#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*

*\*/*

*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,

    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();

        }

    }

}