CS101 End Sem Practice

Autumn 2024-25

Shubham Raj, Aayush Borkar and Balaji Karedla

References

```
void func(int& ref){
  ref = ref * 2;
}

int main(){
  int a = 10;
  func(a);
  cout << a << endl;
}
```

Pointers

```
void func(int* ptr){
  *ptr = *ptr * 2;
}

int main(){
  int a = 10;
  func(&a);
  cout << a << endl;
}
```

So why do we need pointers?

Arrays (*arr* is just a pointer to the first element)

```
void print(int arr[], int n){
  for(int i=0; i<n; i++){
    cout << arr[i] << ' ';
  }
}

int main(){
  int arr[] = {7, 3, 2, 5, 1};
  print(arr, 5);
}
```

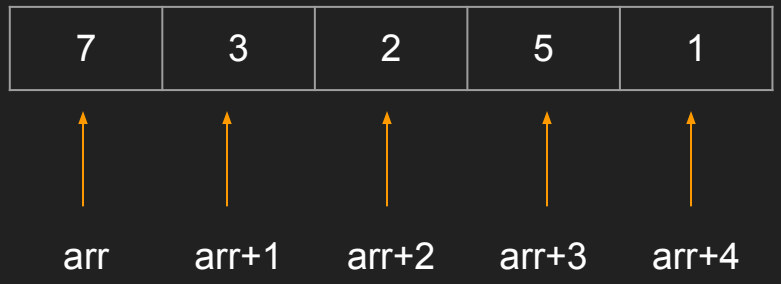
```
void print(int *arr, int n){
  for(int i=0; i<n; i++){
    cout << arr[i] << ' ';
  }
}

int main(){
  int arr[] = {7, 3, 2, 5, 1};
  print(arr, 5);
}
```

But why does arr[i] still works?

writing
`arr[i]`

is the same as writing
`*(arr+i)`




```
void print(int *arr, int n){
  for(int i=0; i<n; i++){
    cout << arr[i] << ' ';
  }
}

int main(){
  int arr[] = {7, 3, 2, 5, 1};
  print(arr, 5);
}
```

`arr[i]` is the same as writing `i[arr]`

Return an array

```
int* copyArr(int arr[]){
  int newArr[5];
  for(int i=0; i<5; i++){
    newArr[i] = arr[i];
  }
  return newArr;
}
int main(){
  int a[5] = {4, 3, 5, 7, 9};
  int *b;
  b = copyArr(a);
  for(int i=0; i<5; i++){
    cout << b[i] << ' ' ;
  }
}
```

a) 4 3 5 7 9
b) 0 0 0 0 0
c) Prints 5 garbage values
d) Unpredictable Behaviour

Return an array

```
int* copyArr(int arr[]){
  int newArr[5];
  for(int i=0; i<5; i++){
    newArr[i] = arr[i];
  }
  return newArr;
}
int main(){
  int a[5] = {4, 3, 5, 7, 9};
  int *b;
  b = copyArr(a);
  for(int i=0; i<5; i++){
    cout << b[i] << ' ' ;
  }
}
```

But why?

The new array in the function was deleted as soon as the function ended

a) 4 3 5 7 9
b) 0 0 0 0 0
c) Prints 5 garbage values
d) Unpredictable Behaviour

How can we fix this?

```
int* copyArr(int arr[]){
  int newArr[5];
  for(int i=0; i<5; i++){
    newArr[i] = arr[i];
  }
  return newArr;
}
int main(){
  int a[5] = {4, 3, 5, 7, 9};
  int *b;
  b = copyArr(a);
  for(int i=0; i<5; i++){
    cout << b[i] << ' ' ;
  }
}
```

Dynamic Memory Allocation

```
int* copyArr(int arr[]){
  int* newArr = new int[5];
  for(int i=0; i<5; i++){
    newArr[i] = arr[i];
  }
  return newArr;
}
int main(){
  int a[5] = {4, 3, 5, 7, 9};
  int *b;
  b = copyArr(a);
  for(int i=0; i<5; i++){
    cout << b[i] << ' ' ;
  }
  delete[] b;
}
```

Question - Pointers and Address

```
int main(){
  int arr[5] = {1, 2, 3, 4, 5};
  for(int i=0; i<6; i++){
    cout << arr+i << ' ';
  }
  cout << endl;

  cout << arr << endl;
  cout << &arr << endl;
  cout << arr+1 << endl;
  cout << (&arr)+1 << endl;
}
```

Output

0x61fef8 0x61fefc 0x61ff00 0x61ff04 0x61ff08 0x61ff0c

Predict the next 4 lines of output

a	0x61fef8 Random Address 0x61fefc 0x61fefc	b	0x61fef8 0x61fef8 0x61fefc 0x61fefc
c	0x61fef8 0x61fef8 0x61fefc 0x61ff0c	d	0x61fef8 Random Address 0x61fefc Random Address

Solution

```
int main(){
  int arr[5] = {1, 2, 3, 4, 5};
  for(int i=0; i<6; i++){
    cout << arr+i << ' ';
  }
  cout << endl;

  cout << arr << endl;
  cout << &arr << endl;
  cout << arr+1 << endl;
  cout << (&arr)+1 << endl;
}
```

Output

0x61fef8 0x61fefc 0x61ff00 0x61ff04 0x61ff08 0x61ff0c

Predict the next 4 lines of output

a	0x61fef8 Random Address 0x61fefc 0x61fefc	b	0x61fef8 0x61fef8 0x61fefc 0x61fefc
c	0x61fef8 0x61fef8 0x61fefc 0x61ff0c	d	0x61fef8 Random Address 0x61fefc Random Address

Question - What if I pass that array in a function

```
void func(int* arr){
  cout << arr << endl;
  cout << &arr << endl;
  cout << arr+1 << endl;
  cout << (&arr)+1 << endl;
}

int main(){
  int arr[5] = {1, 2, 3, 4, 5};
  for(int i=0; i<6; i++){
    cout << arr+i << ' ';
  }

  cout << endl;
  func(arr);
}
```

Output

0x61fef8 0x61fefc 0x61ff00 0x61ff04 0x61ff08 0x61ff0c

Predict the next 4 lines of output

a	0x61fef8 Random Address 0x61fefc 0x61fefc	b	0x61fef8 0x61fef8 0x61fefc 0x61fefc
c	0x61fef8 0x61fef8 0x61fefc 0x61ff0c	d	0x61fef8 Random Address 0x61fefc Random Address

Question - What if I pass that array in a function

```
void func(int* arr){
  cout << arr << endl;
  cout << &arr << endl;
  cout << arr+1 << endl;
  cout << (&arr)+1 << endl;
}

int main(){
  int arr[5] = {1, 2, 3, 4, 5};
  for(int i=0; i<6; i++){
    cout << arr+i << ' ';
  }

  cout << endl;
  func(arr);
}
```

Output

0x61fef8 0x61fefc 0x61ff00 0x61ff04 0x61ff08 0x61ff0c

Predict the next 4 lines of output

a	0x61fef8 Random Address 0x61fefc 0x61fefc	b	0x61fef8 0x61fef8 0x61fefc 0x61fefc
c	0x61fef8 0x61fef8 0x61fefc 0x61ff0c	d	0x61fef8 Random Address 0x61fefc Random Address

Vectors

A vector in C++ is like a resizable array.

How can we initialize vectors?

```
vector<int> first;
vector<int> second (10);
vector<int> third (10, 5);
vector<int> fourth (third);
```

Vector : Member Functions

```cpp
vector<int> arr;

arr.resize(5, 10);

arr[3];

arr.size();
arr.empty();    // arr.size() == 0

arr.front();    // arr[0]
arr.back();     // arr[arr.size()-1]

arr.push_back(8);
arr.pop_back();
arr.clear();
```

Pairs

A data structure consisting of two member elements (both can be of different types).
Member variables can be accessed using “first” and “second”.

```
std::pair<int, int> a = {5, 6};
auto b = std::make_pair(8, "hello");
std::cout << a.first << " " << a.second << std::endl; // prints 5 6
std::cout << b.first << " " << b.second << std::endl; // prints 8 hello
```

Question - Matrix Transpose

Fill in the blanks. Given below is a function to compute the transpose of a given 2D matrix.

```
vector<vector<int>> transpose(const vector<vector<int>> &M) {
    vector<vector<int>> MT(_____, vector<int>(_____));
    for (int i = 0; i < _____; i++)
        for (int j = 0; j < _____; j++)
            MT[j][i] = M[i][j];
    return MT;
}
```

Solution - Matrix Transpose

Fill in the blanks. Given below is a function to compute the transpose of a given 2D matrix.

```
vector<vector<int>> transpose(const vector<vector<int>> &M) {
    vector<vector<int>> MT(M[0].size(), vector<int>(M.size()));
    for (int i = 0; i < _____; i++)
        for (int j = 0; j < _____; j++)
            MT[j][i] = M[i][j];
    return MT;
}
```

Solution - Matrix Transpose

Fill in the blanks. Given below is a function to compute the transpose of a given 2D matrix.

```
vector<vector<int>> transpose(const vector<vector<int>> &M) {
    vector<vector<int>> MT(M[0].size(), vector<int>(M.size()));
    for (int i = 0; i < M.size(); i++)
        for (int j = 0; j < M[0].size(); j++)
            MT[j][i] = M[i][j];
    return MT;
}
```

Question - Two Sum

Given a vector of n integers and a target value, find the number of pairs of integers in the vector whose sum is equal to target.

For example, if v={1, 2, 3, 4, 5, 6} and target=6, there are 2 pairs as follows:

- 1 5
- 2 4

Note: Naive solution with nested for loops will not pass the test cases with larger n

```
#include <map>
#include <vector>

using std::map;
using std::vector;

int countPairs(vector<int> &v, int
target) {
    // TODO: Complete this function
    return 0; // dummy return
}
```

Solution - Two Sum by brute force

Given a vector of n integers and a target value, find the number of pairs of integers in the vector whose sum is equal to target.

For example, if v={1, 2, 3, 4, 5, 6} and target=6, there are 2 pairs as follows:

- 1 5
- 2 4

Note: Naive solution with nested for loops will not pass the test cases with larger n

```
int countPairs(vector<int> &v, int
target) {
    int count = 0;
    for (int i=0; i<v.size(); i++)
        for (int j=0; j<v.size();
j++)
            if (v[i] + v[j] ==
target)
                count++;
    return count;
}
```

Solution - Two Sum using maps

Given a vector of n integers and a target value, find the number of pairs of integers in the vector whose sum is equal to target.

For example, if v={1, 2, 3, 4, 5, 6} and target=6, there are 2 pairs as follows:

- 1 5
- 2 4

Note: Naive solution with nested for loops will not pass the test cases with larger n

```
int countPairs(vector<int> &v, int
target) {
    map<int, int> freq;
    int count = 0;
    for (int x : v) {
        int complement = target - x;
        count += freq[complement];
        freq[x]++;
    }
    return count;
}
```

What if complement or x don't already exist in freq? Any new key is initialized with a value of 0 by default

Question - Dice Combinations

Count the number of ways to construct sum n by throwing a dice one or more times. Each throw produces an outcome between 1 and 6.

For example, if n=3, there are 4 ways:

- 1+1+1
- 1+2
- 2+1
- 3

```
int diceCmb(int n) {
    if (_____)
        return 1;
    if (_____)
        return 0;
    return _____;
}
```

Solution - Dice Combinations

Count the number of ways to construct sum n by throwing a dice one or more times. Each throw produces an outcome between 1 and 6.

For example, if n=3, there are 4 ways:

- 1+1+1
- 1+2
- 2+1
- 3

```
int diceCmb(int n) {
    if (_____)
        return 1;
    if (_____)
        return 0;
    return diceCmb(n - 1)
        + diceCmb(n - 2)
        + diceCmb(n - 3)
        + diceCmb(n - 4)
        + diceCmb(n - 5)
        + diceCmb(n - 6);
}
```

Solution - Dice Combinations

Count the number of ways to construct sum n by throwing a dice one or more times. Each throw produces an outcome between 1 and 6.

For example, if n=3, there are 4 ways:

- 1+1+1
- 1+2
- 2+1
- 3

```
int diceCmb(int n) {
    if (n == 0)
        return 1;
    if (n < 0)
        return 0;
    return diceCmb(n - 1)
        + diceCmb(n - 2)
        + diceCmb(n - 3)
        + diceCmb(n - 4)
        + diceCmb(n - 5)
        + diceCmb(n - 6);
}
```

Solution - Dice Combinations with Memoization

```
#include <vector>
std::vector<int> memo;
// memo.resize(n, -1); // to be done in main
int diceCmb(int n) {
    if (n == 0)
        return 1;
    if (n < 0)
        return 0;
    if (memo[n] != -1)
        return memo[n];
    memo[n] = diceCmb(n - 1) + diceCmb(n - 2) + diceCmb(n - 3) +
              diceCmb(n - 4) + diceCmb(n - 5) + diceCmb(n - 6);
    return memo[n];
}
```

New vector to act as the memo

If n is found in memo, i.e., if memo[n] is not -1, then our answer is memo[n]

Otherwise, compute the answer as usual and store it as memo[n]

Classes

```
class Account{
  public:
    string accHolderName;
    int balance;
    vector<string> transactions;
};
int main(){
  Account acc;
  acc.accHolderName = "Alex";
  acc.balance = 0;
  acc.transactions.push_back("Init with Rs 0")
}
```

Constructors

```
class Account{
  private:
    string accHolderName;
    int balance;
    vector<string> transactions;
  public:
    Account(string holderName){
      accHolderName = holderName;
      balance = 0;
      transactions.push_back("Init with Rs 0");
    }
};

int main(){
  Account acc ("Alex");
}
```

```
struct Node{
    int val;
    Node* next;

    Node(int value): val(value) {};
};

int counter = 53;

void addNode(Node &first){
    Node second (counter++);

    first.next = &second;
}
```

Predict the output !!

a) 53 54	c) 53 0
b) 54 55	d) Unpredictable Behaviour

```
int main(){
    Node head (counter++);

    addNode(head);

    cout << head.val << endl;
    cout << head.next->val << endl;
}
```

```
struct Node{
    int val;
    Node* next;

    Node(int value): val(value) {};
};

int counter = 53;

void addNode(Node &first){
    Node* second = new Node (counter++);

    first.next = second;
}
```

Solution:

Make the node dynamically

Ideally the head node should also be defined dynamically
And we should delete the memory we took at the end of the code

```
int main(){
    Node head (counter++);

    addNode(head);

    cout << head.val << endl;
    cout << head.next->val << endl;
}
```

Question - Swapping function

Choose ALL the correct options to fill in the blanks

```
void swap(__ blank A__) {
   __ blank B __;
   temp = a;
   a = b;
   b = temp;
}

main_program {
   int x = 0; y = 1;
   swap(__ blank C __);
   cout << x << " " << y << endl;
}
```

A) int a, int b
   int temp
   x, y

B) int &a, int &b
   int *temp
   x, y

C) int *a, int *b
   int *temp
   &x, &y

D) int *a, int *b
   int temp
   &x, &y

Solution - Swapping function

Answer (b)

```
void swap(__ blank A__) {
    __ blank B __;
    temp = a;
    a = b;
    b = temp;
}

main_program {
    int x = 0; y = 1;
    swap(__ blank C __);
    cout << x << " " << y << endl;
}
```

int &a, int &b
int *temp
x, y

Question - Linked List

Implement a Linked List with methods to insert, remove and clear the list
Also, three different constructors (default, construct with an array and a vector), a copy constructor and a destructor for the List

Also, implement the += operator and the ostream << operator for the Linked List

Solution - Linked List

```
#include <vector>
#include <iostream>

struct Node {
   int data;
   Node* next;
};

class LinkedList{
   public:
   Node* head;
   Node* tail;
   int size;

   LinkedList();
   LinkedList(int data[], int len);
   LinkedList(std::vector<int> data);
   LinkedList(const LinkedList &list);
   ~LinkedList();

   void insert(int data);
   void remove(int data);
   void clear();

   LinkedList &operator+=(const LinkedList &list);
   friend std::ostream &operator<<(std::ostream &os,
LinkedList &list);
};
```

Solution - Linked List

```
void LinkedList::insert(int data){
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;
    if(head == nullptr){
        head = newNode;
        tail = newNode;
    }else{
        tail->next = newNode;
        tail = newNode;
    }
    size++;
}
```

Solution - Linked List

```
void LinkedList::remove(int data) {
    Node* current = head;
    Node* previous = nullptr;
    while (current != nullptr) {
        if (current->data == data) {
            if (previous == nullptr) {
                head = current->next;
            } else {
                previous->next = current->next;
            }
            size--;
            return;
        }
        previous = current;
        current = current->next;
    }
}
```

Solution - Linked List

```
void LinkedList::clear() {
    Node* current = head;
    Node* next = nullptr;
    while (current != nullptr) {
        next = current->next;
        delete current;
        current = next;
    }
    head = nullptr;
    tail = nullptr;
    size = 0;
}
```

Solution - Linked List

```
LinkedList::LinkedList(){
   head = nullptr;
   tail = nullptr;
   size = 0;
}

LinkedList::LinkedList(int data[], int len) {
   head = nullptr;
   tail = nullptr;
   size = 0;
   for (int i = 0; i < len; i++) {
       insert(data[i]);
   }
}
```

Solution - Linked List

```
LinkedList::LinkedList(vector<int> data) {
    head = nullptr;
    tail = nullptr;
    size = 0;
    for (int i = 0; i < data.size(); i++) {
        insert(data[i]);
    }
}

LinkedList::LinkedList(const LinkedList &list) {
    head = nullptr;
    tail = nullptr;
    size = 0;
    Node* current = list.head;
    while (current != nullptr) {
        insert(current->data);
        current = current->next;
    }
}
```

Solution - Linked List

```
LinkedList::~LinkedList(){
    clear();
}

LinkedList &LinkedList::operator+=(const LinkedList &list) {
    LinkedList temp = list;
    if (head == nullptr) {
        head = temp.head;
        tail = temp.tail;
    } else {
        tail->next = temp.head;
        tail = temp.tail;
    }
    temp.head = nullptr;
    temp.tail = nullptr;
    size += temp.size;
    return *this;
}
```

Solution - Linked List

```
ostream &operator<<(ostream &os, LinkedList &list) {
    Node* current = list.head;
    while (current != nullptr) {
        os << current->data << " ";
        current = current->next;
    }
    return os;
}
```

Something we want you guys to explore….

```
int main() {
1     const int* a;
2     int* const b;
3     int x = 0;
4     int* const d = &x;
5     const int* e = &x;
6     const int* const c = &x;
7     const int* const f;
8     int y = 0;
9     *d = 3;
10    *e = 4;
11    *c = 5;

12     d = &y;
13     e = &y;
14     c = &y;
}
```