Advanced Programming

Recursive Functions and Arrays with Pointers in C++

Dr. Zeyad Safaa Younus Saffawi

Recursive Functions

- A **recursive function** is a function that **calls itself** during its execution.
- Recursion is useful when solving problems that can be divided into **smaller sub-problems of the same type**.
- A recursive function must contain:
  1. **Base Case** → condition to stop recursion
  2. **Recursive Case** → the function calls itself

Example: Factorial Function

- The factorial of a number is defined as:

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 1$$

Example:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Iterative Implementation (Using Loop)

```
int factorial_iter(int n)
{
   int i, f = 1;
   for(i = 1; i <= n; i++)
      f *= i;
   return f;
}
```

Explanation:

- The **loop multiplies numbers** from **1 to n**
- The **result** is stored in **f**

Recursive Implementation

```
int factorial_rec(int n)
{
if (n == 1)
        return 1;

else
        return n * factorial_rec(n - 1);
}
```

Explanation:

- **Base case**:

  n == 1 → return 1

- **Recursive case**:

  n * factorial_rec(n-1)

Example:

factorial_rec(5)
= 5 × factorial_rec(4)
= 5 × 4 × factorial_rec(3)
= 5 × 4 × 3 × factorial_rec(2)
= 5 × 4 × 3 × 2 × factorial_rec(1)
= 120

Pointer to an Array

- In C++, the **array name acts as a pointer to the first element of the array**.
- Example:

```
double balance[10];
```

- The name balance represents the address of the first element:

```
balance = &balance[0]
```

- Example of pointer declaration:

```
double *p;
double balance[10];
p = balance;
```

- Now p points to the first element of the array.
- Array elements can be accessed using pointer arithmetic.
- Example:

***(p + 0) → balance[0]**
***(p + 1) → balance[1]**
***(p + 2) → balance[2]**

Accessing Array Elements Using Pointer

Example :

```
#include <iostream>
using namespace std;
int main()
{
   double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
   double *p;
   p = balance;
   cout << "Array values using pointer" << endl;

   for(int i = 0; i < 5; i++)
   {
      cout << *(p + i) << endl;
   }
   cout << "Array values using balance as address" << endl;
   for(int i = 0; i < 5; i++)
   {
      cout << *(balance + i) << endl;
   }
   return 0;
}
```

- Both methods give the **same result** because the array name behaves like a pointer.

Passing Arrays to Functions

- C++ does not allow passing an **entire array directly** to a function.
- Instead, we pass the **address of the first element**.
- There are three ways to declare array parameters.

Way 1: Using Pointer

```
void myFunction(int *param)
{
}
```

Way 2: Sized Array

```
void myFunction(int param[10])
{
}
```

Way 3: Unsized Array

```
void myFunction(int param[])
{
}
```

- All three methods are treated by the compiler as **pointer to the array**.

Example: Calculate Average of an Array

```
#include <iostream>
using namespace std;
double getAverage(int arr[], int size);
{
int i, sum = 0;double avg;
for(i = 0; i < size; i++)
{
sum += arr[i];
}
avg = double(sum) / size;
return avg;
}
int main()
{
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;
   avg = getAverage(balance, 5);
   cout << "Average value is: " << avg << endl;
   return 0;
}
```


Output:

Average value is: 214.4

Returning Arrays from Functions

- C++ cannot return an entire array directly.
- Instead, the function returns a **pointer to the array**.
- Example declaration:

```
int * myFunction()
{

}
```

note:

- If we return an array from a function, it should be declared as **static**.

Example: Returning Random Numbers

```
#include <iostream>
#include <ctime>
using namespace std;
int* getRandom()
{
   static int r[10];
   srand((unsigned)time(NULL));
   for(int i = 0; i < 10; i++)
   {
      r[i] = rand();
   }
   return r;
}
int main()
{
   int *p;
   p = getRandom();
   for(int i = 0; i < 10; i++)
   {
      cout << *(p + i) << endl;
   }
   return 0;
}
```

This program:

1. Generates **10 random numbers**
2. Stores them in an array
3. Returns the array using a **pointer**

References

- Gaddis, T. (2014). *Starting out with C++: From control structures through objects* (8th ed.). Pearson.
- Soulié, J. (2007, April 24). C++ language tutorial. cplusplus.com.
- Tutorials Point. (n.d.). *Learn C++ programming language*. Tutorials Point.