Bài 1

- **Danh sách liên kết (Linked List):**

+ Cấu trúc: Lớp LinkedList sử dụng danh sách liên kết để lưu trữ các đối tượng Appointment. Mỗi phần tử trong danh sách liên kết là một Node, chứa một đối tượng Appointment và một con trỏ next tới phần tử tiếp theo.

+ Lý do sử dụng:

- **Thêm vào danh sách theo thứ tự**: Danh sách liên kết là lựa chọn phù hợp khi cần thêm phần tử vào danh sách mà không cần quan tâm đến vị trí của nó, miễn sao các phần tử được sắp xếp theo một thứ tự nhất định. Trong trường hợp này, lịch hẹn cần được thêm vào và sắp xếp theo giờ, và danh sách liên kết cho phép việc này thực hiện một cách linh hoạt và hiệu quả.
- **Dễ dàng xóa phần tử**: Danh sách liên kết dễ dàng thực hiện thao tác xóa phần tử mà không cần phải thay đổi quá nhiều phần còn lại của cấu trúc dữ liệu. Khi xóa một cuộc hẹn, ta chỉ cần điều chỉnh các con trỏ (next) mà không cần phải dịch chuyển các phần tử khác như trong mảng.
- **Khả năng mở rộng linh hoạt**: Danh sách liên kết cho phép dễ dàng thêm vào hoặc xóa phần tử mà không cần phải thay đổi kích thước cố định của mảng. Điều này rất tiện lợi khi số lượng cuộc hẹn có thể thay đổi liên tục.

```
#include <iostream>

#include <string>

// Cấu trúc lưu trữ thông tin một cuộc hẹn

struct Appointment

{

  std::string datetime;   // Ngày giờ cuộc hẹn (yyyyMMddhhmm)

  std::string patientName; // Tên bệnh nhân

  std::string doctorName;  // Tên bác sĩ

  std::string notes;      // Ghi chú về cuộc hẹn


  // Hàm khởi tạo cấu trúc Appointment
```

```
    Appointment(std::string dt, std::string pName, std::string dName, std::string n)
        : datetime(dt), patientName(pName), doctorName(dName), notes(n) {}
};

// Cấu trúc nút trong danh sách liên kết
struct Node
{
    Appointment data;  // Dữ liệu cuộc hẹn
    Node *next;       // Con trỏ đến nút tiếp theo trong danh sách

    // Hàm khởi tạo nút với dữ liệu cuộc hẹn
    Node(Appointment a) : data(a), next(nullptr) {}
};

// Lớp quản lý danh sách liên kết các cuộc hẹn
class LinkedList
{
public:
    LinkedList() : head(nullptr) {}  // Hàm khởi tạo danh sách rỗng
    ~LinkedList() // Hàm hủy danh sách liên kết
    {
        while (head) // Duyệt và xóa các nút trong danh sách
        {
            Node *temp = head;
            head = head->next;
            delete temp;
```

```
    }
  }

  // Thêm một cuộc hẹn vào danh sách theo thứ tự thời gian
  void addAppointment(const Appointment &a)
  {
    Node *newNode = new Node(a);  // Tạo một nút mới chứa cuộc hẹn
    if (!head || head->data.datetime > a.datetime) // Nếu danh sách rỗng hoặc đầu danh sách có thời gian lớn hơn cuộc hẹn
    {
      newNode->next = head;  // Chèn vào đầu danh sách
      head = newNode;       // Cập nhật đầu danh sách
      return;
    }
    Node *temp = head;
    while (temp->next && temp->next->data.datetime < a.datetime) // Duyệt qua danh sách để tìm vị trí chèn
    {
      temp = temp->next;
    }
    newNode->next = temp->next; // Chèn cuộc hẹn vào vị trí tìm được
    temp->next = newNode;      // Cập nhật con trỏ của nút trước đó
  }

  // Xóa một cuộc hẹn theo tên bệnh nhân
  void removeAppointment(const std::string &patientName)
  {
```

```
    Node *temp = head, *prev = nullptr;
    while (temp && temp->data.patientName != patientName) // Duyệt danh sách để tìm bệnh nhân cần xóa
    {
        prev = temp;
        temp = temp->next;
    }
    if (!temp) // Nếu không tìm thấy bệnh nhân
        return;
    if (!prev) // Nếu là nút đầu tiên
        head = temp->next;
    else
        prev->next = temp->next; // Cập nhật con trỏ của nút trước đó
    delete temp; // Xóa nút
}

// Hiển thị danh sách lịch hẹn theo khung giờ
void displayAppointments() const
{
    Node *temp = head;
    if (!temp) // Nếu danh sách rỗng
    {
        std::cout << "Khong co lich hen\n";
        return;
    }
    std::cout << "Danh sach lich hen theo khung gio:\n";
```

```
    std::string currentHour; // Biến lưu trữ khung giờ hiện tại

    // Duyệt qua từng cuộc hẹn trong danh sách
    while (temp)
    {
      if (temp->data.datetime.length() >= 12) // Kiểm tra định dạng datetime hợp lệ
      {
        std::string hour = temp->data.datetime.substr(8, 2); // Lấy giờ từ datetime (yyyyMMddhhmm)

        // Nếu khung giờ chưa được hiển thị, hiển thị nó
        if (hour != currentHour)
        {
          currentHour = hour;
          std::cout << "\nKhung gio " << currentHour << ":00 - " << currentHour << ":59\n";
        }

        // Hiển thị thông tin cuộc hẹn
        std::cout << temp->data.datetime << " - "
              << temp->data.patientName << " - "
              << temp->data.doctorName << " - "
              << temp->data.notes << "\n";
      }
      else
      {
```

```
            std::cout << "Lich hen co datetime khong hop le: " << temp->data.datetime << "\n";
        }

        temp = temp->next; // Di chuyển đến nút tiếp theo trong danh sách
    }
  }

  // Tìm kiếm cuộc hẹn theo tên bệnh nhân
  Appointment *searchAppointment(const std::string &patientName)
  {
    Node *temp = head;
    while (temp)
    {
      if (temp->data.patientName == patientName) // Nếu tìm thấy bệnh nhân
        return &temp->data;  // Trả về địa chỉ cuộc hẹn
      temp = temp->next;  // Di chuyển đến nút tiếp theo
    }
    return nullptr; // Nếu không tìm thấy
  }

private:
  Node *head; // Con trỏ đến đầu danh sách liên kết
};

int main()
{
```

```
LinkedList ll; // Tạo danh sách liên kết để quản lý các cuộc hẹn
int choice;
do
{
   std::cout << "\nMenu:\n";
   std::cout << "1. Them lich hen\n";
   std::cout << "2. Hien thi danh sach lich hen\n";
   std::cout << "3. Tim kiem lich hen theo ten benh nhan\n";
   std::cout << "4. Xoa lich hen\n";
   std::cout << "5. Thoat\n";
   std::cout << "Chon: ";
   std::cin >> choice;
   std::cin.ignore();

   if (choice == 1)
   {
      // Nhập thông tin cuộc hẹn và thêm vào danh sách
      std::string datetime, patientName, doctorName, notes;
      std::cout << "Nhap ngay gio (yyyyMMddhhmm): ";
      std::getline(std::cin, datetime);
      std::cout << "Nhap ten benh nhan: ";
      std::getline(std::cin, patientName);
      std::cout << "Nhap ten bac si: ";
      std::getline(std::cin, doctorName);
      std::cout << "Nhap ghi chu: ";
      std::getline(std::cin, notes);
```

```
    ll.addAppointment(Appointment(datetime, patientName, doctorName, notes));
}
else if (choice == 2)
{
    // Hiển thị danh sách lịch hẹn
    ll.displayAppointments();
}
else if (choice == 3)
{
    // Tìm kiếm lịch hẹn theo tên bệnh nhân
    std::string searchName;
    std::cout << "Nhap ten benh nhan: ";
    std::getline(std::cin, searchName);
    Appointment *found = ll.searchAppointment(searchName);
    if (found)
    {
        std::cout << found->datetime << " - "
                  << found->patientName << " - "
                  << found->doctorName << " - "
                  << found->notes << "\n";
    }
    else
    {
        std::cout << "Khong tim thay\n";
    }
}
```

```
    else if (choice == 4)
    {
        // Xóa lịch hẹn theo tên bệnh nhân
        std::string removeName;
        std::cout << "Nhap ten benh nhan can xoa: ";
        std::getline(std::cin, removeName);
        ll.removeAppointment(removeName);
    }
  } while (choice != 5);
  return 0;
}
```

1. Khởi tạo chương trình: Lớp LinkedList sẽ tạo một danh sách liên kết (head là con trỏ đầu danh sách). Danh sách này dùng để quản lý các cuộc hẹn, Khi chương trình bắt đầu, danh sách này là rỗng (head là nullptr).

2. Menu chính: Chương trình liên tục hiển thị một menu với 5 lựa chọn:

Dưới đây là chi tiết các lựa chọn:

3. Lựa chọn 1: Thêm lịch hẹn

- Khi bạn chọn **1**, chương trình sẽ yêu cầu bạn nhập các thông tin của cuộc hẹn:

+ Ngày giờ (yyyyMMddhhmm): Ví dụ: 202502151030 (ngày 15 tháng 2 năm 2025, giờ 10:30).

+ Tên bệnh nhân.

+ Tên bác sĩ.

+ Ghi chú.

- Sau khi bạn nhập xong, chương trình sẽ tạo một đối tượng Appointment với thông tin bạn nhập và gọi hàm addAppointment để thêm cuộc hẹn vào danh sách liên kết.

- Cuộc hẹn sẽ được thêm vào danh sách theo thứ tự thời gian (sắp xếp từ sớm đến muộn).

4. Lựa chọn 2: Hiển thị danh sách lịch hẹn

- Khi bạn chọn **2**, chương trình sẽ duyệt qua danh sách các cuộc hẹn và hiển thị chúng theo **khung giờ**.

- Lịch hẹn sẽ được nhóm theo giờ (ví dụ: từ 10:00 đến 10:59), và nếu có nhiều cuộc hẹn cùng giờ, chúng sẽ được liệt kê trong cùng khung giờ.

- Nếu danh sách không có cuộc hẹn nào, chương trình sẽ hiển thị thông báo "Khong co lich hen".

5. Lựa chọn 3: Tìm kiếm lịch hẹn theo tên bệnh nhân

- Khi bạn chọn **3**, chương trình yêu cầu bạn nhập tên bệnh nhân.

- Sau đó, chương trình sẽ duyệt qua danh sách các cuộc hẹn và tìm kiếm cuộc hẹn có tên bệnh nhân bạn nhập.

- Nếu tìm thấy, chương trình sẽ hiển thị thông tin cuộc hẹn của bệnh nhân đó. Nếu không tìm thấy, chương trình sẽ thông báo "Khong tim thay".

6. Lựa chọn 4: Xóa lịch hẹn theo tên bệnh nhân

- Khi bạn chọn **4**, chương trình yêu cầu bạn nhập tên bệnh nhân cần xóa.

- Chương trình sẽ duyệt qua danh sách các cuộc hẹn và tìm kiếm cuộc hẹn của bệnh nhân đó. Nếu tìm thấy, nó sẽ xóa cuộc hẹn đó khỏi danh sách.

- Nếu không tìm thấy, chương trình sẽ không làm gì và quay lại menu.

7. Lựa chọn 5: Thoát

- Khi bạn chọn **5**, chương trình sẽ dừng lại và thoát.

8. Quá trình làm việc

- Chương trình sẽ tiếp tục hiển thị menu sau mỗi lựa chọn cho đến khi bạn chọn **5** để thoát.

- Trong suốt quá trình này, danh sách các cuộc hẹn sẽ được lưu trữ trong danh sách liên kết, và các thao tác như thêm, tìm kiếm, xóa sẽ được thực hiện trên danh sách này.

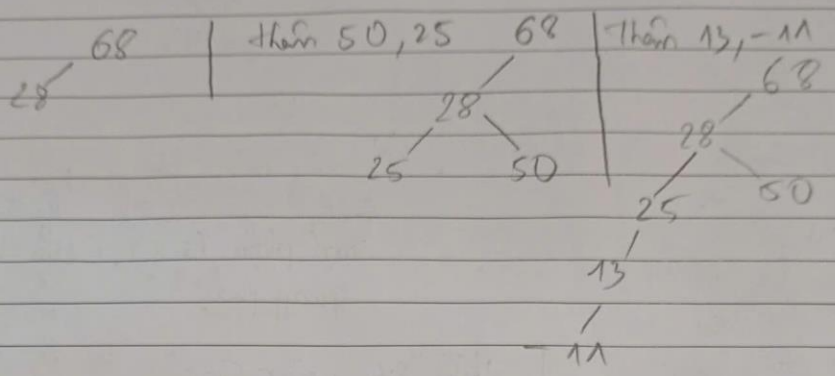
Bài 2:

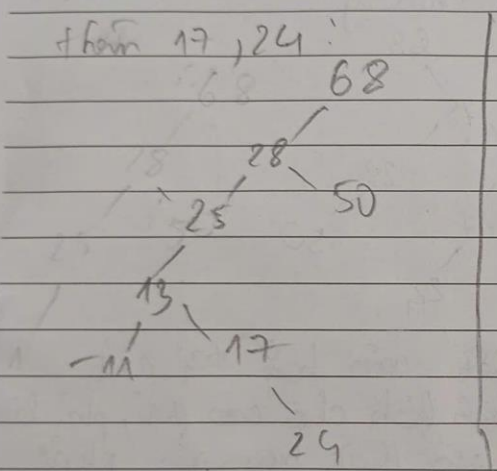
Mo Tu We Th Fri Sa Su Date ... / ... / ...

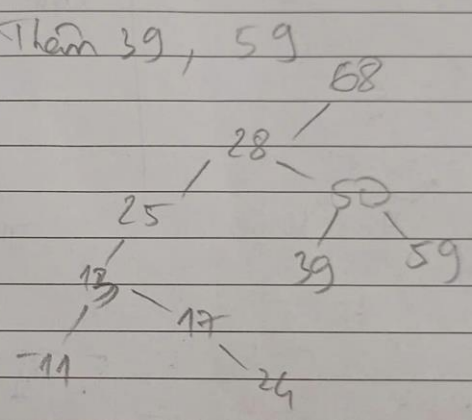
Câu 2:

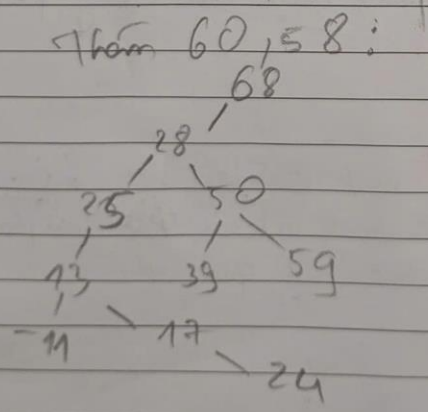
68, 28, 50, 25, 13, -11, 12, 24, 39, 59, 60, 58

a)





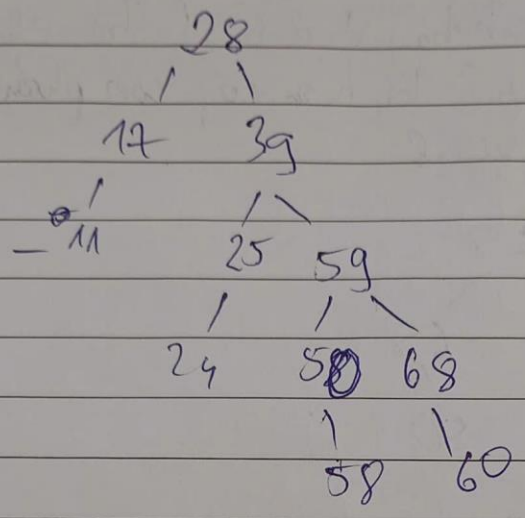




Mo Tu We Th Fri Sa Su

Memo No.....
Date.....

Cây AVL sau khi xoá nút x = 13



c)

```
#include <stdio.h>

// Định nghĩa cấu trúc của cây nhị phân tìm kiếm (BST)
typedef struct Node
{
   int key;
   struct Node *left, *right;
} Node;

// Hàm tạo nút mới bằng cách cấp phát bộ nhớ tĩnh (cách thay thế không dùng malloc)
Node *createNode(int key)
{
   static Node nodes[100]; // Giả sử cây sẽ có tối đa 100 nút
   static int nodeIndex = 0;

   if (nodeIndex >= 100)
   {
      return NULL; // Không đủ bộ nhớ để tạo nút mới
   }

   nodes[nodeIndex].key = key;
   nodes[nodeIndex].left = nodes[nodeIndex].right = NULL;
   return &nodes[nodeIndex++];
}

// Hàm chèn một giá trị vào cây BST
```

```
Node *insert(Node *root, int key)
{
   if (root == NULL)
   {
      return createNode(key);
   }

   if (key < root->key)
   {
      root->left = insert(root->left, key);
   }
   else if (key > root->key)
   {
      root->right = insert(root->right, key);
   }

   return root;
}

// Hàm kiểm tra một số có phải là số nguyên tố không
int isPrime(int n)
{
   if (n <= 1)
      return 0;
   for (int i = 2; i * i <= n; i++)
   {
```

```
        if (n % i == 0)
        {
            return 0;
        }
    }
    return 1;
}

// Hàm tìm nút có key là số nguyên tố lớn nhất ở nhánh con bên trái của nút gốc
int findMaxPrimeInLeftSubtree(Node *root)
{
    if (root == NULL || root->left == NULL)
    {
        return -1; // Không tìm thấy
    }

    Node *current = root->left;
    int maxPrime = -1;

    while (current != NULL)
    {
        if (isPrime(current->key) && current->key > maxPrime)
        {
            maxPrime = current->key;
        }
        current = current->right;
```

```
    }

    return maxPrime;
}

// Hàm tính tổng giá trị của tất cả các nút có key là số chẵn và có 1 nhánh con bên phải
int sumEvenNodesWithRightChild(Node *root)
{
    if (root == NULL)
    {
        return 0;
    }

    int sum = 0;
    if (root->key % 2 == 0 && root->right != NULL)
    {
        sum += root->key;
    }

    sum += sumEvenNodesWithRightChild(root->left);
    sum += sumEvenNodesWithRightChild(root->right);

    return sum;
}

// Hàm in cây theo kiểu trung tự (in-order)
```

```
void inorder(Node *root)
{
   if (root == NULL)
      return;
   inorder(root->left);
   printf("%d ", root->key);
   inorder(root->right);
}

int main()
{
   Node *root = NULL;
   int n, value;

   printf("Nhap so luong nut trong cay: ");
   scanf("%d", &n);

   printf("Nhap cac gia tri cho cac nut trong cay:\n");
   for (int i = 0; i < n; i++)
   {
      printf("Nhap gia tri nut %d: ", i + 1);
      scanf("%d", &value);
      root = insert(root, value);
   }

   printf("Cay nhi phan tim kiem: ");
```

```
    inorder(root);
    printf("\n");

    // Tìm số nguyên tố lớn nhất ở nhánh con bên trái của nút gốc
    int maxPrime = findMaxPrimeInLeftSubtree(root);
    if (maxPrime != -1)
    {
        printf("So nguyen to lon nhat o nhanh con ben trai cua nut goc: %d\n", maxPrime);
    }
    else
    {
        printf("Khong tim thay so nguyen to trong nhanh con ben trai cua nut goc.\n");
    }

    // Tính tổng giá trị của tất cả các nút có key là số chẵn và có 1 nhánh con bên phải
    int sum = sumEvenNodesWithRightChild(root);
    printf("Tong cac nut co key la so chan va co 1 nhanh con ben phai: %d\n", sum);

    return 0;
}
```

Chèn một nút vào cây:

- Hàm insert chèn giá trị vào cây. Cây được duy trì theo quy tắc:

+ Nếu giá trị nhỏ hơn giá trị của nút hiện tại (root), nó sẽ được chèn vào nhánh con bên trái (root->left).

+ Nếu giá trị lớn hơn giá trị của nút hiện tại, nó sẽ được chèn vào nhánh con bên phải (root->right).

- Base case: Khi cây rỗng (root == NULL), một nút mới sẽ được tạo ra và trả về (với giá trị key).

- Đệ quy: Hàm gọi lại chính nó để tiếp tục tìm đúng vị trí để chèn nút mới.

In cây theo thứ tự:

- Hàm inorder in cây theo thứ tự

- In-order traversal: Duyệt cây theo thứ tự "left -> root -> right", có nghĩa là nó sẽ in các giá trị của cây theo thứ tự tăng dần:

+ Đầu tiên, cây con bên trái sẽ được duyệt.

+ Sau đó, giá trị của nút hiện tại sẽ được in ra.

+ Cuối cùng, cây con bên phải sẽ được duyệt.

Tìm số nguyên tố lớn nhất trong nhánh con bên trái của nút gốc:

Hàm findMaxPrimeInLeftSubtree tìm số nguyên tố lớn nhất trong nhánh con bên trái của nút gốc.

- Điều kiện: Nếu cây rỗng (root == NULL) hoặc cây không có nhánh con bên trái (root->left == NULL), hàm trả về -1, tức là không có số nguyên tố nào.

- Duyệt qua nhánh trái: Hàm duyệt qua cây con bên trái của nút gốc bằng cách bắt đầu từ root->left.

- Kiểm tra số nguyên tố: Với mỗi nút trong cây con bên trái, hàm kiểm tra xem giá trị của nó có phải là số nguyên tố không. Nếu có, và nó lớn hơn giá trị nguyên tố đã tìm được trước đó, hàm cập nhật giá trị maxPrime.

Tính tổng giá trị các nút có key là số chẵn và có một nhánh con bên phải:

Hàm sumEvenNodesWithRightChild tính tổng các giá trị của các nút có key là số chẵn và có một nhánh con bên phải.

- Điều kiện: Hàm kiểm tra xem giá trị của nút hiện tại có phải là số chẵn và có nhánh con bên phải hay không.

- Đệ quy: Hàm gọi lại chính nó để duyệt qua các nút trong cây, bao gồm cả nhánh trái và nhánh phải.