Tutorato Lezione 9

Tutor: Maria Angela Pellegrino
mapellegrino@unisa.it

Esercizio 1

Realizzare delle funzioni che copi il contenuto di uno Stack in una Lista (mantenendo il contenuto dello Stack).

Esercizio 1 - stack to list

```
List toList(Stack s) {
        List l = newList();
        Stack s2 = newStack();
        while(!isEmptyStack(s)) {
                Item it = top(s);
                pop(s);
                addHead(l, it);
                push(s2, it);
        }
        while(!isEmptyStack(s2)) {
                Item it = top(s2);
                pop(s2);
                push(s, it);
        }
        return l;
}
```

Esercizio 2

Stampare gli elementi di uno Stack
in maniera ricorsiva

Esercizio 2 - print stack ricorsiva

```
void printItems(Item *items, int n) {
        outputItem(*items);
        if(n > 1) {
                printItems(items + 1, n - 1);
        }
}

void printStack(Stack s) {
        printItems(s->elements, s->top);
}
```

Esercizio 3

Aggiungere all'ADT Lista l'operatore List **newRandomList**(int n) che restituisce una lista con n elementi casuali. Fare in modo che possa funzionare con qualsiasi tipo di Item

Esercizio 3 - newRandom List

```
List newRandomList(int n) {
        List l = newList();
        for(int i = 0; i < n; i++) {
                addHead(l,
randomItem());
        }
        return l;
}
```

in *item.h* Item randomItem();

in *item-int.c*

```
Item randomItem(){
        int *pt = malloc(sizeof(int));
        *pt = rand() % 100;
        return pt;
}
```

Esercizio 3 - newRandom List

```
List newRandomList(int n) {
        List l = newList();
        for(int i = 0; i < n; i++) {
                addHead(l,
randomItem());
        }
        return l;
}
```

in *item.h* Item randomItem();

in *item-string.c*

```
Item randomItem(){
        int n = rand() % 100;
        char* str = malloc(sizeof(char) * 3);
        snprintf(str, 3, "%d", n);
        return str;
}
```

Esercizio 4

Dato un intero n, implementare un algoritmo che generi un albero binario con n nodi, casuale per contenuto dei nodi e struttura.

Esercizio 4 - newRandom Tree

```
BTree newRandomTree(int nNodes) {
        if(nNodes <= 0)
        return NULL;

        BTree root = malloc(sizeof(struct
node));
        root->value = randomItem();

        BTree freeNodes[nNodes];
        int freeSize = 1;
        freeNodes[0] = root;
        ...
```

```
...
    for(int i = 1; i < nNodes; i++) {
                BTree node = malloc(sizeof(struct node));
                node->value = randomItem();
                int r = rand() % freeSize;
                if(freeNodes[r]->left == NULL && freeNodes[r]->right ==
NULL) {
                        if(rand() % 2) freeNodes[r]->left = node;
                        else freeNodes[r]->right = node;
                } else {
                        if(freeNodes[r]->right) freeNodes[r]->left =
node;
                        else if(freeNodes[r]->left) freeNodes[r]->right =
node;
                        else printf("non dovrei mai finire qui\n");
                        freeSize--;
                        for(int j = freeSize; j > r; j--)          freeNodes[j
- 1] = freeNodes[j];
                }
```

Esercizio 5

Progettare e implementare un **ADT** per le **espressioni aritmetiche**
(ad es. ((6.1-7)+((5/3)*4))).
– Rappresentare le espressioni mediante un albero binario.
– I nodi intermedi che rappresentano gli operatori.
– Le foglie rappresentando gli operandi.
– Fornire operatori che consentano di definire somma, sottrazione, moltiplicazione divisione, numero.
– Consentire la stampa dell'espressione.
– Consentire il calcolo del valore dell'espressione.

Esercizio 5 - espressione aritmetica ADT

Expr:

op → char
val → double
left → Expr
right → Expr

eNum(double) → Expr
eSum(Expr, Expr) → Expr
eSub(Expr, Expr) → Expr
eMul(Expr, Expr) → Expr
eDiv(Expr, Expr) → Expr

Esercizio 5 - espressione aritmetica ADT

expr.h

```
typedef struct node *Expr;

Expr eNum(double);
Expr eSum(Expr, Expr);
Expr eSub(Expr, Expr);
Expr eMul(Expr, Expr);
Expr eDiv(Expr, Expr);

void printExpr(Expr e);
double eEval(Expr e);
```

Esercizio 5 - espressione aritmetica ADT

expr.c

```
struct node{
        char op;
        double val;
        struct node *left;
        struct node *right;
};
Expr eNum(double val) {
        Expr a = malloc(sizeof(struct node));
        a->op = '\0';
        a->val = val;
        a->left = NULL;
        a->right = NULL;
        return a;
}
```

Esercizio 5 - espressione aritmetica ADT

expr.c

```
Expr eSum(Expr l, Expr r) {
        Expr a = malloc(sizeof(struct node));
        a->op = '+';
        a->val = 0;
        a->left = l;
        a->right = r;
        return a;
}
```

Esercizio 5 - espressione aritmetica ADT

expr.c

```
Expr eSub(Expr l, Expr r) {
        Expr a = malloc(sizeof(struct node));
        a->op = '-';
        a->val = 0;
        a->left = l;
        a->right = r;
        return a;
}
```

Esercizio 5 - espressione aritmetica ADT

expr.c

```
Expr eMul(Expr l, Expr r) {
        Expr a = malloc(sizeof(struct node));
        a->op = '*';
        a->val = 0;
        a->left = l;
        a->right = r;
        return a;
}
```

Esercizio 5 - espressione aritmetica ADT

expr.c

```
Expr eDiv(Expr l, Expr r)  {
        Expr a = malloc(sizeof(struct node));
        a->op = '/';
        a->val = 0;
        a->left = l;
        a->right = r;
        return a;
}
```

Esercizio 5 - espressione aritmetica ADT

expr.c

```
void printExpr(Expr e) {
        if(e->op == '\0') {
                printf("%g", e->val);
        } else {
                printf("(");
                printExpr(e->left);
                printf("%c", e->op);
                printExpr(e->right);
                printf(")");
        }
}
```

expr.c

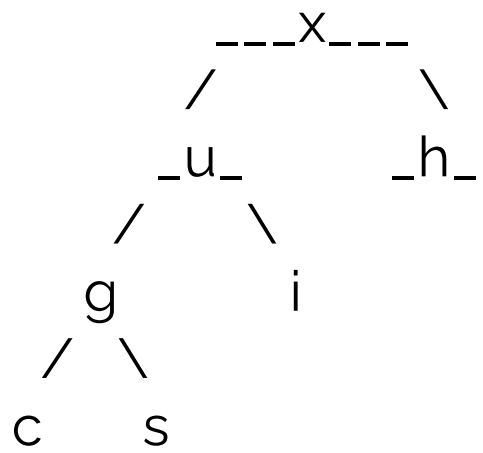
```
double eEval(Expr e) {
        switch(e->op) {
                case '\0': return e->val; break;
                case '+' : return eEval(e->left) + eEval(e->right); break;
                case '-' : return eEval(e->left) - eEval(e->right); break;
                case '*' : return eEval(e->left) * eEval(e->right); break;
                case '/' : return eEval(e->left) / eEval(e->right); break;
        }
}
```

*((6.1-7)+((5/3)*4)))*

```
int main() {
        Expr e = eSum(
        eSub(eNum(6.1), eNum(7)),
        eMul(
            eDiv(eNum(5), eNum(3)),
            eNum(4)
        )
    );
        printf("e: ");
        printExpr(e);
        printf("\neval: %g\n", eEval(e));
        return 0;
}
```

Esercizio 6

Stampare un albero binario come ASCII art (vedi esempio in basso) –
Si può ipotizzare che tutti i nodi dell'albero contengano un singolo carattere



```
void printTree(BTree bt){
        if (isEmptyTree(bt)) return;
        int height = 0, ndsLastLv = 0, qsize = 0, qsizeNoNulls = 0;
        List items = newList();
        Queue q = newQueue();
        enqueue(q, bt);
        qsize++; qsizeNoNulls++;
    while(qsizeNoNulls > 0) {
                for (ndsLastLv = qsize; ndsLastLv > 0; ndsLastLv--) {
                        BTree left, right;
                        BTree node = dequeue(q); qsize--;
                        if(node) {
                                addListTail(items, node->value);
                                qsizeNoNulls--;
                        } else { addListTail(items, NULL); }
                        left = getLeft(node);
                        enqueue(q, left); qsize++;
                        if(left) qsizeNoNulls++;
                        right = getRight(node);
                        enqueue(q, right); qsize++;
                        if(right) qsizeNoNulls++;
                }
                height++;
        } ...
```

```
...
        int n = sizeList(items);
        for(int i = 1, pad = n / 2; i < n; i *= 2, pad /= 2) {
                Item lv[i];
                if(i == 1) { lv[0] = removeHead(items); }
                else {
                        for(int j = 0; j < i; j++) {
                                lv[j] = removeHead(items);
                                if(lv[j]) {
                                        printf("%*s", pad, "");
                                        printf("%c", j % 2 ? '\\' : '/');
                                        printf("%*s", pad, "");

                                } else {
                                        printf("%*s", (pad*2)+1, "");
                                }
                                printf(" ");
                        }
                        printf("\n");
                }
                ...
```

```
...
                    for(int j = 0; j < i; j++) {
                        if(lv[j]) {
                            printf("%*s", (pad+1)/2, "");
                            for(int k = pad/2; k > 0; k--)
                                printf("_");
                            outputItemLen(lv[j], 1);
                            for(int k = pad/2; k > 0; k--)
                                printf("_");
                            printf("%*s", (pad+1)/2, "");
                        } else {
                            printf("%*s", (pad*2)+1, "");
                        }
                        printf(" ");
                    }
                    printf("\n");
                }
            }
```