Estructuras de Datos

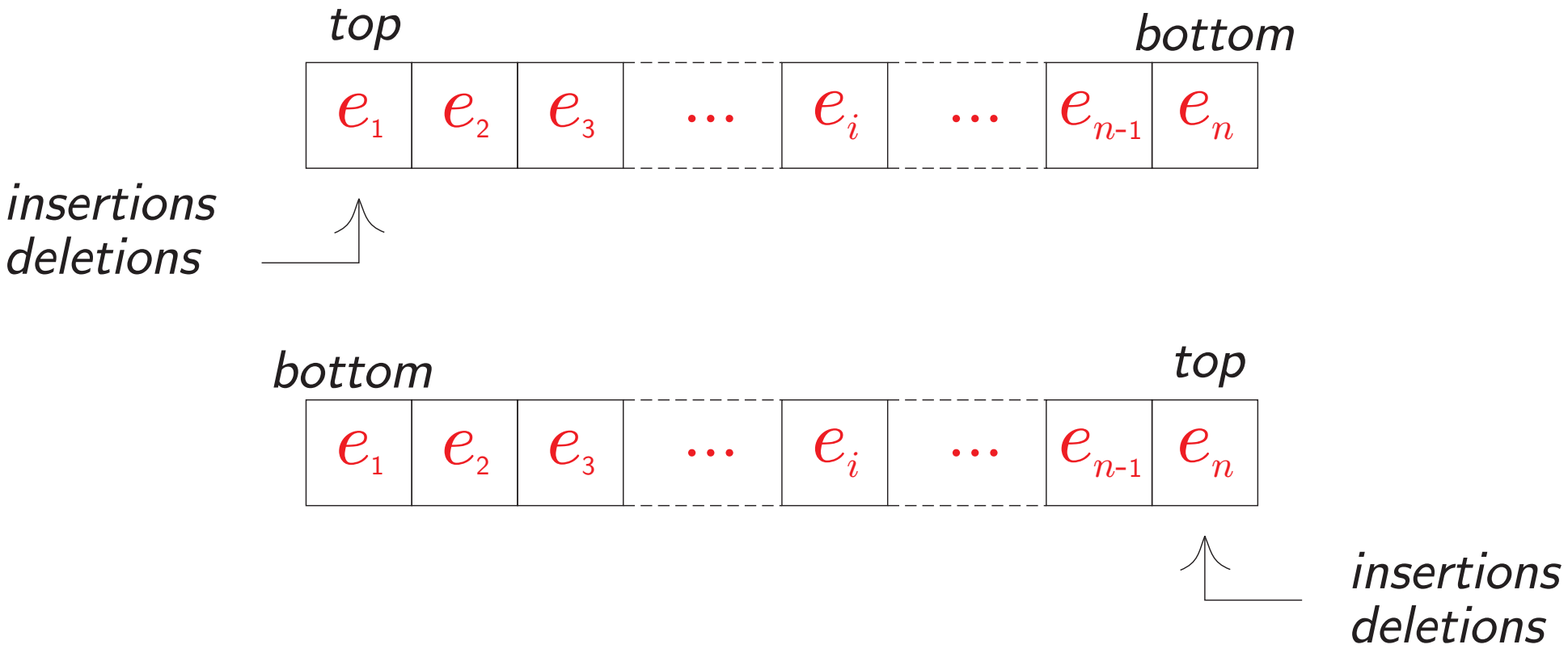
Stack Data Structure

Session 5

- **Stack Data Structure**
  - ▷ Array-based Representation
  - ▷ Linked Representation
  - ▷ Applications
    - ◇ Parentheses Matching
    - ◇ Switchbox Routing

Stacks Data Structure

A *stack* is a linear list in which insertions (also called additions) and deletions take place at the *same* end. This end is called the *top*. The other end is called the *bottom*.



In other words, a stack is a **LIFO** (last-in-first out) list

Stack Data Structure

– Array-based Representation –

Basic design decision:

> *designate the left end of the array as the bottom and the right end as the top*

arraylist.h

```
#ifndef ARRAYSTACK
#define ARRAYSTACK

#include <stdio.h>
#include <stdlib.h>
#define CAP 100

typedef struct Stack Stack;

struct Stack{
   int a[ CAP ];
   int top;
   void (*push) ( Stack *, int );
   void (*pop) ( Stack * );
   int (*peek) ( Stack * );
   void (*display) ( Stack * );
   int (*empty) ( Stack * );
};

void push( Stack * x, int e ){ ... }
void pop( Stack * x ){ ... }
int peek( Stack * x ){ ... }
void display( Stack * x ){ ... }
int empty( Stack * x ){ ... }
Stack createStack( ){ ... }

#endif
```

createStack

```
Stack createStack( ){
   Stack s;
   s.top = -1;
   s.push = &push;
   s.pop = &pop;
   s.peek = &peek;
   s.display = &display;
   s.empty = &empty;
   return s;
}
```

push, pop

```
void push( Stack * x, int e ){
   if( x->top == CAP - 1 ){
      fprintf( stderr, "Error: stack is full\n" );
      return;
   }
   x->a[ ++x->top ] = e;
   return;
}


void pop( Stack * x ){
   if( x->top == -1 ){
      fprintf( stderr, "Error: stack is empty\n" );
      return;
   }
   x->top--;
   return;
}
```

peek, display, empty

```
int peek( Stack * x ){
   if( x->top == -1 ){
      fprintf( stderr, "Error: stack is empty\n" );
      return;
   }
   return x->a[ x->top ];
}


void display( Stack * x ){
   int i;
   printf( "Stack: [top] " );
   for( i = x->top; i >= 0; i-- )
      printf( "%d ", x->a[ i ] );
   printf( "[bottom]\n" );
   return;
}


int empty( Stack * x ){
   return x->top == -1;
}
```

teststack.c

```c
#include <stdio.h>
#include "arraystack.h"

int main(){
   Stack maria = createStack( );
   maria.push( &maria, 100 );
   maria.push( &maria, 80 );
   maria.push( &maria, 15 );
   maria.push( &maria, 24 );
   maria.push( &maria, 135 );
   maria.display( &maria );
   maria.pop( &maria );
   maria.display( &maria );
   while( !maria.empty( &maria ) ){
      printf( "Deleting␣top␣element␣%d\n", maria.peek( &maria ) );
      maria.pop( &maria );
   }
   return 0;
}
```

Compilation and Running

```
C:\Users\Yoan\Desktop\code\progs>gcc teststack.c

C:\Users\Yoan\Desktop\code\progs>a
Stack: [top] 135 24 15 80 100 [bottom]
Stack: [top] 24 15 80 100 [bottom]
Deleting top element 24
Deleting top element 15
Deleting top element 80
Deleting top element 100

C:\Users\Yoan\Desktop\code\progs>
```

Complexity of Operations

$$
\begin{aligned}
\text{push} &: \Theta(1) \\
\text{pop} &: \Theta(1) \\
\text{peek} &: \Theta(1) \\
\text{display} &: \Theta(n) \\
\text{empty} &: \Theta(1) \\
\text{createStack} &: \Theta(1)
\end{aligned}
$$

Stack Data Structure

– Linked Representation –

Basic design decision:

> *designate the left end of the list as the top and the other end as the bottom*

linkedlist.h

```
#ifndef LINKEDSTACK
#define LINKEDSTACK

#include <stdio.h>
#include <stdlib.h>

typedef struct Stack Stack;
typedef struct Node Node;

struct Node{
   int item;
   Node * next;
};

struct Stack{
   Node * top;
   void (*push) ( Stack *, int );
   void (*pop) ( Stack * );
   int (*peek) ( Stack * );
   void (*display) ( Stack * );
   int (*empty) ( Stack * );
};
```

```
void push( Stack * x, int e ){ ... }
void pop( Stack * x ){ ... }
int peek( Stack * x ){ ... }
void display( Stack * x ){ ... }
int empty( Stack * x ){ ... }
Stack createStack( ){ ... }

#endif
```

createStack

```
Stack createStack( ){
   Stack s;
   s.top = NULL;
   s.push = &push;
   s.pop = &pop;
   s.peek = &peek;
   s.display = &display;
   s.empty = &empty;
   return s;
}
```

push, pop

```
void push( Stack * x, int e ){
   Node * y = malloc( sizeof( Node ) );
   y->item = e;
   y->next = x->top;
   x->top = y;
   return;
}
```

```
void pop( Stack * x ){
   if( x->top == NULL ){
      fprintf( stderr, "Error: stack is empty\n" );
      return;
   }
   Node * y = x->top;
   x->top = x->top->next;
   free( y );
   return;
}
```

peek, display, empty

```
int peek( Stack * x ){
   if( x->top == NULL ){
      fprintf( stderr, "Error: stack is empty\n" );
      return;
   }
   return x->top->item;
}

void display( Stack * x ){
   Node * y = x->top;
   printf( "Stack: [top] " );
   while( y ){
      printf( "%d ", y->item );
      y = y->next;
   }
   printf( "[bottom]\n" );
   return;
}

int empty( Stack * x ){
   return x->top == NULL;
}
```

teststack.c

```c
#include <stdio.h>
#include "linkedstack.h"

int main(){
   Stack maria = createStack( );
   maria.push( &maria, 100 );
   maria.push( &maria, 80 );
   maria.push( &maria, 15 );
   maria.push( &maria, 24 );
   maria.push( &maria, 135 );
   maria.display( &maria );
   maria.pop( &maria );
   maria.display( &maria );
   while( !maria.empty( &maria ) ){
      printf( "Deleting␣top␣element␣%d\n", maria.peek( &maria ) );
      maria.pop( &maria );
   }
   return 0;
}
```

Compilation and Running

```
C:\Users\Yoan\Desktop\code\progs>gcc teststack.c

C:\Users\Yoan\Desktop\code\progs>a
Stack: [top] 135 24 15 80 100 [bottom]
Stack: [top] 24 15 80 100 [bottom]
Deleting top element 24
Deleting top element 15
Deleting top element 80
Deleting top element 100

C:\Users\Yoan\Desktop\code\progs>
```

Complexity of Operations

$$
\begin{aligned}
\text{push} &: \Theta(1) \\
\text{pop} &: \Theta(1) \\
\text{peek} &: \Theta(1) \\
\text{display} &: \Theta(n) \\
\text{empty} &: \Theta(1) \\
\text{createStack} &: \Theta(1)
\end{aligned}
$$

Stack Application

– Parentheses Matching –

How do we match parentheses in an expression?

- `(((a+b)*c+d*e)/((f+g)-h+i))`
- `(a*(a+b))/(b+d)`

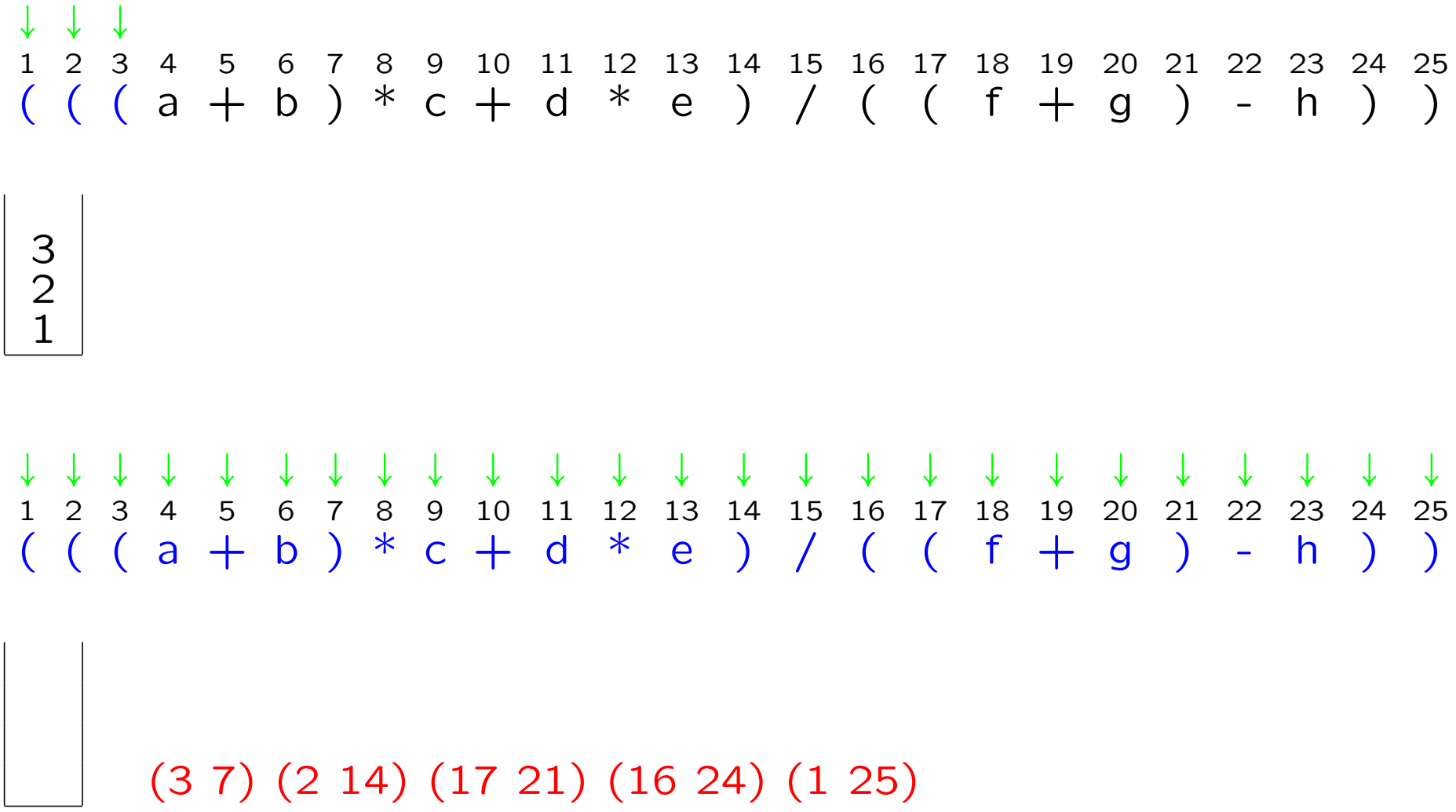
Parentheses Matching

– Strategy –

- Scan expression from left to right
- When a left parenthesis is encountered, add its position to the stack
- When a right parenthesis is encountered, remove matching position from stack

Parentheses Matching

– Example –



Parentheses Matching
– Implementation –

parenthesismatching.c

```
#include <stdio.h>
#include <string.h>
#include "arraystack.h"

void printMatchedPairs( char* expr ){ ... }

int main( ){
   char x[256];
   printf( "Type␣an␣expression␣\n" );
   gets( x );
   printf( "The␣pairs␣of␣matching␣parentheses␣in␣%s␣are:\n", x );
   printMatchedPairs( x );
   return 0;
}
```

printMatchedPairs

```
void printMatchedPairs( char* expr ){
   int i;
   Stack s = createStack( );
   int n = strlen( expr );
   for( i = 0; i < n; i++ )
      if( expr[ i ] == '(' )
         s.push( &s, i );
      else if( expr[ i ] == ')' )
         if( !s.empty( &s ) ){
             printf( "%d␣%d\n", s.peek( &s ), i );
             s.pop( &s );
         }
         else{
            printf( "No␣match␣for␣right␣parenthesis␣at␣%d\n", i );
         }
   while( !s.empty( &s ) ){
      printf( "No␣match␣for␣left␣parenthesis␣at␣%d\n", s.peek( &s ) );
      s.pop( &s );
   }
}
```

Compilation and Running

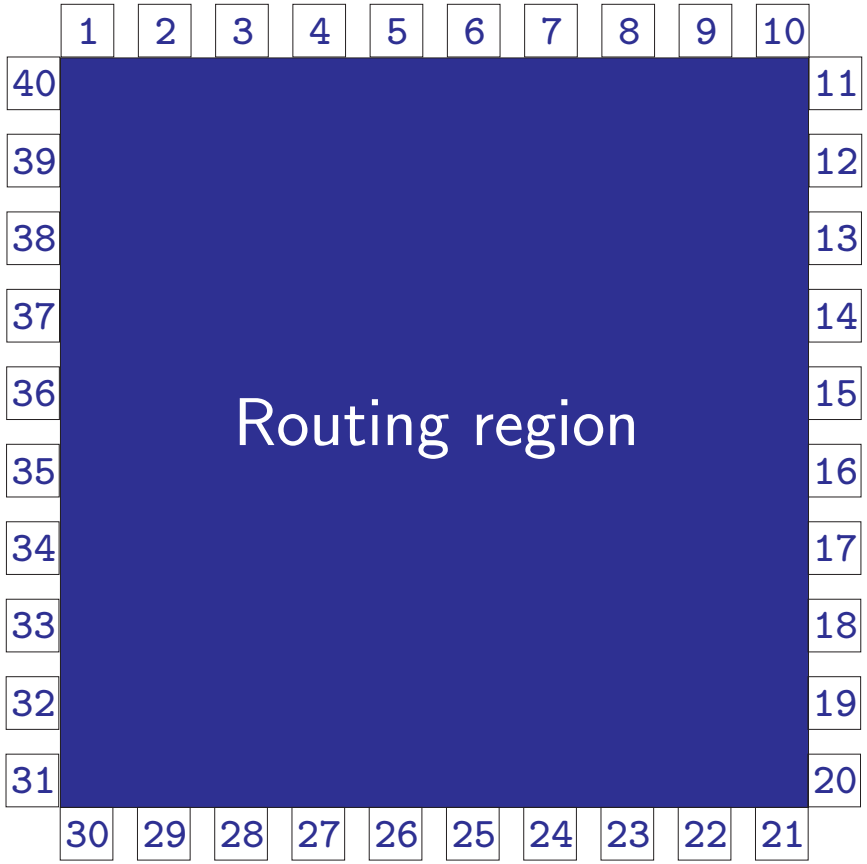
```
C:\Users\Yoan\Desktop\code\progs>gcc parenthesismatching.c

C:\Users\Yoan\Desktop\code\progs>a
Type an expression
(((a+b)*c+d*e)/((f+g)-h+i))
The pairs of matching parentheses in (((a+b)*c+d*e)/((f+g)-h+i)) are:
2 6
1 13
16 20
15 25
0 26
```

Stack Application

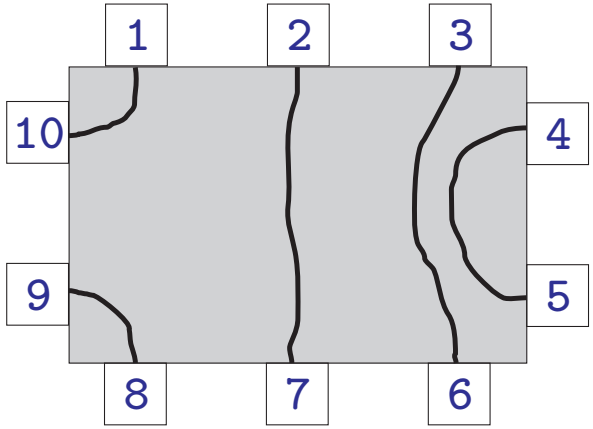
– Switchbox Routing –

The switchbox routing problem arises in the fabrication of computer chips, where certain components need to be connected to other components.



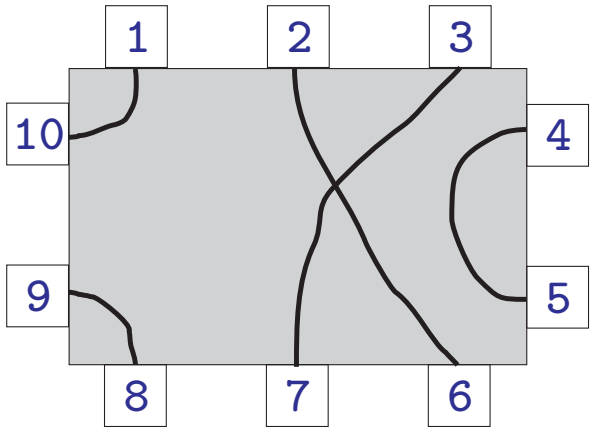
Switchbox Routing
– Example –

$$\text{Net}=\{\ \overset{1}{1},\ \overset{2}{2},\ \overset{3}{3},\ \overset{4}{4},\ \overset{5}{4},\ \overset{6}{3},\ \overset{7}{2},\ \overset{8}{5},\ \overset{9}{5},\ \overset{10}{1}\ \}$$



Routable !

$$\text{Net}=\{\ \overset{1}{1},\ \overset{2}{2},\ \overset{3}{3},\ \overset{4}{4},\ \overset{5}{4},\ \overset{6}{2},\ \overset{7}{3},\ \overset{8}{5},\ \overset{9}{5},\ \overset{10}{1}\ \}$$



NOT Routable!

Switchbox Routing
– Implementation –

switchbox.c

```
#include <stdio.h>
#include "arraystack.h"

void checkBox( int * net, int n ){ ... }

int main( ){
   int i, n, * net;
   printf( "Type␣number␣of␣pins␣in␣switch␣box\n" );
   scanf( "%d", &n );
   net = ( int * ) malloc( sizeof( int ) * n );
   printf( "Type␣net␣numbers␣for␣pins␣1␣through␣%d\n", n );
   for( i = 0; i < n; i++ )
      scanf( "%d", &net[ i ] );
   checkBox( net, n );
   return 0;
}
```

checkBox

```
void checkBox( int * net, int n ){
   int i;
   Stack s = createStack( );
   for( i = 0; i < n; i++ )
      if( !s.empty( &s ) )
         if( net[ i ] == net[ s.peek( &s ) ] )
            s.pop( &s );
         else s.push( &s, i );
      else s.push( &s, i );
   if( s.empty( &s ) ){
      printf( "Switch box is routable\n" );
      return;
   }
   printf( "Switch box is not routable\n" );
   return;
}
```

Compilation and Running

```
C:\Users\Yoan\Desktop\code\progs>gcc switchbox.c

C:\Users\Yoan\Desktop\code\progs>a
Type number of pins in switch box
10
Type net numbers for pins 1 through 10
1 2 3 4 4 3 2 5 5 1
Switch box is routable

C:\Users\Yoan\Desktop\code\progs>a
Type number of pins in switch box
10
Type net numbers for pins 1 through 10
1 2 3 4 4 2 3 5 5 1
Switch box is not routable

C:\Users\Yoan\Desktop\code\progs>
```