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));
}
```