```cpp
#include <iostream>
void EndMessage(); // Some prototype - ignore if you are reading notes.
// Example function
void Print(int count, char ch) {
    using namespace std;
    for (int i = 0; i < count; ++i) {
        cout << ch;
    }
    cout << endl;
}

// Invoke this function in main() using a normal function call
int main() {
    Print(5, '#');
    return 0;
}

// Invoking the function through a function pointer
int main(){
    void (*Print_ptr)(int, char) = &Print;
    // Note that the NAME of the function is the address of the function, and Print_ptr is a function pointer variable, so we initialise the function pointer variable with the address of the function.
    // The ampersand is hence optional.
    void (*Print_ptr)(int, char) = Print; // this works too!

    // Invoking the function
    (*Print_ptr)(5, '#'); // this works
    Print_ptr(5, '#'); // this works - as if the function pointer is a normal function.
    // So, you see that we don't depend on the
```

```
    name of the function to call the function,


    /// Applications of Function pointers:
      atexit()
    // You are allowed to invoke some function in
      your program just before it terminates. So
      we'll use atexit() to print a message just
      before the program exits.
    //atexit(function ADDRESS) --> is the syntax
      for the function call.
    atexit(EndMessage); // see definition below
    // So this function atexit registers the
      pointer to the function internally,and the
      function pointer is invoked after main().
      So it doesn't matter where you define
      atexit.
}

// function that Atexit calls:
void EndMessage(){
    using namespace std;
    cout << "End of program" << endl;
}

```