PAROLE - C++

Ci sono due modi per rappresentare un testo in C++: o creare un *array di char*, o usare il tipo predefinito *string.*

Il tipo string è più moderno, ha più funzionalità ed è più facile da utilizzare ma esiste solo in C++ e non in C. Gli array di char sono l'unico modo utilizzabile per rappresentare le parole in C ma questo metodo è anche utlizzabile (seppur sconsigliato) in C++.

1 - IL TIPO STRING

- Bisogna aggiungere l'header `#include<string>`.
- Si usa la parola `string` per dichiarare una variabile di tipo string.
- Esempi di dichiarazione:

```
string parola;      // dichiara una variabile di tipo string, che si chiama parola
cin >> parola;      // attende l'input dell'utente e memorizza quanto inserito nella variabile parola
string parola_due = "ciao"; // dichiara una di tipo string, che si chiama parola_due e ci assegna il valore "ciao"
```

- La funzione `str.length()` ci permette di ottenere la lunghezza della stringa str.
- Possiamo utilizzare la notazione classica degli array per accedere ai singoli caratteri della stringa.
- La funzione `str1.compare(str2)` restituisce 0 se str1 e str2 contengono gli stessi caratteri.
- Esempi d'uso:

```
string miaparola = "ciao";
cout << miaparola.length() << endl;   // stampa la lunghezza di "ciao" ovvero 4.
cout << miaparola[0] << endl;         // stampa il primo carattere di "ciao" ovvero 'c'.
cout << miaparola[1] << endl;         // stampa il secondo carattere di "ciao" ovvero 'i'.
cout << miaparola[2] << endl;         // stampa 'a'
cout << miaparola    << endl;         // stampa "ciao"
miaparola[1] = 'a';                   // assegna 'c' al secondo carattere di miaparola
miaparola[2] = 'i';                   // assegna 'i' al terzo carattere di miaparola
cout << miaparola;                    // Cosa stampa?

string altraParola = "cane";
cout << altraParola.compare("cane")     << endl; // stampa 0
cout << altraParola.compare(miaparola) << endl; // stampa un valore diverso da 0
cout << miaparola.compare(altraParola) << endl; // stampa un valore diverso da 0
```

Riassunto concetti base.

- **`string`** è un tipo da usare per le variabili (come int, float ecc…)
- **`#include<string>`** è l'header che ci permette di utilizzare string.

Se str è una variabile di tipo string abbiamo che:

- **`str.length()`** restituisce la lunghezza della parola contenuta nella variabile str.
- **`str[0]`** è il primo carattere della parola contenuta nella variabile str.
- **`str[i]`** è l'i - esimo carattere della parola contenuta nella variabile str.
- **`str[ str.length() – 1 ]`** è l'ultimo carattere della parola contenuta in str.

2 – ARRAY DI CHAR

In C non esiste un vero e proprio tipo per rappresentare le parole, perciò, possiamo pensare alle parole come una sequenza di lettere ovvero un array di tipo char.

- Bisogna aggiungere l'header `#include<string.h>`.
- Si usa la sintassi classica di un array per la dichiarazione ma esiste anche una sintassi semplificata.
- Esempi di dichiarazione:

```
char parola[100];   // dichiara un array di caratteri con 100 posizioni libere (99 in realtà)
cin >> parola;      // attende l'input dell'utente e memorizza quanto inserito nella variabile parola
char parola_due[5] = "ciao"; // dichiaro parola_due e ci assegno il valore "ciao"
char parola_tre[5] = {'c', 'i', 'a', 'o', '\0'};   // altra sintassi di assegnazione
```

Quindi trattandosi di un array dobbiamo dimensionare a priori l'array stesso.

Come è possibile notare, esistono due modi per l'assegnazione di un valore. La sintassi classica di un array come in *parola_tre* oppure una sintassi più pulita come in *parola_due*.

Come mai a *parola_due* abbiamo assegnato 5 celle e non 4 come la lunghezza della parola "ciao"?

Utilizzando gli array di char per memorizzare le parole è necessario che l'ultimo carattere sia il carattere nullo ovvero abbia valore '\0'; questo permette al programma di sapere dove finisce la parola. Se utilizziamo la sintassi di dichiarazione come in *parola_due* il carattere nullo verrà aggiunto automaticamente ma dobbiamo ricordarci di avere una cella libera altrimenti il compilatore darà errore. Se utilizziamo la sintassi classica degli array il carattere nullo non verrà aggiunto automaticamente e dobbiamo ricordarci noi di inserirlo.

- La funzione `strlen(str)` ci permette di ottenere la lunghezza della stringa contenuta nell' array di char str.
- Possiamo utilizzare la notazione classica degli array per accedere ai singoli caratteri dell'array di char, proprio come abbiamo fatto con il tipo string.
- La funzione `strcmp(str1, str2)` restituisce 0 se str1 e str2 contengono gli stessi caratteri
- Esempi d'uso:

```
char miaparola[10] = "ciao";
cout << strlen(miaparola) << endl;    // stampa la lunghezza di "ciao" ovvero 4.
cout << miaparola[0] << endl;         // stampa il primo carattere di "ciao" ovvero 'c'.
cout << miaparola[1] << endl;         // stampa il secondo carattere di "ciao" ovvero 'i'.
cout << miaparola[2] << endl;         // stampa 'a'
cout << miaparola    << endl;         // stampa "ciao"
miaparola[1] = 'a';                   // assegna 'c' al secondo carattere di miaparola
miaparola[2] = 'i';                   // assegna 'i' al terzo carattere di miaparola
cout << miaparola << endl;            // Cosa stampa?

char altraParola[10] = "cane";
cout << strcmp(altraParola, "cane")     << endl; // stampa 0
cout << strcmp(altraParola, miaparola) << endl; // stampa un valore diverso da 0
cout << strcmp(miaparola, altraParola) << endl; // stampa un valore diverso da 0
```

3 – ALCUNI ESEMPI ED ESERCIZI

Esempio: Come potremmo calcolare la lunghezza di una parola contenuta in un array di char senza utilizzare la funzione strlen?

Ci basta fare un ciclo while scorrendo nell'array fintanto che i caratteri considerati sono diversi dal carattere nullo.

```
int myStrlen(char s[]){
	int lunghezza = 0;
	while(s[lunghezza] != '\0'){
		lunghezza ++;
	}
	return lunghezza;
}
```

Esempio: Come potremmo dire se due parole contenute in due array di char contengono gli stessi caratteri senza utilizzare la funzione strcmp?

Se le parole hanno lunghezza diversa allora non avranno gli stessi caratteri. Se hanno la stessa lunghezza devo controllare con un ciclo carattere per carattere: se trovo un carattere diverso le parole hanno gli stessi caratteri, se alla fine del ciclo non ho trovato caratteri diversi, le parole hanno gli stessi caratteri.

```
int myStrcmp(char s1[], char s2[]){
	int l1 = strlen(s1);
	int l2 = strlen(s2);

	if(l1 != l2){
		return 1;
	}else{
		for(int i = 0; i < l1; i++){
			if(s1[i] != s2[i]){
				return 1;
			}
		}
		return 0;
	}
}
```

Esempio: Come potremmo calcolare se una parola è palindroma? Vediamo il caso di una parola contenuta in una stringa, ma il caso con gli array di char è analogo.

Ci basta controllare che ogni carattere in posizione i sia uguale al carattere in posizione speculare n-i-1.

Ad esempio nella parola "ciaooaic" il carattere in posizione 0 'c' è uguale al carattere in posizione 8-0-1=7, il carattere in posizione 1 'i' è uguale al carattere in posizione 8-1-1=6, il carattere in posizione 2 'a' è uguale al carattere in posizione 8-2-1=5 ed infine il carattere in posizione 3 'o' è uguale al carattere in posizione 8-3-1=4; deduciamo quindi che "ciaooaic" è una parola palindroma.

```
bool verificaParolaPalindroma(string parola){
	int n = parola.length();
	for (int i = 0; i <= n/2; i++) {
		if(parola [i] != parola [n-i -1]) {
			return false;
		}
	}
	return true;
}
```