TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI PHÂN HIỆU THÀNH PHỐ HỒ CHÍ MINH



**BÁO CÁO BÀI TẬP LỚN**

Môn: Lập trình nâng cao

Đề tài: Xây dựng máy tính cơ bản trên ngôn ngữ C

**Giảng viên:** Cô Trần Thị Dung

**Thành viên nhóm:**

Nguyễn Ngọc Tuấn Anh

Phan Thái Lâm

Bùi Quốc Lâm

**Lớp:** CQ.60.CNTT

MỤC LỤC

MỤC LỤC……………………………………………………………………………...1

LỜI MỞ ĐẦU………………………………………………………………………….3

CHƯƠNG 1: NỘI DUNG LÝ THUYẾT

**1.1.Danh sách liên kết đơn**……………………………………………………..……...4

**1.2.Con trỏ**………………………………………………...……………………….......5

**1.3.Thuật toán tìm kiếm**……………………………………………………….……...6

1.3.1.Tìm kiếm tuyến tính……………………………………………………………....6

1.3.2.Tìm kiếm nhị phân……………………………………………………………......9

1.3.3.Tìm kiếm nội suy……………………………………………………………......10

**1.4.Thuật toán sắp xếp**…………………………………………………….……........11

1.4.1.Sắp xếp chọn……………………………………………………………….........11

1.4.2.Sắp xếp chèn…………………………………………………………….............13

1.4.3.Sắp xếp nổi bọt……………………………………………………………..........14

1.4.4.Sắp xếp trộn……………………………………………………………..............16

**1.5.Thao tác với tệp**………………………………………………………………….19

1.5.1.Khái niệm………………………………………………………………….……19

1.5.2.Mở tệp…………………………………………………………………………...20

1.5.3.Đóng tệp………………………………………………………………………...21

1.5.4.Ghi dữ liệu cho tệp………………………………………………………………22

1.5.5.Đọc dữ liệu cho tệp………………………………………………………………23

CHƯƠNG 2: TỔNG QUAN VỀ ĐỀ TÀI

**2.1.Lý do chọn đề tài**…………………………………………………………............25

**2.2.Mục tiêu nghiên cứu**………………………………………..…………….............25

**2.3.Gỉai thích code**………………………………………………………...….............25

**3.3.Hướng dẫn sử dụng**………………………………………………………...….....26

CHƯƠNG 3 : TỔNG KẾT

**3.1.Nhược điểm của nhóm**…………………………………..……….………………27

**3.2.Tài liệu tham khảo**…………………………………………………………..…...27

LỜI MỞ ĐẦU

Trong thời kỳ phát triển đất nước cũng với công nghệ. Thì các ứng dụng về công nghệ thông tin ngày càng phổ biến khắp nơi từ sản xuất, buôn bán, xuất nhập khẩu, văn phòng, bất kể việc gì cũng cần đến công nghệ. Theo xu hướng phát triển của toàn cầu, tụi em quyết định lựa chọn môn lập trình để trao dồi kiến thức về chuyên ngành cũng như kiến thức thực tế, từ đó định hướng được quá trình học tập. Qua đó, để tìm hiểu sâu về lập trình ngôn ngữ C ++, chúng em đã đi vào làm bài tập lớn sau. Với ý tưởng tính toán các phép tính cơ bản thường xuất hiện trong đời sống. Nhóm chúng em quyết định lựa chọn đề tài làm máy tính cơ bản để làm bài tập này. Ngoài ra, trong phần báo cáo bên dưới tụi em còn tìm hiểu về các thao tác cơ bản trong C++ : con trỏ, tệp, liên kết đơn,…. Từ đây có thể hiểu rõ hơn về ngôn ngữ lập trình và củng cố kiếm thức cơ bản trong C++.

Sau đây là phần trình bày bài thuyết trình của nhóm em. Cảm ơn cô đã xem.

CHƯƠNG 1: NỘI DUNG LÝ THUYẾT

**1.1.Danh sách liên kết đơn**

* Danh sách liên kết đơn là một tập hợp các Node được phân bố động, được sắp xếp theo cách sao cho mỗi Node chứa “một giá trị”(Data) và “một con trỏ”(Next). Con trỏ sẽ trỏ đến phần tử kế tiếp của danh sách liên kết đó. Nếu con trỏ mà trỏ tới NULL, nghĩa là đó là phần tử cuối cùng của linked list.
* Danh sách liên kết là một trong những cấu trúc dữ liệu đơn giản và phổ biến nhất. Nó có thể được dùng để hiện thực một số [kiểu dữ liệu trừu tượng](https://vi.wikipedia.org/wiki/Ki%E1%BB%83u_d%E1%BB%AF_li%E1%BB%87u_tr%E1%BB%ABu_t%C6%B0%E1%BB%A3ng) phổ biến khác, bao gồm [danh sách](https://vi.wikipedia.org/w/index.php?title=Danh_s%C3%A1ch_(ki%E1%BB%83u_d%E1%BB%AF_li%E1%BB%87u_tr%E1%BB%ABu_t%C6%B0%E1%BB%A3ng)&action=edit&redlink=1) (list), [ngăn xếp](https://vi.wikipedia.org/wiki/Ng%C4%83n_x%E1%BA%BFp) (stack), [hàng đợi](https://vi.wikipedia.org/wiki/H%C3%A0ng_%C4%91%E1%BB%A3i), [mảng kết hợp](https://vi.wikipedia.org/w/index.php?title=M%E1%BA%A3ng_k%E1%BA%BFt_h%E1%BB%A3p&action=edit&redlink=1), và [S-expression](https://vi.wikipedia.org/w/index.php?title=S-expression&action=edit&redlink=1), mặc dù không có gì lạ khi hiện thực các cấu trúc dữ liệu đó mà không dựa trên nền tảng của danh sách liên kết.
* Lợi ích chính của danh sách liên kết so với [mảng](https://vi.wikipedia.org/wiki/M%E1%BA%A3ng_(c%E1%BA%A5u_tr%C3%BAc_d%E1%BB%AF_li%E1%BB%87u)) thông thường là các phần tử danh sách có thể được chèn hay xóa một cách dễ dàng mà không cần phân bổ lại hoặc sắp xếp lại toàn bộ cấu trúc vì các mục dữ liệu không cần được lưu trữ [liên tục](https://vi.wiktionary.org/wiki/contiguous) trong bộ nhớ hay trên đĩa, trong khi tái cấu trúc một mảng tại [thời gian chạy](https://vi.wikipedia.org/w/index.php?title=Run_time_(program_lifecycle_phase)&action=edit&redlink=1) là một hoạt động tốn kém hơn nhiều. Danh sách liên kết cho phép chèn hay xóa nút tại bất kì điểm nào trong danh sách.
* Mặc khác, vì bản thân danh sách liên kết được liên kết đơn giản nên không cho phép [truy cập ngẫu nhiên](https://vi.wikipedia.org/w/index.php?title=Truy_c%E1%BA%ADp_ng%E1%BA%ABu_nhi%C3%AAn&action=edit&redlink=1) tới dữ liệu hoặc bất kì hình thức đánh chỉ mục hiệu quả nào, nhiều toán tử cơ bản như lấy nút cuối cùng của danh sách, tìm một nút có chứa dữ liệu đã cho, hay tìm vị trí của nút để chèn một nút mới sẽ yêu cầu lặp qua hầu hết hoặc tất cả các phần tử của danh sách. Những ưu điểm và nhược điểm của danh sách liên kết được đưa ra dưới đây. Danh sách liên kết là động, vì vậy độ dài của nó có thể tăng hay giảm khi cần thiết. Mỗi nút không cần phải theo nút trước đó trong bộ nhớ.



* Danh sách các kiểu danh sách liên kết:
* Danh sách liên kết đơn(Single linked list): Chỉ có sự kết nối từ phần tử phía trước tới phần tử phía sau.
* Danh sách liên kết đôi(Double linked list): Có sự kết nối 2 chiều giữa phần tử phía trước với phần tử phía sau
* Danh sách liên kết vòng(Circular Linked List): Có thêm sự kết nối giữa 2 phần tử đầu tiên và phần tử cuối cùng để tạo thành vòng khép kín.
* Một số hoạt động cơ bản có thể được thực hiện bởi một danh sách liên kết:
* Hoạt động chèn: thêm một phần tử vào đầu danh sách liên kết.
* Hoạt động xóa (phần tử đầu): xóa một phần tử tại đầu danh sách liên kết.
* Hiển thị: hiển thị toàn bộ danh sách.
* Hoạt động tìm kiếm: tìm kiếm phần tử bởi sử dụng khóa (key) đã cung cấp.
* Hoạt động xóa (bởi sử dụng khóa): xóa một phần tử bởi sử dụng khóa (key) đã cung cấp.

**1.2.Con trỏ**

* **Con trỏ là gì?**Con trỏ trong C cũng chỉ là biến, cũng có thể khai báo, khởi tạo và lưu trữ giá trị và có địa chỉ của riêng nó. Nhưng biến con trỏ không lưu giá trị bình thường, nó là biến trỏ tới 1 địa chỉ khác, tức mang giá trị là 1 **địa chỉ**. Chúng ta cùng thống nhất 1 số khái niệm khi làm việc với con trỏ :
* Giá trị của con trỏ: địa chỉ mà con trỏ trỏ đến.
* Địa chỉ của con trỏ: địa chỉ của bản thân biến con trỏ đó.
* Giá trị của biến nơi con trỏ đang trỏ tới.
* Địa chỉ của biến nơi con trỏ đang trỏ tới = giá trị của con trỏ.
* Có một vài phép toán quan trọng, sẽ giúp chúng ta làm việc với con trỏ một cách thường xuyên: **a**) chúng ta định nghĩa biến con trỏ, **b**) gán địa chỉ của biến đến một con trở và **c**) cuối cùng truy cập các giá trị biến địa chỉ trong biến con trỏ. Điều này được thực hiện bởi toán tử \* trả về giá trị các các biến chứa trong địa chỉ được xác định bởi toán tử này.
* Cách khai báo con trỏ : Con trỏ trong C cũng có thể khai báo giống như biến bình thường, tên biến là một định danh hợp lệ. Cú pháp như sau:

VD : <kiểu dữ liệu> \* <tên biến>

Trong đó: Kiểu dữ liệu có thể là: void, int, float, double,… Dấu \* trước tên biến là ký hiệu báo cho trình biên dịch biết ta đang khai báo con trỏ.

VD:

int \*p\_i; // Khai báo con trỏ để trỏ tới biến kiểu nguyên

int \*p\_f; // Khai báo con trỏ để trỏ tới biến kiểu thực

int \*p\_char; // Khai báo con trỏ để trỏ tới biến kiểu ký tự

* Sau khi khai báo con trỏ, bạn cần khởi tạo giá trị cho nó. Nếu con trỏ được sử dụng mà không được khởi tạo, giá trị của nó sẽ là giá trị rác, điều này sẽ làm chương trình của bạn chạy không đúng, thậm chí là nguy hiểm nếu giá trị rác đó chẳng may lại chính là địa chỉ của 1 biến nào đó bạn đang dùng.

VD :

int\*p, value;

Value = 5;

P = &value;

VD :

int \*contro;

double \*phithuebao;

float \*hocphi;

char \*ho, \*ten;

**1.3.Thuật toán tìm kiếm**

**1.3.1.Tìm kiếm tuyến tính**

* Giải thuật chính của tìm kiếm tuyến tính chính là: so sánh phần tử cần tìm với tất cả các phần tử có trong mảng hoặc danh sách cần tìm. Chạy từ phần tử đầu đến cuối và so sánh từng đôi một, nếu bằng thì thông báo có, ngược lại nếu đã đi hết dãy mà vẫn chưa có phần tử nào thõa mãn thì cho kết quả là không tìm thấy.
* Các bước giải thuật :

Bước 1: Khởi tạo biến i và gán biến i bằng 0;

Bước 2: so sánh a[i] với giá trị cầm tìm.

+ Nếu tìm được giá trị a[i] bằng giá trị cần tìm thì dừng lại và dừng. Ngược lại

+  Nếu  a[i] khác giá trị cần tìm thì sang bước 3

Bước 3: Tăng i lên một đơn vị, nếu i bằng số phần tử trừ 1 của mảng thì dừng lại và cho kết quả là không tìm thấy. Ngược lại quay lại bước 2.

VD: Code tổng quát

int TimKiemTuyenTinh(int a[], int n, int x)

{

for (int i = 0; i < n;i++)

if (a[i] == x)

return 1;

return 0;

}<strong> </strong>

**VD**: Cho dãy A gồm các phần tử: 11 4 3 9 8 0 2 45. Dùng giải thuật tìm kiếm tuyến tính để tìm xem có phần tử 8 nằm ở trong mảng hay không.

void NhapMang(int a[], int &n)

{

printf("Cho biet so phan tu cua mang: ");

scanf("%d", &n);

for (int i = 0; i<n; i++)

{

printf("Gia tri phan tu a[%d]=", i);

scanf("%d", &a[i]);

}

}

void XuatMang(int a[], int n)

{

for (int i = 0; i<n; i++)

printf("%4d", a[i]);

}

int TimKiemTuyenTinh(int a[], int n, int x)

{

for (int i = 0; i < n;i++)

if (a[i] == x)

return 1;

return 0;

}

#define Max 100

void main()

{

int A[Max];

int N;

int X;

NhapMang(A, N);

printf("Nhap phan tu can tim");

scanf("%d", &X);

int b=TimKiemTuyenTinh(A, N, X);

if (b == 1)

printf("%d co trong mang",X);

else

{

printf("%d khong co trong mang", X);

}

getch();

}

VD: một ví dụ khác về Struct

|  |
| --- |
| #include<stdio.h>  #include<conio.h>  #define Max 100  struct SinhVien  {  char Ma[10];  char Ho[10];  char Ten[10];  int NamSinh;  };  void NhapMangSV(SinhVien a[], int &n);  void XuatMangSV(SinhVien a[], int n);  void NhapMangSV(SinhVien a[], int &n)  {  do{  printf("Cho biet so Sinh vien: ");  scanf("%d", &n);  } while (n <= 0);  for (int i = 1; i <= n; i++)  {  printf("Thong tin Sinh vien thu %d la: \n", i);  printf("Ma so: \n");  fflush(stdin);  gets(a[i].Ma);  printf("Ho :\n");  fflush(stdin);  gets(a[i].Ho);  printf("Ten :\n");  fflush(stdin);  gets(a[i].Ten);  printf("Nam sinh :\n");  scanf("%d", &a[i].NamSinh);  }  }    int TimSV(SinhVien a[], int n, char ten[10])  {  for (int i = 0; i < n; i++)  if (a[i].Ten == ten)  return 1;  return 0;  }  void main()  {  SinhVien A[100];  int N;  int X;  char TEN[10];  NhapMangSV(A, N);  printf("Nhap ten SV can tim");  fflush(stdin);  gets(TEN);  int y=TimSV(A, N, TEN);  if (y == 1)  printf("Co sinh vien ten %s trong danh sach", TEN);  else  {  printf("Khong co sinh vien %s trong danh sach", TEN);  }  getch();  }  **1.3.2.Tìm kiếm nhị phân**   * Thuật toán Tìm kiếm nhị phân (Binary Search) là một thuật toán cao cấp tìm kiếm tuyến tính hơn với thời gian chạy là O(log n). Đối với các danh sách lớn, thuật toán này tốt hơn hẳn tìm[kiếm tuyến tính](https://dnmtechs.com/vi/thuat-toan-tim-kiem-tuyen-tinh-tim-kiem-tuan-tu/), nhưng nó đòi hỏi danh sách phải được sắp xếp từ trước và đòi hỏi khả năng truy nhập ngẫu nhiên (random access). * Thuật toán tìm kiếm nhị phân dùng để tìm kiếm phần tử trong một danh sách đã được [sắp xếp](https://dnmtechs.com/vi/thuat-toan-sap-xep/), ví dụ như trong một danh bạ điện thoại sắp xếp theo tên, có thể tìm kiếm số điện thoại của một người theo tên người đó. * Tìm kiếm nhị phân so sánh giá trị đích với phần tử ở giữa của mảng. Nếu chúng không bằng nhau, một nửa trong đó không chứa mục tiêu tìm kiếm bị loại bỏ và việc tìm kiếm tiếp tục ở nửa còn lại, một lần nữa lấy phần tử ở giữa được chọn để so sánh với giá trị đích và lặp lại điều này cho đến khi tìm thấy giá trị đích. Nếu tìm kiếm kết thúc với nửa còn lại trống, mục tiêu không nằm trong mảng. Mặc dù ý tưởng rất đơn giản, nhưng việc thực hiện tìm kiếm nhị phân chính xác đòi hỏi phải chú ý đến một số điểm tinh tế về điều kiện thoát và tính toán điểm giữa của nó.   C:\Users\Admin\Desktop\thuat-toan-tim-kiem-nhi-phan-binary-search.png |

**Hình 1.3.2**

Minh họa Thuật toán tìm kiếm Nhị phân (Binary Search)Trong ví dụ minh họa ở hình trên, Tìm vị trí phần tử có giá trị là **23** trong một mảng**A[2,5,8,12,16,23,38,56,72,91]** đã được sắp xếp. Về cơ bản bạn loại bỏ được một nửa các yếu tố chỉ sau một lần so sánh.

1. So sánh x với phần tử ở giữa.
2. Nếu x khớp với phần tử ở giữa, chúng ta trả về chỉ số giữa.
3. Khác Nếu x lớn hơn phần tử mid, thì x chỉ có thể nằm trong nửa phân đoạn bên phải sau phần tử mid. Vì vậy, chúng ta chỉ tìm kiếm ở nữa phải của mảng.
4. Khác (x nhỏ hơn) tiếp tục cho nửa bên trái.
5. Lặp lại đến khi tìm ra x hoặc trả về null khi x không nằm trong mảng.

VD:

|  |  |
| --- | --- |
|  | #include <iostream>  using namespace std;  int binarySearch(int arr[], int l, int r, int x)  {      if (r >= l) {          int mid = l + (r - l) / 2;          if (arr[mid] == x)              return mid;          if (arr[mid] > x)              return binarySearch(arr, l, mid - 1, x);            return binarySearch(arr, mid + 1, r, x);      }        return -1;  }    int main(void)  {      int arr[] = { 2, 3, 4, 10, 40 };      int x = 10;      int n = sizeof(arr) / sizeof(arr[0]);      int result = binarySearch(arr, 0, n - 1, x);      (result == -1) ? cout << "Element is not present in array"                     : cout << "Element is present at index " << result;      return 0;  } |

**1.3.3.Tìm kiếm nội suy**

* Tìm kiếm nội suy (interpolation search) là một thuật toán cải tiến từ thuật toán tìm kiếm nhị phân. Thay vì xác định điểm chính giữa của danh sách, thuật toán tìm kiếm nội suy xác định điểm gần với vị trí của phần tử cần tìm, do đó tối ưu được thời gian hơn so với thuật toán tìm kiếm nhị phân. Độ phức tạp thời gian cũng vì thế mà tốt hơn là O(log(log(n))).
* Tuy nhiên, thuật toán tìm kiếm nhị phân luôn ổn định với độ phức tạp thời gian là O(log(n)), thuật toán tìm kiếm nội suy lại không như vậy. Trong những trường hợp xấu nhất như dãy tăng/giảm phân bố không đều, thuật toán tìm kiếm này đạt độ phức tạp là O(n), không khác gì dùng thuật toán tìm kiếm tuyến tính cả. Do đó, bạn nên sử dụng thuật toán tìm kiếm nhị phân để đảm bảo được độ phức tạp O(log(n)).

VD: Có mảng A, n phần tử bắt đầu từ 0 và tăng dần. Tìm x trong mảng này dùng thuật toán tìm kiếm nội suy như sau:

1. Gán left = 0, right = n – 1.
2. Gán mid = left + (right – left) \* (x – a[left]) / (a[right] – a[left]):

Nếu như A[mid] == x: Dừng lại và trả về giá trị của mid.

Nếu như A[mid] > x: right = mid – 1

Nếu như A[mid] < x: left = mid + 1

1. Nếu left <= right và x >= A[left] và x <= A[right] (x còn nằm trong đoạn [A[left]; A[right]]):

Đúng thì quay lại bước 2.

Sai thì dừng và trả về kết quả -1 (không tìm thấy x)

Và tương tự với mảng giảm dần, bạn chỉ cần sửa lại:

A[mid] < x: right = mid – 1

A[mid] > x: left = mid + 1

int InterpolationSearch(int A[], int n, int x){

int left = 0;

int right = n - 1;

int mid;

while (left <= right && x >= A[left] && x <= A[right])

{

mid = left + (right - left) \* (x - A[left]) / (A[right] - A[left]);

if (A[mid] == x)

return mid;

if (A[mid] > x)

right = mid - 1;

else if (A[mid] < x)

left = mid + 1;

}

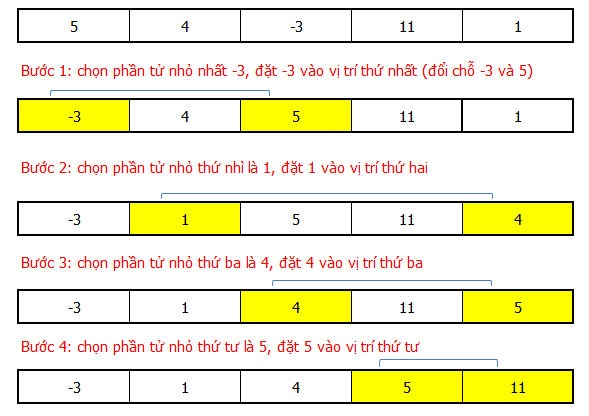
return -1; // Không tìm thấy x

}

**1.4.Thuật toán sắp xếp**

**1.4.1.Sắp xếp chọn**

* Sắp xếp chọn (selection sort) là phương pháp sắp xếp bằng cách chọn phần tử bé nhất xếp vào vị trí thứ nhất, tương tự với các phần tử nhỏ thứ hai, thứ ba,…
* Ý tưởng chính của thuật toán là xuất phát từ cuối (hoặc đầu) dãy, đổi chổ các cặp phần tử kế cận để đưa phần tử nhỏ (lớn) hơn trong cặp phần tử đó về vị trí đúng đầu (cuối) dãy hiện hành, sau đó sẽ không xét đến nó ở vị trí tiếp theo, do vậy ở lần xử lý thứ i sẽ có vị trí đầu dãy là i. Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét thì dừng giải thuật.

VD: Sắp xếp dãy 5, 4, -3, 11, 1 theo chiều tăng dần

**Hình 1.4.1**

VD:

void Sapxep(int a[], int n)

{

for (int i = 0; i < n - 1; i++)

{

int min = i;

for (int j = i + 1; j < n; j++)

if (a[min]>a[j])

min = j;

swap(a[i], a[min]);

}

}

void swap(int &a, int &b)

{

int x = a;

a = b;

b = x;

}

void main()

{

int A[100];

NhapMang(A, N);

Sapxep(A, N);

printf("Mang sau khi sap xep la: ");

XuatMang(A, N);

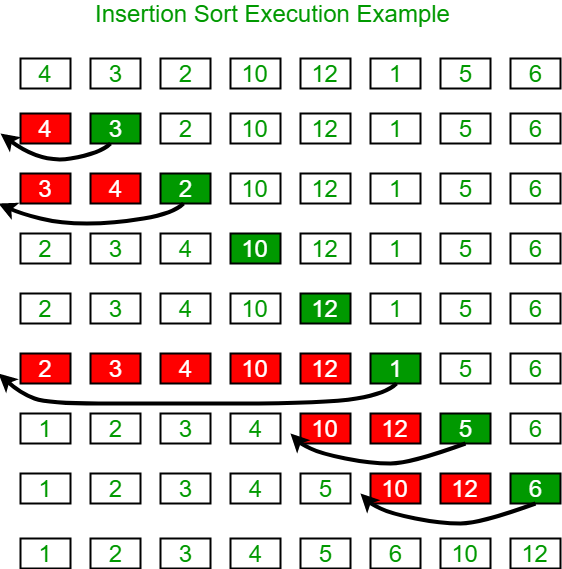
getch();

}

**1.4.2.Sắp xếp chèn**

* Sắp xếp chèn (insertion sort) là thuật toán sắp xếp cho một dãy đã có thứ tự. Chèn thêm một phần tử vào vị trí thích hợp của dãy số đã sắp xếp sao cho dãy số vẫn là dãy sắp xếp có thứ tự.
* Thuật toán sắp xếp chèn thực hiện sắp xếp dãy số theo cách duyệt từng phần tử và chèn từng phần tử đó vào đúng vị trí trong mảng con(dãy số từ đầu đến phần tử phía trước nó) đã sắp xếp sao cho dãy số trong mảng sắp đã xếp đó vẫn đảm bảo tính chất của một dãy số tăng dần.

VD:



**Hình 1.4.2**

VD:

#include <stdio.h>

#include <math.h>

void insertionSort(int arr[], int n)

{

   int i, key, j;

   for (i = 1; i < n; i++)

   {

       key = arr[i];

       j = i-1;

       /\* Di chuyển các phần tử có giá trị lớn hơn giá trị

       key về sau một vị trí so với vị trí ban đầu

       của nó \*/

       while (j >= 0 && arr[j] > key)

       {

           arr[j+1] = arr[j];

           j = j-1;

       }

       arr[j+1] = key;

   }

}

void printArray(int arr[], int n)

{

   int i;

   for (i=0; i < n; i++)

       printf("%d ", arr[i]);

   printf("\n");

}

int main()

{

    int arr[] = {12, 11, 13, 5, 6};

    int n = sizeof(arr)/sizeof(arr[0]);

    insertionSort(arr, n);

    printf("Sorted array: \n");

    printArray(arr, n);

    return 0;

}

**1.4.3.Sắp xếp nổi bọt**

* Sắp xếp nổi bọt (*bubble sort*) là phương pháp sắp xếp đơn giản, dễ hiểu thường được dạy trong khoa học máy tính. Nó so sánh hai phần tử cuối, nếu phần tử đứng trước lớn hơn phần tử đứng sau thì đổi chỗ chúng cho nhau. Tiếp tục làm như vậy với cặp phần tử tiếp theo cho đến cuối đầu dãy số. Sau đó nó quay lại với hai phần tử cuối cho đến khi không còn cần phải đổi chỗ nữa.
* Thuật toán sắp xếp nổi bọt thực hiện sắp xếp dãy số bằng cách lặp lại công việc đổi chỗ 2 số liên tiếp nhau nếu chúng đứng sai thứ tự(số sau bé hơn số trước với trường hợp sắp xếp tăng dần) cho đến khi dãy số được sắp xếp.

VD:

Giả sử chúng ta cần sắp xếp dãy số [5 1 4 2 8] này tăng dần.  
**Lần lặp đầu tiên:**  
( **5** **1** 4 2 8 ) –> ( **1** **5** 4 2 8 ), Ở đây, thuật toán sẽ so sánh hai phần tử đầu tiên, và đổi chỗ cho nhau do 5 > 1.  
( 1 **5** **4** 2 8 ) –>  ( 1 **4** **5** 2 8 ), Đổi chỗ do 5 > 4  
( 1 4 **5** **2** 8 ) –>  ( 1 4 **2** **5** 8 ), Đổi chỗ do 5 > 2  
( 1 4 2 **5** **8** ) –> ( 1 4 2 **5** **8** ), Ở đây, hai phần tử đang xét đã đúng thứ tự (8 > 5), vậy ta không cần đổi chỗ.

**Lần lặp thứ 2:**  
( **1** **4** 2 5 8 ) –> ( **1** **4** 2 5 8 )  
( 1 **4** **2** 5 8 ) –> ( 1 **2** **4** 5 8 ), Đổi chỗ do 4 > 2  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –>  ( 1 2 4 **5** **8** )  
Bây giờ, dãy số đã được sắp xếp, Nhưng thuật toán của chúng ta không nhận ra điều đó ngay được. Thuật toán sẽ cần thêm một lần lặp nữa để kết luận dãy đã sắp xếp khi và khi khi nó đi từ đầu tới cuối mà không có bất kỳ lần đổi chỗ nào được thực hiện.

**Lần lặp thứ 3:**  
( **1** **2** 4 5 8 ) –> ( **1** **2** 4 5 8 )  
( 1 **2** **4** 5 8 ) –> ( 1 **2** **4** 5 8 )  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –> ( 1 2 4 **5** **8** )

VD:

#include <stdio.h>

void swap(int &x, int &y)

{

    int temp = x;

    x = y;

    y = temp;

}

void bubbleSort(int arr[], int n)

{

    int i, j;

    bool haveSwap = false;

    for (i = 0; i < n-1; i++){

        haveSwap = false;

        for (j = 0; j < n-i-1; j++){

            if (arr[j] > arr[j+1]){

                swap(arr[j], arr[j+1]);

                haveSwap = true;

            }

        }

        if(haveSwap == false){

            break;

        }

    }

}

void printArray(int arr[], int size)

{

    int i;

    for (i=0; i < size; i++)

        printf("%d ", arr[i]);

    printf("n");

}

int main()

{

    int arr[] = {64, 34, 25, 12, 22, 11, 90};

    int n = sizeof(arr)/sizeof(arr[0]);

    bubbleSort(arr, n);

    printf("Sorted array: n");

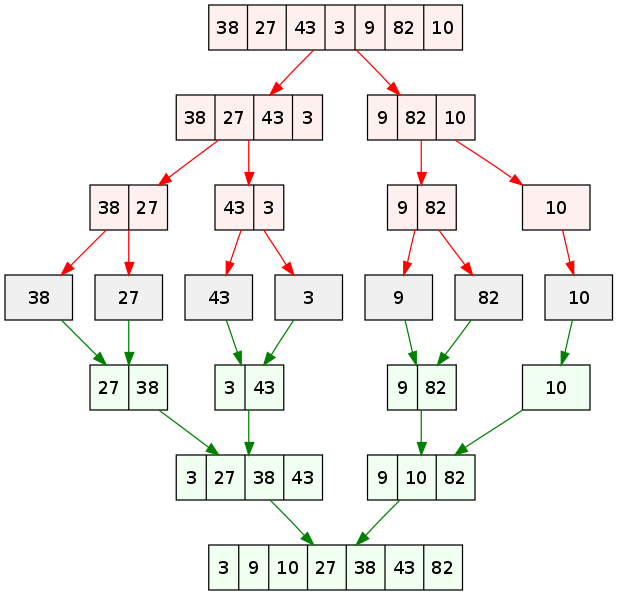
    printArray(arr, n);

    return 0;

}

**1.4.4.Sắp xếp trộn**

* Sắp xếp trộn (merge sort) cùng với sắp xếp nhanh là hai thuật toán sắp xếp dựa vào tư tưởng “chia để trị” (divide and conquer). Thủ tục cơ bản là việc trộn hai danh sách đã được sắp xếp vào một danh sách mới theo thứ tự. Nó có thể bắt đầu trộn bằng cách so sánh hai phần tử một (chẳng hạn phần tử thứ nhất với phần tử thứ hai, sau đó thứ ba với thứ tư…) và sau khi kết thúc bước 1 nó chuyển sang bước 2. Ở bước 2 nó trộn các danh sách hai phần tử thành các danh sách bốn phần tử. Cứ như vậy cho đến khi hai danh sách cuối cùng được trộn thành một.
* Merge sort là một thuật toán chia để trị. Thuật toán này chia mảng cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa mảng đã chia. Sau cùng gộp các nửa đó thành mảng đã sắp xếp. Hàm merge() được sử dụng để gộp hai nửa mảng. Hàm merge(arr, l, m, r) là tiến trình quan trọng nhất sẽ gộp hai nửa mảng thành 1 mảng sắp xếp, các nửa mảng là arr[l…m] và arr[m+1…r] sau khi gộp sẽ thành một mảng duy nhất đã sắp xếp.

VD: 

**Hình 1.4.4**

Nhìn kỹ hơn vào sơ đồ này, chúng ta có thể thấy mảng ban đầu được lặp lại hành động chia cho tới khi kích thước các mảng sau chia là 1. Khi kích thước các mảng con là 1, tiến trình gộp sẽ bắt đầu thực hiện gộp lại các mảng này cho tới khi hoàn thành và chỉ còn một mảng đã sắp xếp.

VD:

#include<stdlib.h>

#include<stdio.h>

void merge(int arr[], int l, int m, int r)

{

    int i, j, k;

    int n1 = m - l + 1;

    int n2 =  r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)

        L[i] = arr[l + i];

    for (j = 0; j < n2; j++)

        R[j] = arr[m + 1+ j];

    i = 0;

    j = 0;

    k = l;

    while (i < n1 && j < n2)

    {

        if (L[i] <= R[j])

        {

            arr[k] = L[i];

            i++;

        }

        else

        {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

    while (i < n1)

    {

        arr[k] = L[i];

        i++;

        k++;

    }

    while (j < n2)

    {

        arr[k] = R[j];

        j++;

        k++;

    }

}

void mergeSort(int arr[], int l, int r)

{

    if (l < r)

    {

   int m = l+(r-l)/2;

        mergeSort(arr, l, m);

        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);

    }

}

void printArray(int A[], int size)

{

    int i;

    for (i=0; i < size; i++)

        printf("%d ", A[i]);

    printf("\n");

}

int main()

{

    int arr[] = {12, 11, 13, 5, 6, 7};

    int arr\_size = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");

    printArray(arr, arr\_size);

    mergeSort(arr, 0, arr\_size - 1);

    printf("\nSorted array is \n");

    printArray(arr, arr\_size);

    return 0;

}

**1.5.Thao tác với tệp**

**1.5.1.Khái niệm**

* Nếu máy tính có các ổ cứng, hoặc có USB hoặc bất kỳ thiết bị lưu trữ nào thì chắc chắn chúng ta đã từng làm việc với File. Khi chơi một game offline, thông tin nhân vật, điểm số, ... sẽ được lưu trữ trong File để khi chương trình game bị tắt đi thì không phải chơi lại từ đầu. Khi các cài đặt cấu hình cho một phần mềm và tắt đi, cấu hình đó được lưu vào File để lần làm việc tiếp theo sẽ sử dụng. Hay khi biên dịch một chương trình C++ trên Visual Studio 2015, C++ Compiler của Visual studio sẽ đọc mã nguồn đã viết trong các file \*.cpp để kiểm tra lỗi và dịch chúng sang file \*.obj. Ngay cả hệ điều hành Windows mà chúng ta đang sử dụng cũng là tập hợp của rất nhiều file được lưu trữ bên trong phân vùng ổ đĩa dùng cho Hệ điều hành...
* Đó là một vài ví dụ cho thấy sự tồn tại của File trong máy tính. Vậy thì chúng ta đã thao tác với những File đó như thế nào?
* Làm việc với File chúng ta chỉ có các thao tác cơ bản như: tạo file mới, đọc dữ liệu trong file, ghi dữ liệu vào file, xóa file... Và chúng ta làm điều đó hằng ngày, khi chúng ta chơi game, khi xem phim trên máy tính, ... và ngay cả khi chúng ta lập trình, mã nguồn của chúng ta được lưu xuống File mã nguồn khi nhấn tổ hợp phím Ctrl + S.
* Một file trên máy tính là một tài nguyên dùng để lưu trữ thông tin lâu dài, sử dụng cho các chương trình máy tính.
* Cũng giống như việc lưu trữ dữ liệu tạm thời trên RAM, file cũng lưu trữ dữ liệu dưới dạng nhị phân (0 hoặc 1), tuy nhiên tùy vào định dạng của file và cách chuyển đổi của mỗi phần mềm đọc file mà chúng ta có những kiểu thông tin khác nhau. Ví dụ file .png thì được chuyển về dạng hình ảnh, phần mềm Microsoft Word chuyển dãy bit nhị phân về dạng text...
* **Trong ngôn ngữ lập trình C/C++**: File là kiểu đối tượng, nó xác định một stream và chứa các thông tin cần thiết để điều khiển, bao gồm một con trỏ trỏ đến buffer của nó, các chỉ mục và trạng thái của nó. Có thể hiểu File (trong ngôn ngữ lập trình C/C++) là một kiểu đối tượng mà thông qua nó chúng ta có thể thao tác với dữ liệu được lưu trữ bên trong File (chứ không phải là một File trên máy tính).
* Để làm việc với file, cần biết vị trí của file (thông qua đường dẫn) để con trỏ kiểu FILE có thể tạo được luồng dữ liệu giữa người dùng và file trên thiết bị lưu trữ.

VD:

#include <iostream>

#include <cstdio>

int main()

{

const char \*filePath = "C:/Users/ADMIN/Desktop/my\_document.txt";

FILE \*file;

return 0;

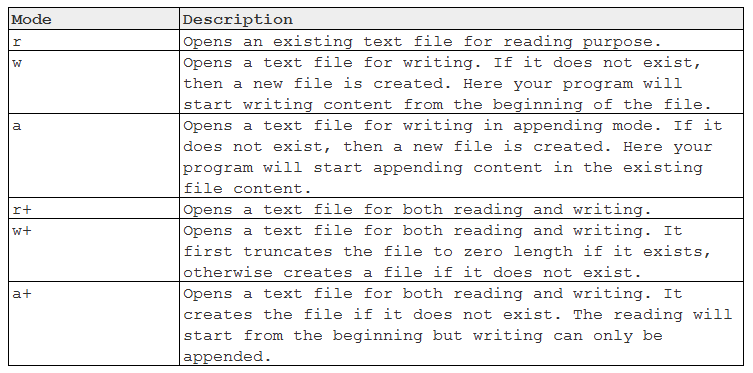
}

**1.5.2.Mở tệp**

* Để mở một file, các bạn có thể sử dụng hàm fopen được định nghĩa trong thư viện cstdio:

FILE\* fopen(const char \*file, const char \*mode);

* Hàm fopen cho phép tạo một kết nối đến file với đường dẫn được lưu trữ bởi tham số thứ nhất. Nếu file không tồn tại, file mới sẽ được tạo ra với tên file như trong đường dẫn. Tham số thứ hai xác định kiểu truy cập vào file. Bảng dưới đây liệt kê các mode dùng để mở một file trong C:



* Nếu mở file thành công, một địa chỉ của một đối tượng kiểu FILE sẽ được trả về. Nếu mở file thất bại thì trả về NULL.

**const** **char** \*filePath = "C:/Users/ADMIN/Desktop/my\_document.txt";

FILE \*file;

file = fopen(filePath, "r");

**if** (!file)

std::cout << "Can not open this file" << std::endl;

**else**

std::cout << "File is opened" << std::endl;

**1.5.3.Đóng tệp**

* Sau khi thao tác với file xong, các bạn cần đóng file lại để tránh những lỗi phát sinh ngoài ý muốn. Để đóng file, chúng ta sử dụng hàm fclose:

int fclose(**FILE** \***file**);

* Trong đó, file là con trỏ được dùng để lưu trữ địa chỉ của đối tượng FILE đang mở. Nếu đóng file thành công thì trả về giá trị 0, ngược lại trả về **EOF** (End of file).

const char \*filePath = "C:/Users/ADMIN/Desktop/my\_document.txt";

FILE \*file;

file = fopen(filePath, "r");

if (!file)

std::cout << "Can not open this file" << std::endl;

else

std::cout << "File is opened" << std::endl;

fclose(file);

* Hàm fclose sẽ giải phóng tất cả dữ liệu chưa được xử lý trên file nếu chúng vẫn còn lưu trong buffer, đóng file lại, và giải phóng tất cả vùng nhớ mà đối tượng FILE sử dụng.

**1.5.4.Ghi dữ liệu cho tệp**

* Khi đã thực hiện được thao tác mở và đóng file, nhưng lúc này, file mới tạo ra vẫn chưa có dữ liệu nên mình sẽ thực hiện thao tác ghi dữ liệu vào file trước. Để mở file cho chế độ ghi file, chúng ta có các mode "w", "r+", "w+", "a", "a+". Mình chỉ muốn ghi dữ liệu nên mình sẽ chọn mode "w". Nhưng trước hết, chúng ta nên tách thao tác ghi file ra một hàm riêng có dạng:

**void** writeToFile(**FILE** \***file**);

* Hàm này sẽ được gọi sau khi mở file và trước khi đóng file.

**const** **char** \*filePath = "C:/Users/ADMIN/Desktop/my\_document.txt";

FILE \*file;

file = fopen(filePath, "w");

**if** (!file)

std::cout << "Can not open this file" << std::endl;

**else**

std::cout << "File is opened" << std::endl;

writeToFile(file);

fclose(file);

* Để ghi dữ liệu vào file, chúng ta có các hàm đã được định nghĩa sẵn trong thư viện cstdio như sau:
* **fputc:** sẽ ghi ký tự có mã ASCII là c vào file được trỏ đến bởi con trỏ f. Giá trị trả về là EOF nếu ghi dữ liệu thất bại, trả về mã ASCII của kí tự được ghi vào nếu thực hiện thành công.

**VD:**

**void** **writeToFile**(FILE \*file)

{

**int** c = fputc('A', file);

std::cout << c << std::endl;

}

* **fputs:** ghi một C-Style string vào file được trỏ đến bởi con trỏ f cho đến khi gặp kí tự '\0'.

VD:

**void** writeToFile(**FILE** \***file**)

{

**int** c = fputs("hello", **file**);

}

* **fprintf:** tương tự hàm printf trong ngôn ngữ C. Tuy nhiên, hàm printf được mặc định liên kết với đối tượng FILE có tên là stdout nên sử dụng hàm printf sẽ ghi nội dung ra màn hình.
* Muốn ghi dữ liệu vào file nên chúng ta sẽ truyền vào tham số thứ nhất là một con trỏ kiểu FILE khác với các đối tượng stdout, stdin hay stderr.

**int** **fprintf**(FILE \*f, **const** **char** \*format, ...);

VD:

**void** writeToFile(**FILE** \***file**)

{

**for** (**int** i = 1; i <= 5; i++)

fprintf(**file**, "This is an example line %d\n", i);

}

**1.5.5.Đọc dữ liệu cho tệp**

* Đầu tiên mình sẽ tạo một hàm khác có tên là readFromFile như sau:

**void** readFromFile(**FILE** \***file**)

{

*//read data*

}

* Để làm các ví dụ trong phần này, mình sẽ gọi hàm này sau khi đã gọi hàm writeToFile.

**const** **char** \*filePath = "C:/Users/ADMIN/Desktop/my\_document.txt";

FILE \*file;

file = fopen(filePath, "w+");

**if** (!file)

std::cout << "Can not open this file" << std::endl;

**else**

std::cout << "File is opened" << std::endl;

writeToFile(file);

readFromFile(file);

fclose(file);

* **fgetc:** đọc ra một kí tự trong file, internal file position indicator sẽ chuyển đến kí tự tiếp theo. Giá trị trả về là mã ASCII của kí tự đã đọc được.

**VD:**

**void** **readFromFile**(FILE \*file)

{

std::cout << (**char**)fgetc(file) << std::endl;

}

* **fgets:** đọc từ file ra (n - 1) kí tự, việc đọc dữ liệu sẽ bị dừng nếu đọc được kí tự new line '\n' hoặc EOF. Chuỗi kí tự đọc được sẽ lưu vào vùng nhớ được quản lý bởi con trỏ buf, nếu đọc dữ liệu thành công thì trả về địa chỉ của buf, ngược lại trả về NULL.

**VD:**

**void** **readFromFile**(FILE \*file)

{

**char** str[255];

std::cout << fgets(str, 255, file) << std::endl;

std::cout << str << std::endl;

}

* **fscanf:** cũng hoạt động tương tự hàm scanf trong ngôn ngữ C, tuy nhiên, hàm scanf được thiết lập liên kết mặc định đến file stdin. Như vậy, fscanf(stdin, format, ...) sẽ tương đương với scanf(format, ...).
* Tương tự như ghi dữ liệu vào file với định dạng cho trước, chúng ta cũng có thể đọc dữ liệu từ file với một định dạng nào đó.

**int** **fscanf**(FILE \*f, **const** **char** \*format, ...);

VD:

**void** **readFromFile**(FILE \*file)

{

**char** str[255];

fscanf(file, "%[^\n]", str);

std::cout << str << std::endl;

}

CHƯƠNG 2: TỔNG QUAN VỀ ĐỀ TÀI

**2.1.Lý do chọn đề tài**

Trong xã hội ngày càng phát triển hiện nay, khoa học công nghệ là thứ không thể thiếu đối với mỗi quốc gia, doanh nghiệp, trường học hay mỗi cá nhân, đặc biệt là công nghệ thông tin. Với sự phát triển nhanh một cách không ngừng nghĩ như vậy của công nghệ thông tin đã giúp giải quyết các công việc học tập, nguyên cứu, quản lý thông tin,… một cách dễ dàng và tiện lợi. Thấy được điều đó, chúng em cần tìm hiểu sâu hơn về ngôn ngữ lập trình để phát triến các công nghệ sau này. Đó là lý do chúng em chọn đề tài làm máy tính cơ bản để tìm hiểu các bước đầu để đi đến chuyên sâu trong ngôn ngữ lập trình C.

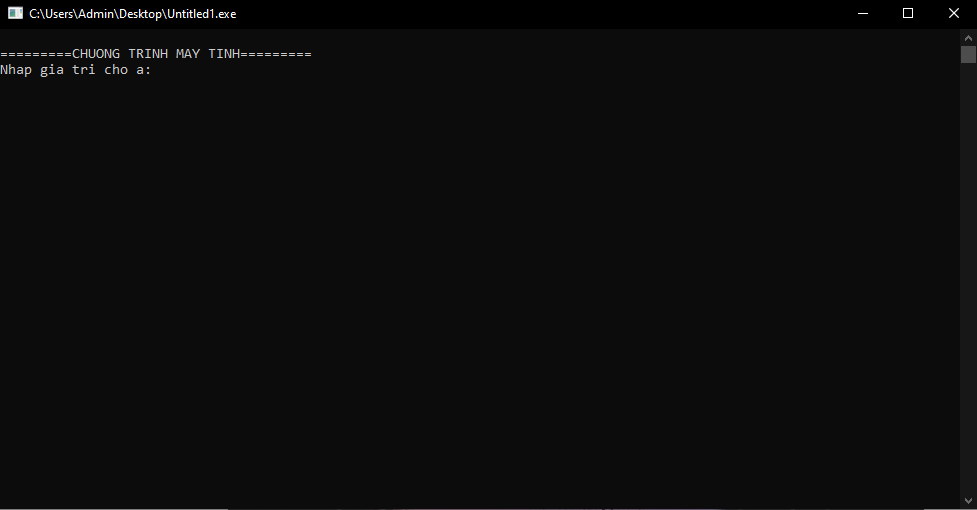
**2.2.Mục tiêu nghiên cứu**

* Xây dựng ứng dụng máy tính cơ bản trên nền tảng ngôn ngữ lập trình C.
* Tìm hiểu về lý thuyết con trỏ, cấu trúc dữ liệu, hàm,…

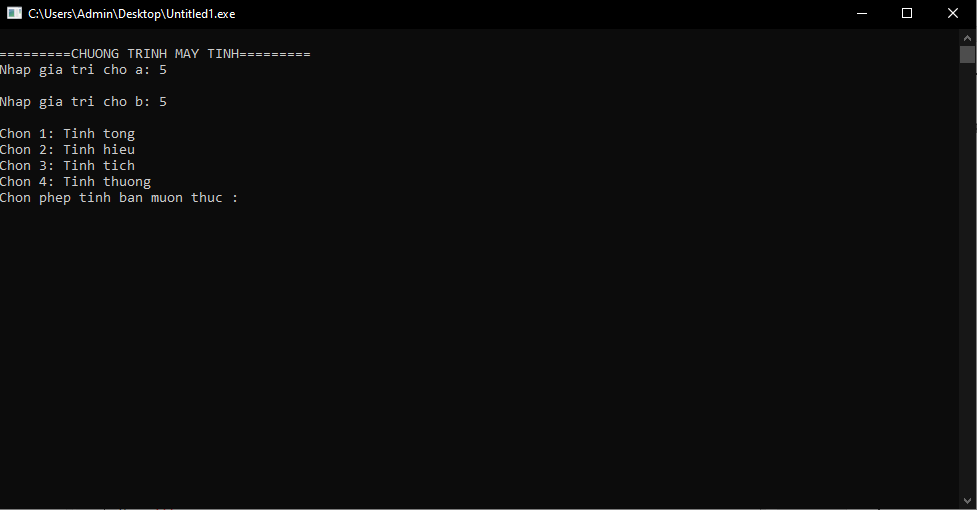
**2.3.Gỉai thích code**

* Em khai báo 3 thư viện
* <stdio.h> : Header file trong Thư viện C chuẩn định nghĩa 3 kiểu biến, một số macro và các hàm đa dạng để thực hiện input và output.
* <conio.h> : Thư viện này hỗ trợ các hàm giúp bạn thực hiện các thao tác input hoặc output từ màn hình console (để sử dụng hàm : getch() ).
* <math.h> : Header filetrong Thư viện C định nghĩa các hàm toán học đa dạng và một macro. Tất cả các hàm có sẵn trong thư viện này nhận double như là một tham số và trả về kết quả ở kiểu double.
* Tiếp theo khai báo các hàm tính tổng, hiệu, tích, thương. Trong hàm tính tích, sử dụng thêm mệnh đề if - else để tạo điều kiện, giới hạn phần tử a, b nhập vào phải nhỏ hơn 100000. Trong hàm tính thương, em cũng sử dụng tiệp mệnh đề if - else để phép tính có mẫu khác 0 thì mới thực hiện được.
* Sau đó, em tạo hàm main, khai báo kiểu dữ liệu cho a,b, key, luachon. Sau đó sử dụng vòng lặp do - while để thực hiện được nhiều phép tính.
* Hiển thị menu lựa chọn sẽ sử dụng hàm printf - sanf để hiển thị các lựa chọn .
* Để tạo menu tiếp theo em sử dụng lệnh switch - case là cấu trúc rẻ nhánh, tương tự if – else.
* luachon phải bắt buộc là giá trị hằng, có thể là biểu thức nhưng kết quả cần là hằng số.
* Trong đó, luachon sẽ được tính toán 1 lần duy nhất và sau đó so sánh với các giá trị của các case.
* Nếu có 1 case nào đó khớp giá trị, các khối lệnh tương ứng sau case đó sẽ được thực hiện cho tới khi gặp lệnh break. Do đó, nếu chúng ta không sử dụng break thì tất cả các case kể từ case khớp giá trị đều được thực hiện.
* Case default sẽ được thực hiện nếu không có case nào khớp giá trị với luachon.

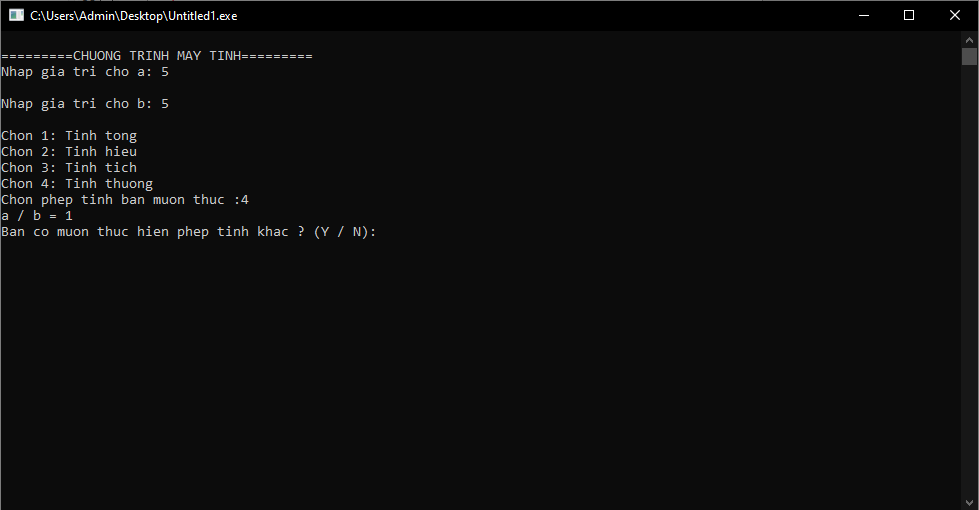
**3.3.Hướng dẫn sử dụng**



* Đây là giao diện để chúng ta nhập số a,b



* Sau khi nhập a, b. Sẽ xuất hiện menu chọn phép tính. Bấm chọn phép tính mong muốn.



* Kết quả sẻ hiện thị ngay bên dưới. Bấm Y nếu muốn thực hiện phép tính khác. Hoặc N để kết thúc chương trình.

CHƯƠNG 3 : TỔNG KẾT

**3.1.Nhược điểm của nhóm**

* Còn yếu về phần viết code, ý tưởng chưa được phong phú, đa dạng.
* Thời gian làm bài tập, báo cáo và thuyết trình vượt quà thời gian dự tính ban đầu của nhóm đưa ra.
* Còn mất thời gian về việc lên ý tưởng, chọn đề tài, phân công.

**3.2.Tài liệu tham khảo**

* sinhvientot.net
* vi.wikipedia.org
* nguyenvanhieu.vn
* daynhauhoc.com
* <https://www.facebook.com/groups/CNTT.VN.2019>
* quantrimang.com