```cpp
#include <iostream>
using namespace std;

using ProcessId = int;

struct Process
{
    Process *previousProcess = nullptr;
    Process *nextProcess = nullptr;
    ProcessId id;

    Process(ProcessId id) : id(id) {}
};

class Scheduler
{
    Process *head = nullptr;

public:
    /*
          Adds a process with id = pid to the end of the linked list
    */
    void add_process(ProcessId pid)
    {
        Process *newProcess = new Process(pid);
        if (head == nullptr)
        {
            head = newProcess;
        }
        else
        {
            Process *curr = head;
            while (curr->nextProcess != nullptr)
            {
                curr = curr->nextProcess;
            }
            curr->nextProcess = newProcess;
            newProcess->previousProcess = curr;
        }
    }

    /*
          Deletes the process with id == pid
    */
    void delete_process(ProcessId pid)
    {
        Process *curr = head;
        while (curr != nullptr)
        {
            if (curr->id == pid)
            {
                if (curr->previousProcess != nullptr)
                {
                    curr->previousProcess->nextProcess = curr->nextProcess;
                }
                else
                {
                    head = curr->nextProcess;
                }

                if (curr->nextProcess != nullptr)
                {
                    curr->nextProcess->previousProcess = curr-
>previousProcess;
                }

                delete curr;
                break;
            }
            curr = curr->nextProcess;
        }
    }

    /*
          Adds a process with id == newId after the process with id == pid
    */
```

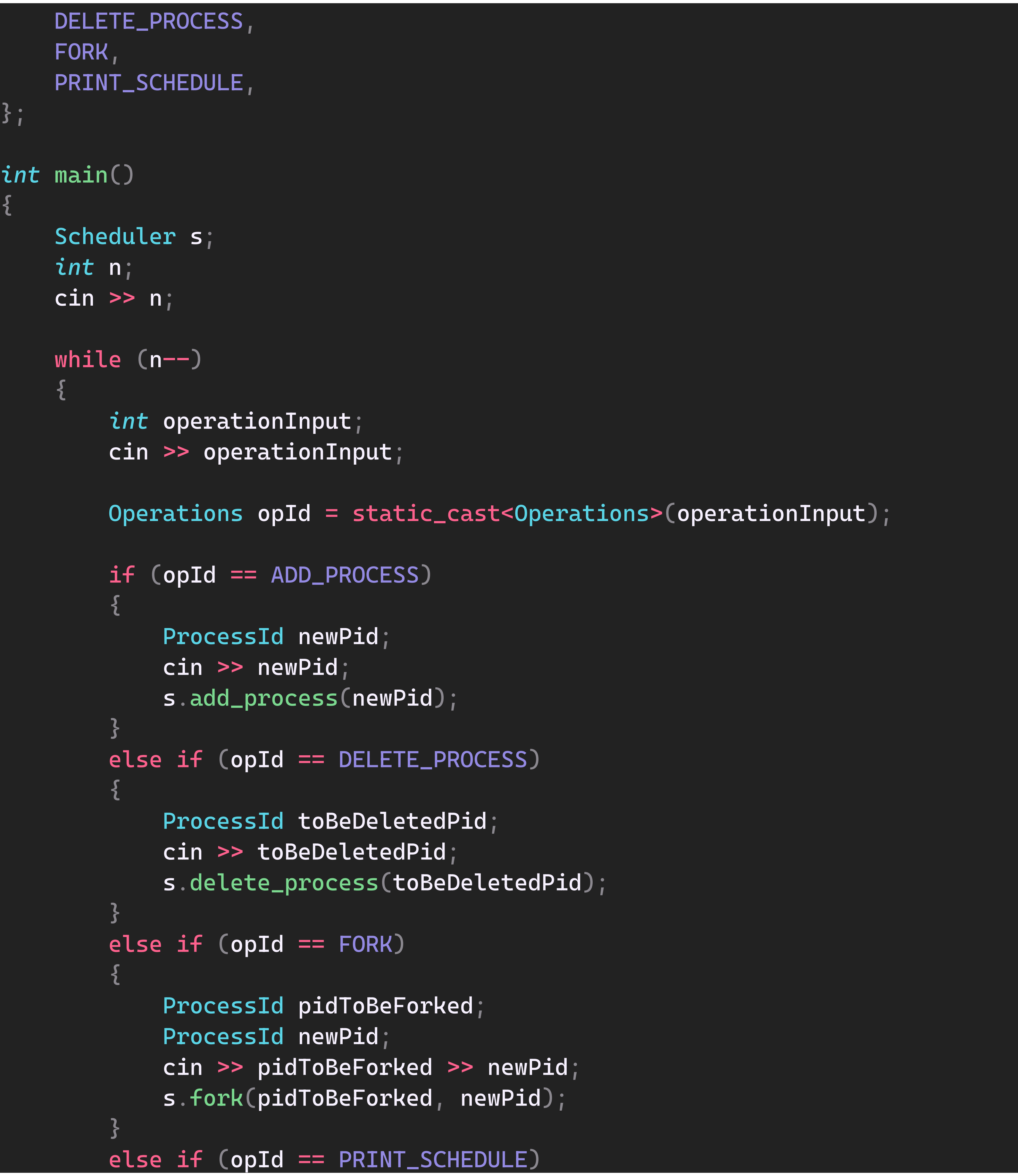
```cpp
    void fork(ProcessId pid, ProcessId newId)
    {
        Process *curr = head;
        while (curr != nullptr)
        {
            if (curr->id == pid)
            {
                Process *newProcess = new Process(newId);
                newProcess->previousProcess = curr;
                newProcess->nextProcess = curr->nextProcess;

                if (curr->nextProcess != nullptr)
                {
                    curr->nextProcess->previousProcess = newProcess;
                }

                curr->nextProcess = newProcess;
                break;
            }
            curr = curr->nextProcess;
        }
    }

    void print_schedule()
    {
        Process *curr = head;
        while (curr != nullptr)
        {
            cout << curr->id << " ";
            curr = curr->nextProcess;
        }
        cout << endl;
    }
};

enum Operations
{
    ADD_PROCESS,
```



```cpp
    DELETE_PROCESS,
    FORK,
    PRINT_SCHEDULE,
};

int main()
{
    Scheduler s;
    int n;
    cin >> n;

    while (n--)
    {
        int operationInput;
        cin >> operationInput;

        Operations opId = static_cast<Operations>(operationInput);

        if (opId == ADD_PROCESS)
        {
            ProcessId newPid;
            cin >> newPid;
            s.add_process(newPid);
        }
        else if (opId == DELETE_PROCESS)
        {
            ProcessId toBeDeletedPid;
            cin >> toBeDeletedPid;
            s.delete_process(toBeDeletedPid);
        }
        else if (opId == FORK)
        {
            ProcessId pidToBeForked;
            ProcessId newPid;
            cin >> pidToBeForked >> newPid;
            s.fork(pidToBeForked, newPid);
        }
        else if (opId == PRINT_SCHEDULE)
        {
            s.print_schedule();
        }
    }
}
```