**Arrays as parameters**

At some point, we may need to pass an array to a function as a parameter. In C/C++, it is not possible to pass the entire block of memory represented by an array to a function directly as an argument. But what can be passed instead is its address. In practice, this has almost the same effect, and it is a much faster and more efficient operation.

To accept an array as parameter for a function, the parameters can be declared as the array type, but with empty brackets, omitting the actual size of the array. For example:

|  |  |  |
| --- | --- | --- |
|  | *void* procedure (*int* arg[]) |  |

This function accepts a parameter of type "array of int" called arg. In order to pass to this function an array declared as:

|  |  |  |
| --- | --- | --- |
|  | *int* myarray [40]; |  |

it would be enough to write a call like this:

|  |  |  |
| --- | --- | --- |
|  | procedure (myarray); |  |

Here you have a complete example:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | *// arrays as parameters*  *#include <iostream>*  *using* *namespace* std;  *void* printarray (*int* arg[], *int* length) {  *for* (*int* n=0; n<length; ++n)  cout << arg[n] << ' ';  cout << '\n';  }  *int* main ()  {  *int* firstarray[] = {5, 10, 15};  *int* secondarray[] = {2, 4, 6, 8, 10};  printarray (firstarray,3);  printarray (secondarray,5);  } | 5 10 15  2 4 6 8 10 |  |

In the code above, the first parameter (int arg[]) accepts any array whose elements are of type int, whatever its length. For that reason, we have included a second parameter that tells the function the length of each array that we pass to it as its first parameter. This allows the for loop that prints out the array to know the range to iterate in the array passed, without going out of range.

In a function declaration, it is also possible to include multidimensional arrays. The format for a tridimensional array parameter is:

|  |  |  |
| --- | --- | --- |
|  | base\_type[][depth][depth] |  |

For example, a function with a multidimensional array as argument could be:

|  |  |  |
| --- | --- | --- |
|  | *void* procedure (*int* myarray[][3][4]) |  |

Notice that the first brackets [] are left empty, while the following ones specify sizes for their respective dimensions. This is necessary in order for the compiler to be able to determine the depth of each additional dimension.

In a way, passing an array as argument always loses a dimension. The reason behind is that, for historical reasons, arrays cannot be directly copied, and thus what is really passed is a pointer. This is a common source of errors for novice programmers.

|  |  |
| --- | --- |
| // in C example  #include <stdio.h>  void printarray (int arg[],int lenglh);  int main ()  {  int firstarray[] = {1, 2, 3,4,5},i;  printarray (firstarray,5);  printf("After function calling: ");  for(i=0;i<5;i++)  printf("%d ",firstarray[i]);  return 0;  }  void printarray (int arg[], int length) {  int n;  for (n=0; n<length; ++n)  {  printf("%d ", arg[n]);  arg[n]=arg[n]+1;  }  printf("\n");  } | 1 2 3 4 5  After function calling: 2 3 4 5 6 |