0522042 Programlama Dilleri II
Lab Session Week 6
14/04/2011

Y.T.Ü. Matematik Müh.
```c
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

void F(double *vec, int n)
{
    double mval;
    int im=0, i;
    if (n == 1) return;
    mval = vec[0];
    for(i=1;i<n;i++)
        if (vec[i]<mval) {
            mval = vec[i];
            im = i;
        }
    vec[im] = vec[0];
    vec[0] = mval;
    F(vec + 1, n -1);
    // F(&vec[1], n - 1);
    return;
}

int main()
{
    int i;
    double a[]={1.2,5,-2.4,8,0};
    F(a, 5);
    for(i=0;i<5;i++)
        printf("a[%d]...=%lf\n",i,*a+i);
    getch();
    return 0;
}
```

What will be the output?
The goal of today’s session:
➢ To practice and learn how to use self-refencial structures in C: Case Study STACKS

Applications

Stacks are ubiquitous in the computing world.

Expression evaluation and syntax parsing

Calculators employing reverse Polish notation use a stack structure to hold values. Expressions can be represented in prefix, postfix or infix notations. Conversion from one form of the expression to another form may be accomplished using a stack. Many compilers use a stack for parsing the syntax of expressions, program blocks etc. before translating into low level code. Most of the programming languages are context-free languages allowing them to be parsed with stack based machines.

Example (general)

The calculation: 1 + 2 * 4 + 3 can be written down like this in postfix notation with the advantage of no precedence rules and parentheses needed:

```
1 2 4 * + 3 +
```

The expression is evaluated from the left to right using a stack:

1. when encountering an operand: push it
2. when encountering an operator: pop two operands, evaluate the result and push it.

Like the following way (the Stack is displayed after Operation has taken place):

<table>
<thead>
<tr>
<th>Input</th>
<th>Operation</th>
<th>Stack (after op)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Push operand 1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Push operand 2, 1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Push operand 4, 2, 1</td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>Multiply 8, 1</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>Add 9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Push operand 3, 9</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>Add 12</td>
<td></td>
</tr>
</tbody>
</table>

The final result, 12, lies on the top of the stack at the end of the calculation.

Today’s Example problem to work on:

> Write and test a C code that computes the result of the following expression using stacks

\[((5 \times 9) + 8) \times ((4 \times 6) + 7)\]

\[5 \times 9 + 4 \times 6 + 7 + \ast\]

```c
struct stackNode
{
    int data;
    struct stackNode *nextPtr;
};
typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;
StackNodePtr push( StackNodePtr, int );
StackNodePtr pop( StackNodePtr, int * );
int isEmpty( StackNodePtr );
void printStack( StackNodePtr );
int main()
{
    StackNodePtr stackPtr = NULL;
    ...
```
Homework(s)

How do you Implement a Single Link List in C?