# Bài 1. Stack bằng danh sách liên kết (Linked Stack)

## Mô tả:

Quản lý một stack có số phần tử lớn, kiểu dữ liệu int, dùng danh sách liên kết đơn. Cần có các thao tác: khởi tạo, kiểm tra rỗng, push, pop, và ứng dụng đổi số thập phân sang nhị phân.

## Giải thuật:

1. Tạo cấu trúc Node (data, next).  
2. Stack chứa con trỏ top trỏ tới phần tử đầu.  
3. push(): cấp phát Node mới, gán next = top, rồi top = node.  
4. pop(): kiểm tra rỗng, nếu không thì lấy top->data, top = top->next.  
5. Ứng dụng: chia n cho 2, push phần dư, lặp đến 0, rồi pop để in ra nhị phân.

## Code:

#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct Node {  
 int data;  
 struct Node\* next;  
} Node;  
  
typedef struct {  
 Node\* top;  
} Stack;  
  
void init(Stack\* s){ s->top = NULL; }  
int isEmpty(Stack\* s){ return s->top == NULL; }  
  
void push(Stack\* s, int x){  
 Node\* p = (Node\*)malloc(sizeof(Node));  
 p->data = x; p->next = s->top; s->top = p;  
}  
  
int pop(Stack\* s){  
 if(isEmpty(s)) return -1;  
 Node\* t = s->top; int x = t->data;  
 s->top = t->next; free(t);  
 return x;  
}  
  
void toBinary(int n){  
 Stack s; init(&s);  
 while(n>0){ push(&s, n%2); n/=2; }  
 while(!isEmpty(&s)) printf("%d", pop(&s));  
 printf("\n");  
}  
  
int main(){  
 toBinary(25); // Output: 11001  
 return 0;  
}

## Giải thích:

Stack lưu phần dư theo thứ tự ngược, nên khi pop ra sẽ cho kết quả đúng. Cách dùng danh sách liên kết giúp không giới hạn kích thước stack.

## Kết quả mẫu:

Input: 25 → Output: 11001

# Bài 2. Queue bằng danh sách liên kết (Linked Queue)

## Mô tả:

Cài đặt queue số nguyên bằng danh sách liên kết đơn, hỗ trợ enqueue và dequeue.

## Giải thuật:

1. Dùng 2 con trỏ: front và rear.  
2. enqueue(): thêm node mới vào sau rear.  
3. dequeue(): lấy phần tử ở front và di chuyển front tới node kế tiếp.

## Code:

#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct Node {  
 int data;  
 struct Node\* next;  
} Node;  
  
typedef struct {  
 Node\* front;  
 Node\* rear;  
} Queue;  
  
void init(Queue\* q){ q->front = q->rear = NULL; }  
int isEmpty(Queue\* q){ return q->front == NULL; }  
  
void enqueue(Queue\* q, int x){  
 Node\* p = (Node\*)malloc(sizeof(Node));  
 p->data = x; p->next = NULL;  
 if(isEmpty(q)) q->front = q->rear = p;  
 else { q->rear->next = p; q->rear = p; }  
}  
  
int dequeue(Queue\* q){  
 if(isEmpty(q)) return -1;  
 Node\* t = q->front; int x = t->data;  
 q->front = t->next; if(q->front==NULL) q->rear=NULL; free(t);  
 return x;  
}  
  
int main(){  
 Queue q; init(&q);  
 enqueue(&q,10); enqueue(&q,20);  
 printf("%d ", dequeue(&q)); //10  
 printf("%d", dequeue(&q)); //20  
 return 0;  
}

## Giải thích:

Dữ liệu vào theo FIFO (vào trước ra trước). Con trỏ front chỉ phần tử đầu, rear chỉ phần tử cuối.

## Kết quả mẫu:

Output: 10 20

# Bài 3. Stack bằng mảng (Array Stack)

## Mô tả:

Cài đặt stack với mảng kích thước cố định MAX=100, thao tác qua chuỗi lệnh PUSH, POP, TOP, SIZE, END.

## Giải thuật:

1. Dùng mảng int stack[MAX]; biến top ban đầu = -1.  
2. PUSH: kiểm tra tràn, tăng top, gán phần tử.  
3. POP: kiểm tra rỗng, giảm top.  
4. TOP: in phần tử stack[top].

## Code:

#include <stdio.h>  
#include <string.h>  
#define MAX 100  
  
int stack[MAX], top=-1;  
  
void push(int x){ if(top==MAX-1) printf("FULL\n"); else stack[++top]=x; }  
void pop(){ if(top==-1) printf("EMPTY\n"); else top--; }  
void topv(){ if(top==-1) printf("EMPTY\n"); else printf("%d\n",stack[top]); }  
void size(){ printf("%d\n",top+1); }  
  
int main(){  
 char cmd[10]; int x;  
 while(1){  
 scanf("%s", cmd);  
 if(!strcmp(cmd,"PUSH")){ scanf("%d",&x); push(x); }  
 else if(!strcmp(cmd,"POP")) pop();  
 else if(!strcmp(cmd,"TOP")) topv();  
 else if(!strcmp(cmd,"SIZE")) size();  
 else if(!strcmp(cmd,"END")) break;  
 }  
 return 0;  
}

## Giải thích:

Dùng biến top quản lý chỉ số trên cùng của stack. Mỗi thao tác chỉ cần tăng/giảm top nên rất nhanh.

## Kết quả mẫu:

Input:  
PUSH 10  
PUSH 20  
TOP  
POP  
SIZE  
END  
Output:  
20  
1

# Bài 4. Kiểm tra dấu ngoặc & thẻ HTML-like

## Mô tả:

Dùng stack kiểm tra chuỗi ngoặc hợp lệ và cặp thẻ HTML-like dạng <tag></tag>.

## Giải thuật:

1. Với ngoặc: gặp ngoặc mở thì push, gặp ngoặc đóng thì pop và so khớp.  
2. Với thẻ: gặp <name> push tên thẻ, gặp </name> thì pop và kiểm tra trùng khớp.

## Code:

#include <stdio.h>  
#include <string.h>  
#define MAX 1000  
  
char stack[MAX][50]; int top=-1;  
  
void push(char\* s){ strcpy(stack[++top], s); }  
void pop(){ if(top>=0) top--; }  
  
int checkBrackets(char\* s){  
 char st[MAX]; int t=-1;  
 for(int i=0;s[i];i++){  
 char c=s[i];  
 if(c=='('||c=='['||c=='{') st[++t]=c;  
 else if(c==')'||c==']'||c=='}'){  
 if(t<0) return 0;  
 char o=st[t--];  
 if((c==')'&&o!='(')||(c==']'&&o!='[')||(c=='}'&&o!='{')) return 0;  
 }  
 }  
 return t==-1;  
}  
  
int main(){  
 char s1[]="([{}])";  
 printf("%s -> %s\n", s1, checkBrackets(s1)? "YES":"NO");  
 return 0;  
}

## Giải thích:

Khi gặp ngoặc mở thì lưu vào stack; gặp ngoặc đóng thì lấy phần tử trên cùng ra so sánh. Nếu tất cả cặp khớp, chuỗi hợp lệ.

## Kết quả mẫu:

Input: ([{}]) → Output: YES

# Bài 5. Chuyển biểu thức Trung tố → Hậu tố và tính giá trị

## Mô tả:

Chuyển biểu thức trung tố sang hậu tố bằng thuật toán Shunting-yard, sau đó tính giá trị hậu tố.

## Giải thuật:

1. Duyệt từng ký tự:  
 - Nếu là số: đưa vào output.  
 - Nếu là toán tử: so độ ưu tiên với đỉnh stack, pop ra khi cần.  
 - Nếu gặp '(': push; gặp ')': pop đến '('.  
2. Đánh giá hậu tố bằng stack số: gặp số thì push, gặp toán tử thì pop 2 phần tử để tính.

## Code:

#include <stdio.h>  
#include <string.h>  
#include <ctype.h>  
#define MAX 100  
  
int prec(char c){ if(c=='+'||c=='-') return 1; if(c=='\*'||c=='/') return 2; return 0; }  
  
void infixToPostfix(char\* exp){  
 char stack[MAX]; int top=-1;  
 for(int i=0;exp[i];i++){  
 char c=exp[i];  
 if(isdigit(c)) printf("%c ",c);  
 else if(c=='(') stack[++top]=c;  
 else if(c==')'){ while(top>=0 && stack[top]!='(') printf("%c ",stack[top--]); top--; }  
 else{  
 while(top>=0 && prec(stack[top])>=prec(c)) printf("%c ",stack[top--]);  
 stack[++top]=c;  
 }  
 }  
 while(top>=0) printf("%c ",stack[top--]);  
}  
  
int evalPostfix(char\* exp){  
 int stack[MAX], top=-1;  
 for(int i=0;exp[i];i++){  
 char c=exp[i];  
 if(isdigit(c)) stack[++top]=c-'0';  
 else if(c=='+'||c=='-'||c=='\*'||c=='/'){  
 int b=stack[top--], a=stack[top--];  
 if(c=='+') stack[++top]=a+b;  
 if(c=='-') stack[++top]=a-b;  
 if(c=='\*') stack[++top]=a\*b;  
 if(c=='/') stack[++top]=a/b;  
 }  
 }  
 return stack[top];  
}  
  
int main(){  
 char s[]="3+4\*(2-1)";  
 printf("Postfix: "); infixToPostfix(s);  
 printf("\nResult: %d\n", evalPostfix("3421-\*+"));  
 return 0;  
}

## Giải thích:

Thuật toán Shunting-yard giúp xác định thứ tự toán tử mà không cần đệ quy. Stack giúp xử lý dấu ngoặc và ưu tiên chính xác.

## Kết quả mẫu:

Input: 3+4\*(2-1)  
Output:  
Postfix: 3 4 2 1 - \* +  
Result: 7