Arrays

How to copy an array

When using arrays in C++ there are a few things to be aware of with respect to working with the entire array. An error many new users run into is attempting to assign one array to another, which generates error: invalid array assignment. The sketch below demonstrates this error.

```
char arrA[ 5 ] = { 1, 2, 3, 4, 5 };
char arrB[ 5 ];

void setup(){
  arrB = arrA;  //Will not compile!
}

void loop() {}
```

The closest valid syntax is using a pointer to the arrays first element, or using a reference to it. The usage of an arrays reference and pointer is covered in an FAQ here: [SOON! promise].

To achive what is desired in the code above, we need to utilize a loop to copy each element. There are a number of ways this can be achived, and the C++ core provides an a few additional methods to get the desired result.

- **A *for* loop.**

  ```
  char arrA[ 5 ] = { 1, 2, 3, 4, 5 };
  char arrB[ 5 ] = {};

  void setup(){

    for( int i = 0 ; i < 5 ; ++i ){
      arrB[ i ] = arrA[ i ];
    }
  }

  void loop() {}
  ```

- **A *while* loop.**

  ```
  char arrA[ 5 ] = { 1, 2, 3, 4, 5 };
  char arrB[ 5 ] = {};

  void setup(){

    char i = 0;

    while( i < 5 ){
      arrB[ i ] = arrA[ i ];
      ++i;
  ```

```
  }
}

void loop() {}
```

- **A *do while* loop.**

```
char arrA[ 5 ] = { 1, 2, 3, 4, 5 };
char arrB[ 5 ] = {};

void setup(){

  char i = 0;

  do{
    arrB[ i ] = arrA[ i ];
  }while( ++i < 5 );
}

void loop() {}
```

- **A C++11 *ranged for* loop.**

```
char arrA[ 5 ] = { 1, 2, 3, 4, 5 };
char arrB[ 5 ] = {};

void setup(){

  char i = 0;

  for( char &el : arrB ){
    el = arrA[ i++ ];
  }
}
```

An additional method to complete the copy is a standard function which the Arduino environment includes, it is memcpy(). Below is a sketch which accomplishes the same task as the three methods posted above using a call to memcpy().

```
char arrA[ 5 ] = { 1, 2, 3, 4, 5 };
char arrB[ 5 ] = {};

void setup(){
  memcpy( arrB, arrA, 5 );
}

void loop() {}
```

If you are using a char, unsigned char, or byte array there is a way to accomplish the copy without knowing the length of the data. Typically a string is a null-terminated character array, which means an array of characters ending with a null character or zero. However the data does not need to be readable characters, it can be any binary data ending in zero. Some functions are designed to look for the null character, strcpy() is one of them. Here is the skecth modified for strcpy(), notice the extra element in each array. The only catch is the terminating character, if it appears in the middle of an array, the string based functions will consider it as the end of the data.

```
char arrA[ 6 ] = { 1, 2, 3, 4, 5, 0 };
char arrB[ 6 ] = {};

void setup(){
  strcpy( arrB, arrA );
}

void loop() {}
```

So as you can see there are many ways to achieve the same result. The three loops mentioned at the top are a very basic framework for doing any operation on an entire array. If you have a need to access your array in reverse order, please see this article to avoid a common pitfall. I hope this has been a help, there are more array FAQ's to come!

Unique solution ID: #1029
Author: Christopher Andrews
Last update: 2014-04-02 06:16