***// PARENTHESES-MATCHING USING STACK***

#include <stdio.h>

#include <stdlib.h>

typedef struct List List;

typedef struct Node Node;

struct List

{

Node \*head;

int number\_of\_Nodes;

};

struct Node

{

int data;

Node \*link;

};

*/\* initializes a linked List \*/*

List\* List\_initialize(List\* List);

void List\_insert\_front(List\* List, int data);

*/\* Inserts data at specified position 0 <= position < length(List).*

*\* Returns position if insertion is successful, else -1*

*\*/*

int List\_insert\_at(List\* List, int data, int position);

void List\_delete\_front(List \*List);

*/\* Returns a pointer to the Node at specified index, if 0<=index<length(List).*

*\* Returns NULL otherwise*

*\*/*

Node\* List\_get(List \*List, int index);

/\* Returns the element that (\*Node\_ptr) contains \*/

const int Node\_get\_data(Node\* Node\_ptr);

*/\* Removes the last Node of the linked List \*/*

void List\_delete\_rear(List\* List);

void List\_reverse(List\* List);

void List\_print(List\* List);

/\* deallocates resources held by the List \*/

void List\_destroy(List\* List);

typedef struct

{

List \*ptr\_List;

}stack;

void stack\_initialize(stack\* ptr\_stack);

void stack\_push(stack\* ptr\_stack, int data);

void stack\_pop(stack\* ptr\_stack);

int stack\_is\_empty(stack\* ptr\_stack);

const int stack\_peek(stack\* ptr\_stack);

void stack\_destroy(stack\* ptr\_stack);

*/\* The implementations for the above functions are provided below main() \*/*

*/\* Returns 1 if the parenthesis sequence is well formed, else returns 0 \*/*

int match\_parenthesis(const char \* string);

int main()

{

char parenthesis\_sequence[100];

int number\_of\_inputs = 0;

scanf("%d", &number\_of\_inputs);

for(int counter = 0; counter < number\_of\_inputs; ++ counter)

{

scanf("%s", parenthesis\_sequence);

printf("%d\n", match\_parenthesis(parenthesis\_sequence));

}

return 0;

}

List\* List\_initialize(List\* list)

{

list=(List\*)malloc(sizeof(List));

list->head=NULL;

return list;

}

void List\_insert\_front(List\* list, int data)

{

Node \*p=(Node\*)malloc(sizeof(Node));

p->data=data;

p->link=list->head;

list->head=p;

}

void List\_delete\_front(List \*list)

{

if(list->head==NULL)

return;

Node \*temp=list->head;

list->head=list->head->link;

free(temp);

}

void List\_destroy(List\* List)

{

while(List->head!=NULL)

{

List\_delete\_front(List);

}

}

void stack\_initialize(stack\* ptr\_stack)

{

ptr\_stack->ptr\_list=List\_initialize(ptr\_stack->ptr\_list);

}

void stack\_push(stack\* ptr\_stack, int data)

{

List\_insert\_front(ptr\_stack->ptr\_list,data);

}

void stack\_pop(stack\* ptr\_stack)

{

if(ptr\_stack->ptr\_list->head!=NULL)

{

List\_delete\_front(ptr\_stack->ptr\_list);

}

}

int stack\_is\_empty(stack\* ptr\_stack)

{

if(ptr\_stack->ptr\_list->head==NULL)

return 1;

else

return 0;

}

const int stack\_peek(stack\* ptr\_stack)

{

return (ptr\_stack->ptr\_list->head->data);

}

void stack\_destroy(stack\* ptr\_stack)

{

List \*temp=ptr\_stack->ptr\_list;

ptr\_stack->ptr\_list=NULL;

List\_destroy(temp);

ptr\_stack=NULL;

}

int match\_parenthesis(const char \* string)

{

stack new\_stack;

stack\_initialize(&new\_stack);

for(int i=0;i<100;i++)

{

switch(\*(string+i))

{

case'{':

case'(':

case'[':

{ stack\_push(&new\_stack,\*(string+i));

break;

}

case'}':

{

if(stack\_is\_empty(&new\_stack)||stack\_peek(&new\_stack)!='{')

{

stack\_destroy(&new\_stack);

return 0;

}

stack\_pop(&new\_stack);

break;

}

case']':

{

if(stack\_is\_empty(&new\_stack)||stack\_peek(&new\_stack)!='[')

{

stack\_destroy(&new\_stack);

return 0;

}

stack\_pop(&new\_stack);

break;

}

case')':

{

if(stack\_is\_empty(&new\_stack)||stack\_peek(&new\_stack)!='(')

{

stack\_destroy(&new\_stack);

return 0;

}

stack\_pop(&new\_stack);

break;

}

default:

{

if(stack\_is\_empty(&new\_stack))

{

stack\_destroy(&new\_stack);

return 1;

}

else return 0;

}

}

}

}