Name: Sahil

UID: 22BCS10928

Section: 901_KPIT(B)

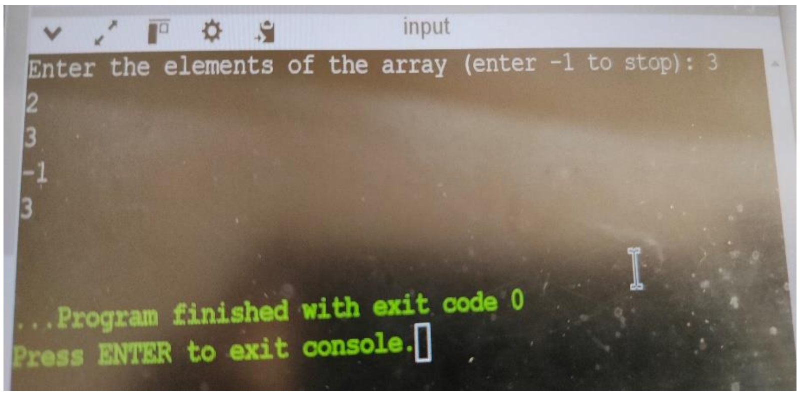
1) Majority Elements

Given an array nums of size n, return the majority element.
The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Answer:

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

int majorityElement(vector<int>& nums) {
   int count = 0, majority = 0;
   for (int num : nums) {
      if (count == 0) majority = num;
      count += (num == majority) ? 1 : -1;
   }
   return majority;
}
int main() {
   vector<int> nums;
   int num;
   cout << "Enter the elements of the array (enter -1 to stop): ";
   while (cin >> num && num != -1) {
      nums.push_back(num);
   }
   cout  << majorityElement(nums) << endl;
   return 0;
}
```

OuTpUt:



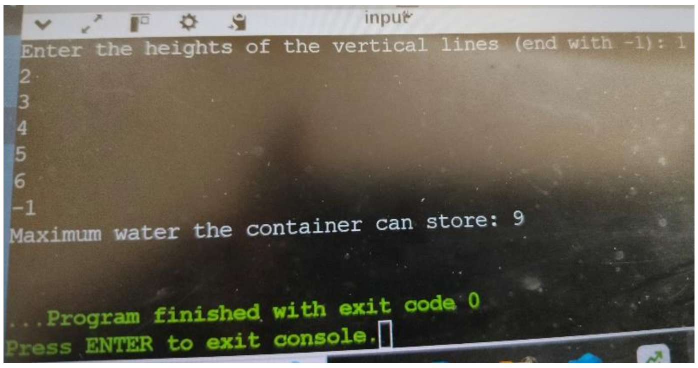
```
Enter the elements of the array (enter -1 to stop): 3
2
3
-1
3


...Program finished with exit code 0
Press ENTER to exit console.
```

2) Container With Most Water

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).
Find two lines that together with the x-axis form a container, such that the container contains the most water.
Return the maximum amount of water a container can store.
Notice that you may not slant the container.
Answer:

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

int maxArea(vector<int>& height) {
    int left = 0, right = height.size() - 1, maxWater = 0;
    while (left < right) {
        int width = right - left;
        int h = min(height[left], height[right]);
        maxWater = max(maxWater, width * h);
        if (height[left] < height[right])
            ++left;
        else
            --right;
    }
    return maxWater;
}

int main() {
    vector<int> height;
    int value;
    cout << "Enter the heights of the vertical lines (end with -1): ";
    while (cin >> value && value != -1) {
        height.push_back(value);
    }
    cout  << maxArea(height) << endl;
    return 0;
}
```

OuTPut:



```
Enter the heights of the vertical lines (end with -1): 1
2
3
4
5
6
-1
Maximum water the container can store: 9


...Program finished with exit code 0
Press ENTER to exit console.
```

3) Jump Game II

You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

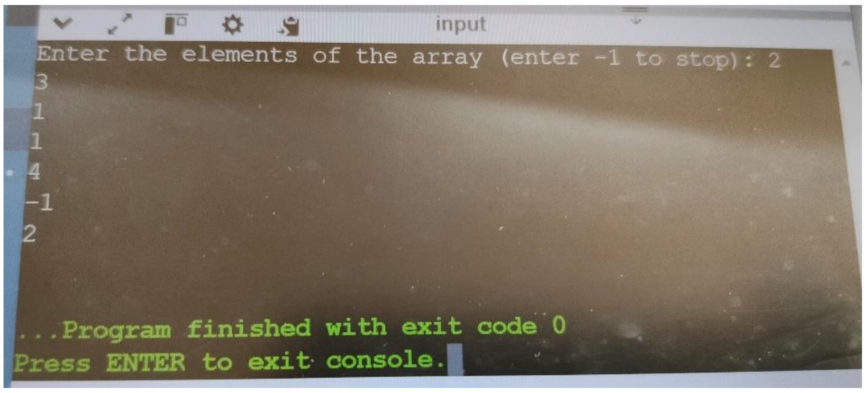
0 <= j <= nums[i] and
i + j < n
Return the minimum number of jumps to reach nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

Answer:

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

int jump(vector<int>& nums) {
   int n = nums.size(), jumps = 0, farthest = 0, end = 0;
   for (int i = 0; i < n - 1; ++i) {
      farthest = max(farthest, i + nums[i]);
      if (i == end) {
         jumps++;
         end = farthest;
      }
   }
```

```
    return jumps;
}

int main() {
    vector<int> nums;
    int num;

    cout << "Enter the elements of the array (enter -1 to stop): ";
    while (cin >> num && num != -1) {
        nums.push_back(num);
    }

    cout  << jump(nums) << endl;
    return 0;
}
```

OuTPuT:



```
Enter the elements of the array (enter -1 to stop): 2
3
1
1
4
-1
2


...Program finished with exit code 0
Press ENTER to exit console.
```

4) Maximum Number of Groups Getting Fresh Donuts

There is a donuts shop that bakes donuts in batches of batchSize. They have a rule where they must serve all of the donuts of a batch before serving any donuts of the next batch. You are given an integer batchSize and an integer array groups, where groups[i] denotes that there is a group of groups[i] customers that will visit the shop. Each customer will get exactly one donut.

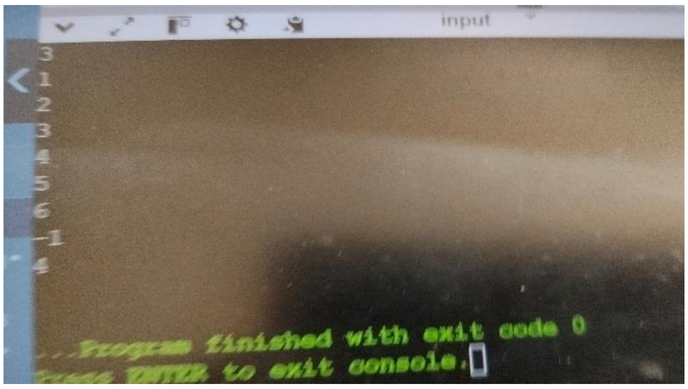
When a group visits the shop, all customers of the group must be served before serving any of the following groups. A group will be happy if they all get fresh donuts. That is, the first customer of the group does not receive a donut that was left over from the previous group.

You can freely rearrange the ordering of the groups. Return the maximum possible number of happy groups after rearranging the groups.

Answer:

```
#include <iostream>
```

```
#include <vector>
#include <unordered_map>
using namespace std;

int maxHappyGroups(int batchSize, vector<int>& groups) {
    unordered_map<int, int> count;
    int happyGroups = 0;

    for (int group : groups) {
        int remainder = group % batchSize;
        if (remainder == 0) {
            happyGroups++;
        } else {
            count[remainder]++;
        }
    }

    for (int i = 1; i < batchSize; ++i) {
        int complement = batchSize - i;
        if (count[i] > 0 && count[complement] > 0) {
            int matches = min(count[i], count[complement]);
            happyGroups += matches;
            count[i] -= matches;
            count[complement] -= matches;
        }
    }

    happyGroups += count[0] / batchSize;

    return happyGroups;
}

int main() {
    int batchSize;
    cin >> batchSize;

    vector<int> groups;
    int groupSize;

    while (cin >> groupSize && groupSize != -1) {
        groups.push_back(groupSize);
    }

    cout << maxHappyGroups(batchSize, groups) << endl;
    return 0;
}
```



```
3
1
2
3
4
5
6
-1
4

...Program finished with exit code 0
Press ENTER to exit console.
```

5) Maximum Twin Sum of a Linked List

In a linked list of size n, where n is even, the ith node (0-indexed) of the linked list is known as the twin of the (n-1-i)th node, if 0 <= i <= (n / 2) - 1.

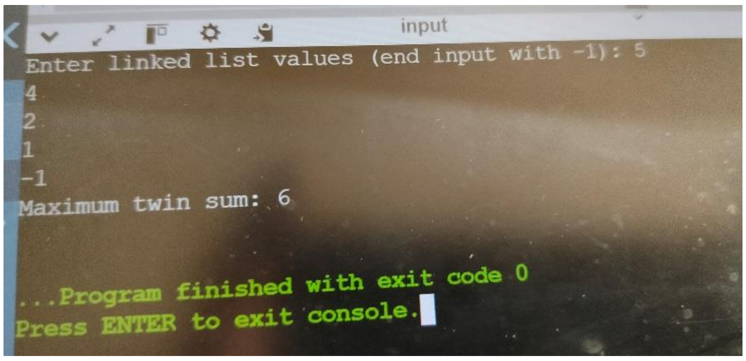
- For example, if n = 4, then node 0 is the twin of node 3, and node 1 is the twin of node 2. These are the only nodes with twins for n = 4.

The twin sum is defined as the sum of a node and its twin.
Given the head of a linked list with even length, return the maximum twin sum of the linked list.

Answer:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

ListNode* createList(const vector<int>& values) {
    ListNode* head = new ListNode(values[0]);
    ListNode* current = head;
    for (int i = 1; i < values.size(); ++i) {
        current->next = new ListNode(values[i]);
        current = current->next;
    }
    return head;
}

int pairSum(ListNode* head) {
    vector<int> values;
    ListNode* current = head;
    while (current) {
        values.push_back(current->val);
        current = current->next;
    }

    int maxTwinSum = 0;
    int n = values.size();
    for (int i = 0; i < n / 2; ++i) {
        maxTwinSum = max(maxTwinSum, values[i] + values[n - 1 - i]);
    }

    return maxTwinSum;
}

int main() {
    vector<int> values;
    int value;

    cout << "Enter linked list values (end input with -1): ";
    while (cin >> value && value != -1) {
        values.push_back(value);
    }

    ListNode* head = createList(values);
    cout << "Maximum twin sum: " << pairSum(head) << endl;

    return 0;
}
```

OuTpUt:



```
Enter linked list values (end input with -1): 5
4
2
1
-1
Maximum twin sum: 6


...Program finished with exit code 0
Press ENTER to exit console.
```