(int i = 0; i < dim-1; i++){
		min = i;
		for(int j = i+1; j < dim; j++){
			if(l[j].id < l[min].id){
				min = j;
			}
		}

		temp = l[i];
		l[i] = l[min];
		l[min] = temp;
	}
}
```