**LINKED LIST**

***// tạo struct NODE***

*struct NODE { int data; NODE\* link; };*

***// tạo struct LIST***

*struct LIST { NODE\* first; NODE\* last; };*

***// INITIAL NODE >> tạo NODE***

*NODE\* setNode(int num) {*

*NODE\* node;*

*node = new NODE; // khoi tao vung nho*

*node->data = num;*

*node->link = NULL;*

*return node; }*

***// INITIAL LIST >> khởi tạo LIST***

*void initialList(LIST& list) {*

*list.first = NULL; list.last = list.first; }*

**// PRINT >> DUYỆT xuất LIST**

void coutList(LIST& list) {

NODE\* node = list.first;

while (node != NULL) {

cout << node->data << " ";

node = node->link; }

cout << endl; }

**// INSERT >> thêm vào đầu LIST**

void insertDataAtFirst(LIST& list, int num) {

NODE\* new\_node = setNode(num);

if (new\_node == NULL) { return;

} else {

if (list.first == NULL) {

list.first = new\_node;

list.last = list.first;

} else {

new\_node->link = list.first;

list.first = new\_node; } }}

**// INSERT >> thêm vào cuối**

void insertElementAtLast(LIST& list, int num) {

NODE\* new\_node = setNode(num);

if (new\_node == NULL) { return;

} else {

if (list.first == NULL) {

list.first = new\_node;

list.last = list.first;

} else {

list.last->link = new\_node;

list.last = new\_node; } }}

**// INSERT >> thêm sau NODE**

void insertNodeAfterNode(LIST& list, NODE\* node\_x, NODE\* new\_node) {

if (node\_x != NULL && new\_node != NULL) {

new\_node->link = node\_x->link;

node\_x->link = new\_node;

if (node\_x == list.last) {

list.last = new\_node; } } }

**// FIND >> tìm NODE = data**

NODE\* findNodeByData(LIST& list, int data) {

NODE\* node = list.first;

while (node != NULL && node->data != data) { node = node->link; }

return node; }

1

***// DELETE >> xóa NODE đầu tiên trong LIST***

*void deleteFirstNode(LIST& list) {*

*if (list.first != NULL) {*

*NODE\* node = list.first;*

*list.first = node->link;*

*if (list.first == NULL) {*

*list.last = NULL; }*

*delete node; } }*

***// DELETE >> xóa NODE cuoi cung trong LIST***

*void deleteLastNode(LIST& list) {*

*if (list.first != NULL) {*

*// nếu LIST chỉ có 1 phần tử*

*if (list.first == list.last) {*

*list.first = NULL;*

*list.last = NULL;*

*} else {*

*NODE\* anchor\_node = list.first;*

*NODE\* bye\_node = anchor\_node->link;*

*while (bye\_node != NULL) {*

*bye\_node = anchor\_node->link;*

*if (bye\_node == list.last) {*

*anchor\_node->link = NULL;*

*list.last = anchor\_node;*

*delete bye\_node;*

*} else {*

*anchor\_node = anchor\_node->link;*

*} } } } }*

**// DELETE >> xóa sau node**

void deleteNodeAfterNode(LIST& list, NODE\* anchor\_node) {

if (anchor\_node != NULL && anchor\_node->link != NULL) {

NODE\* bye\_node = anchor\_node->link;

anchor\_node->link = bye\_node->link;

if (bye\_node == list.last) {

list.last = anchor\_node; }

delete bye\_node; }}

***// DELETE >> xóa LIST***

*void deleteList(LIST& list) {*

*NODE\* tmp\_node;*

*while (list.first != NULL) {*

*tmp\_node = list.first;*

*list.first = tmp\_node->link;*

*free(tmp\_node); }*

*list.last = NULL;}*

***// SORT >> sắp xếp LIST mode: tang/giam - SELECTION SORT***

*void selectionSortList(LIST& list, string mode) {*

*for (NODE\* pre\_node = list.first; pre\_node != list.last;*

*pre\_node = pre\_node->link) {*

*NODE\* min\_node = pre\_node;*

*for (NODE\* next\_node = pre\_node->link; next\_node != NULL;*

*next\_node = next\_node->link) {*

if (mode == "tang") {

if (min\_node->data > next\_node->data) {

min\_node = next\_node;

}

} else if (mode == "giam") {

if (min\_node->data < next\_node->data) {

min\_node = next\_node; } } }

swap(min\_node->data, pre\_node->data);

}}

**STACK**

*#define max 100*

*struct STACK { int top; int list[max];};*

***// INITIAL >> khởi tạo stack rỗng***

*void initialStack(STACK& stack) {*

*stack.top = -1;}*

***// CHECK >> stack rỗng?***

*bool isEmptyStack(STACK stack) {*

*return stack.top == -1;}*

***// CHECK >> stack đầy?***

*bool isFullStack(STACK stack) {*

*return stack.top + 1 == max;}*

***// INSERT >> thêm vào top***

*void pushStack(STACK& stack, int data) {*

*stack.top++;*

*stack.list[stack.top] = data;}*

***// DELETE >> xóa top***

*void popStack(STACK& stack) {*

*if (!isEmptyStack(stack)) {*

*stack.top--; }}*

***// RETURN >> lấy top***

*int topStack(STACK stack) {*

*return stack.list[stack.top];}*

**// COUT >> xuất stack từ top trở xuống hoặc ngược lại**

void coutStack(STACK stack, string mode) {

if (mode == "fromtop") {

for (int i = stack.top; i > -1; i--) {

cout << stack.list[i].data; }

} else if (mode == "frombot") {

for (int i = 0; i <= stack.top; i++) {

cout << stack.list[i].data; } }

cout << endl;}

**// CHECK >> giá trị đã có trong stack hay chưa?**

bool isExistInStack(string data, stack<string> st) {

while (!st.empty()) {

if (st.top() == data) {

return true; }

st.pop(); }

return false;}

**TRUNG TỐ - HẬU TỐ**

**// GET >> độ ưu tiên của các toán tử \* / + -**

int getPriority(char toan\_tu) {

switch (toan\_tu) {

case '\*': return 4;

case '/': return 3;

case '+': return 2;

case '-': return 1;

default: return 0;

}}

**// CONVERT >> chuyển trung tố sang hậu tố**

string trungToSangHauTo(string trungto) {

string hau\_to;

STACK toan\_tu;

initialStack(toan\_tu);

for (size\_t i = 0; i < trungto.size(); i++) {

if (int(trungto[i]) >= 48 && int(trungto[i]) <= 57) {

hau\_to += trungto[i];

} else {

if (trungto[i] == '\*' || trungto[i] == '/' || trungto[i] == '+' ||

trungto[i] == '-') {

if (isEmptyStack(toan\_tu)) {

pushStack(toan\_tu, trungto[i]);

} else {

if (getPriority(topStack(toan\_tu)) <= getPriority(trungto[i])) {

pushStack(toan\_tu, trungto[i]);

}

if (getPriority(topStack(toan\_tu)) > getPriority(trungto[i])) {

hau\_to += topStack(toan\_tu);

popStack(toan\_tu);

pushStack(toan\_tu, trungto[i]);

}

}

} else {

if (trungto[i] == '(') {

pushStack(toan\_tu, trungto[i]);

}

if (trungto[i] == ')') {

while (topStack(toan\_tu) != '(') {

hau\_to += topStack(toan\_tu);

popStack(toan\_tu);

}

popStack(toan\_tu);

} } } }

while (!isEmptyStack(toan\_tu)) {

hau\_to += topStack(toan\_tu);

popStack(toan\_tu);

} return hau\_to; }

**// CALCULATE >> tính biểu thức hậu tố**

int tinhHauTo(string str) {

STACK stack;

initialStack(stack);

// a và b là 2 toán hạng

int a, b;

// lặp chuỗi

for (int i = 0; i < str.size(); i++) {

switch (str[i]) {

case '+':

a = topStack(stack);

popStack(stack);

b = topStack(stack);

popStack(stack);

pushStack(stack, a + b);

break;

case '-':

a = topStack(stack);

popStack(stack);

b = topStack(stack);

popStack(stack);

pushStack(stack, b - a);

break;

case '\*':

a = topStack(stack);

popStack(stack);

b = topStack(stack);

popStack(stack);

pushStack(stack, a \* b);

break;

case ' ':

break;

default:

string num;

// lặp đến khi gặp space

2

while (str[i] != ' ') {

num += str[i];

i++;

}

pushStack(stack, stringToInt(num));

num.clear();

break;

}

}

return topStack(stack);

}

**QUEUE**

#define MAX 100

**//KHAI BÁO HÀNG ĐỢI**

struct Queue {

int head, tail;

char array[MAX]; };

**//KHỞI TẠO**

void createQueue(Queue& q) {

q.head = 0; q.tail = 0; }

**//CHECK RỖNG**

bool isEmpty(Queue q) {

return (q.tail == q.head); }

**//CHECK ĐẦY**

bool isFull(Queue q)

{

if (q.head == 0 && q.tail == MAX - 1)

return true;

if (q.head == q.tail + 1)

return true;

return 0;

}

**//THÊM VÀO QUEUE**

void EnQueue(Queue& q, char text)

{

if (isEmpty(q)) {

q.array[q.tail] = text;

q.tail++; return;

}

if (!isFull(q)) {

q.array[q.tail] = text;

if (q.tail == MAX - 1) {

q.tail = 0; }

else { q.tail++; }

} }

**//LẤY RA**

char DeQueue(Queue& q)

{

if (!isEmpty(q)) {

char s = q.array[q.head];

if (q.head == MAX - 1)

q.head = 0;

else { q.head++; }

return s; } }

**SEARCH**

**// SEARCH >> TÌM KIẾM TUẦN TỰ (CHẬM NHẤT)**

int linearSearch(int a[], int len, int val) {

for (int i = 0; i < len; i++) {

if (a[i] == val) { return i; }

}

return -1; }

**// SEARCH >> TÌM KIẾM NHỊ PHÂN (NHANH HƠN LINEAR SEARCH, chỉ sử dụng với mảng đã có thứ tự)**

int binarySearch(int a[], int len, int val) {

int left = 0, right = len - 1;

while (left <= right) {

int mid = (left + right) / 2;

if (a[mid] == val) {

return mid; }

if (a[mid] > val) {

right = mid - 1;

} else {

left = mid + 1;

} }

return -1; }

**SORT**

**// SORT >> SẮP XẾP NỔI BỌT**

void bubbleSort(int a[], int len, string mode) {

int left = 0, shaker = left;

for (int i = left; i < len; i++) {

for (int j = len - 1; j > left; j--) {

if (mode == "tang") {

if (a[j] < a[j - 1]) {

swap(a[j], a[j - 1]);

shaker = j;

}

} else if (mode == "giam") {

if (a[j] > a[j - 1]) {

swap(a[j], a[j - 1]);

shaker = j;

} } }

left = shaker; } }

**// SORT >> SẮP XẾP CHỌN**

void selectionSort(int a[], int len, string mode) {

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

int min = i;

for (int j = i + 1; j < len; j++) {

if (mode == "giam") {

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

min = j; } }

if (mode == "tang") {

if (a[j] < a[min]) {

min = j; } }

}

if (min != i) {

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

}

}

}

**//HEAP SORT**

**// 1 - HIỆU CHỈNH**

void Shift(int num\_arr[], int left, int right) {

int x, curr, joint;

curr = left;

joint = 2 \* curr + 1;

x = num\_arr[curr];

while (joint <= right) {

if (joint < right) // neu co du 2 thanh phan lien doi

{

if (num\_arr[joint] < num\_arr[joint + 1]) {

joint = joint + 1;

}

}

if (num\_arr[joint] < x) // thoa quan he lien doi

{

break;

}

num\_arr[curr] = num\_arr[joint];

curr = joint;

joint = 2 \* curr + 1;

}

num\_arr[curr] = x;

}

**// 2 - TẠO HEAP**

void CreateHeap(int num\_arr[], int right) {

int left;

for (left = (right - 1) / 2; left >= 0; left--)

Shift(num\_arr, left, right - 1);

}

**// 3 - SẮP XẾP (TĂNG DẦN)**

void HeapSort(int a[], int N) {

int right;

CreateHeap(a, N);

right = N - 1;

while (right > 0) {

int tmp = a[0];

a[0] = a[right];

a[right] = tmp;

right--;

Shift(a, 0, right);

}

}

**//QUICK SORT**

#define Max 100

void QuickSort(int \*a, int left, int right) {

if (left >= right) {

return;

}

int x = a[(left + right) / 2];

int i = left, j = right;

while (i < j) {

while (a[i] < x) {

i++;

}

while (a[j] > x) {

j--;

}

if (i <= j) {

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

i++;

j--;

}

}

QuickSort(a, i, right);

QuickSort(a, left, j);

}

**TREE**

**// KHAI BÁO GỐC**

struct Node {

int data;

Node\* left;

Node\* right;

};

typedef struct Node\* Tree;

**// KHỞI TẠO GỐC**

void KhoiTaoNode(Node\* t) {

t->data = NULL;

t->left = t->right = NULL;

}

**// KHỞI TẠO CÂY**

void KhoiTaoTree(Tree& t) { t = NULL; }

**// TẠO GỐC**

3

Node\* GetNode(int x) {

Node\* p = new Node;

if (p != NULL) {

p->data = x;

p->left = NULL;

p->right = NULL;

}

return p;

}

**// THÊM VÀO CÂY**

void Insert(Tree& root, int x) {

if (root != NULL) {

if (root->data == x) return;

if (x < root->data) {

return Insert(root->left, x);

}

if (x > root->data) {

return Insert(root->right, x);

}

}

root = new Node;

root->data = x;

root->left = root->right = NULL;

}

**// DUYỆT GỐC - TRÁI - PHẢI**

void NLR(Tree t) {

if (t == NULL) return;

cout << t->data << " ";

if (t->left != NULL) {

NLR(t->left);

}

if (t->right != NULL) {

NLR(t->right);

}

}

**// DUYỆT TRÁI - PHẢI - GỐC**

void LRN(Tree t) {

if (t == NULL) return;

if (t->left != NULL) {

LRN(t->left);

}

if (t->right != NULL) {

LRN(t->right);

}

cout << t->data << " ";

}

**// DUYỆT TRÁI - NODE - PHẢI**

void LNR(Tree t) {

if (t == NULL) return;

if (t->left != NULL) {

LNR(t);

}

cout << t->data;

if (t->right != NULL) {

LNR(t);

}

}

**// TÌM NODE**

Node\* Search(Tree t, int x) {

if (t->data == x) return t;

if (x < t->data) {

return Search(t->left, x);

}

if (x > t->data) {

return Search(t->right, x);

}

return NULL;

}

**BONUS**

**// CONVERT >> chuỗi thành số ("123" -> 123)**

int stringToInt(string str) {

int result = 0;

for (size\_t i = 0; i < str.size(); i++) {

result = result \* 10 + str[i] - 48;

}

return result;

}

**// CONVERT >> số thành chuỗi 123 => "123"**

string tostr(int num) {

string rev;

// số thành chuỗi đảo ngược

while (num > 0) {

rev = rev + char(num % 10 + 48);

num /= 10;

}

//đảo ngược chuỗi

string res;

for (int i = rev.size() - 1; i >= 0; i--) {

res += rev[i];

}

return res;

}

**// CHECK >> số nguyên tố?**

bool isPrimeNumber(int num) {

if (num < 2) {

return false;

}

for (int i = 2; i <= sqrt(num); i++) {

if (num % i == 0) {

return false;

}

}

return true;

}

**// CHECK >> số nguyên tố cùng nhau**

bool isSNTCungNhau(int a, int b) {

while ((a != 0) && (b != 0)) {

if (b > a) {

b = b % a;

} else {

a = a % b;

}

}

if (b == 0) {

b = a;

}

if (b == 1) {

return true;

} else {

return false;

}

}

**// CHECK >> số chính phương? (là số có căn bậc 2 là một số nguyên)**

bool isSquareNumber(int num) {

if (sqrt(num) == floor(sqrt(num))) {

return true;

}

return false;

}

**// GET >> chiều dài kí tự của một số**

int lengthOfInt(int num) {

int length = 1;

while (num /= 10) {

length++;

}

return length;

}

**// REVERSE >> đảo ngược số: 1234 => 4321**

int reverseInt(int num) {

// lấy chiều dài số

int len = 1;

int tmp = num;

while (tmp /= 10) {

len++;

}

// tính toán

int rev = 0;

while (num > 0) {

rev = rev + (num % 10) \* (int)pow(10, len - 1);

len--;

num /= 10;

}

return rev;

}

**// GET >> giai thừa cua 1 số**

int factorial(int num) {

int factorial = 1;

for (int i = 0; i < num; i++) {

factorial \*= (num - i);

}

return factorial;

}

**// GET >> tổng chữ số: 123 = 1+2+3 = 6**

int sumOfInt(int num) {

int tong = 0;

while (num > 0) {

tong += num % 10;

num /= 10;

}

return tong;

}

**// GET >> ước chung lớn nhất của 2 số**

int UCLN(long a, long b) { return b == 0 ? a : UCLN(b, a % b); }

**// DATE >> trả về số ngày từ năm 0**

int serial\_julian\_date(int Month, int Day, int Year) {

int a = (14 - Month) / 12;

int m = Month + 12 \* a - 3;

int y = Year + 4800 - a;

return Day + (153 \* m + 2) / 5 + 365 \* y + (y / 4) - (y / 100) + (y / 400) -

32045;

}

**// CHECK >> chuỗi đối xứng?**

bool isPalindrome(string s) {

int foot = s.size() - 1;

int head = 0;

while (head <= foot) {

if (s[head] != s[foot]) {

return false;

}

head++;

foot--;

}

return true;

}

4

**// CONVERT >> chuyển sang chữ thường**

string toLowerCase(string s) {

for (int i = 0; i < s.size(); i++) {

if (s[i] >= 65 && s[i] <= 90) {

s[i] = s[i] + 32;

}

}

return s;

}

**// REVERSE >> đảo ngược chuỗi**

string reverseString(string str) {

string result = "";

for (int i = str.size() - 1; i >= 0; i--) {

result += str[i];

}

return result;

}

**// FILTER >> xóa khoảng trắng**

string spaceFilter(string s) {

string new\_s;

for (int i = 0; i < s.size(); i++) {

if (s[i] != ' ') {

new\_s += s[i];

}

}

return new\_s;

}

**// COMPARE >> so sánh 2 chuỗi theo kiểu từ điển**

bool operator>(string a, string b) {

// lấy độ dài ngắn hơn làm đk dừng loop

int len = a.size();

if (a.size() > b.size()) {

len = b.size();

}

// nếu có bất cứ kí tự nào của a so với b

for (int i = 0; i < len; i++) {

if (a[i] > b[i]) {

return true;

} // lớn hơn => true;

if (a[i] < b[i]) {

return false;

} // bé hơn => false;

// bằng thì tiếp tục vòng lặp

}

// hết vòng lặp thì chuỗi nào dài hơn sẽ lớn hơn

return a.size() > b.size();

/\*

b > a true

c > abc true

abc > ab true

ab > ab false

\*/

}

**//NHẬP XUẤT FILE**

#include <fstream>

int main() {

ifstream in("tenfile.xxx");

ofstream out("tenfile.yyy");

in.close();

out.close();

}

**QUÁ TẢI TOÁN TỬ**

**// KHAI BÁO**

struct PHANSO {

int tu, mau;

};

**// QUÁ TẢI NHẬP**

istream& operator>>(istream& is, PHANSO& ps) {

is >> ps.tu >> ps.mau;

return is;

}

**// QUÁ TẢI XUẤT**

ostream& operator<<(ostream& os, PHANSO ps) {

os << ps.tu << "/" << ps.mau;

return os;

}

**// TOÁN TỬ SO SÁNH**

bool operator==(PHANSO ps1, PHANSO ps2) {

return ps1.tu \* ps2.mau == ps1.mau \* ps2.tu;

}

**LÀM TRÒN**

**// lấy phần thực**

#include <iomanip>

double f = 3.14159; int x = 15;

cout << setfill('0') << setw(5) << x;

// output: 00015

cout << setprecision(5) << f << '\n';

// output: 3.1415

cout << setprecision(9) << f << '\n';

// output: 3.14159

cout << fixed;

cout << setprecision(5) << f << '\n';

// output: 3.14159

cout << setprecision(9) << f << '\n';

// output: 3.141590000

**// ROUND >> làm tròn số**

#include <math.h>

double val = 1.23456789;

cout << ceilf(val \* 1000) / 1000 << endl;

/\* 1.235 - lam tron len\*/

**cout << roundf(val \* 10000) / 10000 << endl; /\* 1.2346 - lam tron gan nhat\*/**

cout << floorf(val \* 100) / 100 << endl;

/\* 1.23 - lam tron xuong\*/