Assignment 1, TDT4205

Kim Rune Solstad

January 16, 2014

Part 1, Theory

Problem 1

Login gikk greit ved å bruke

```
ssh kmirs@stud.ntnu.no
```

Version til gcc er 4.6.3. Version til flex er 2.5.35. Version til bison er 2.5. Fant disse ved å bruke

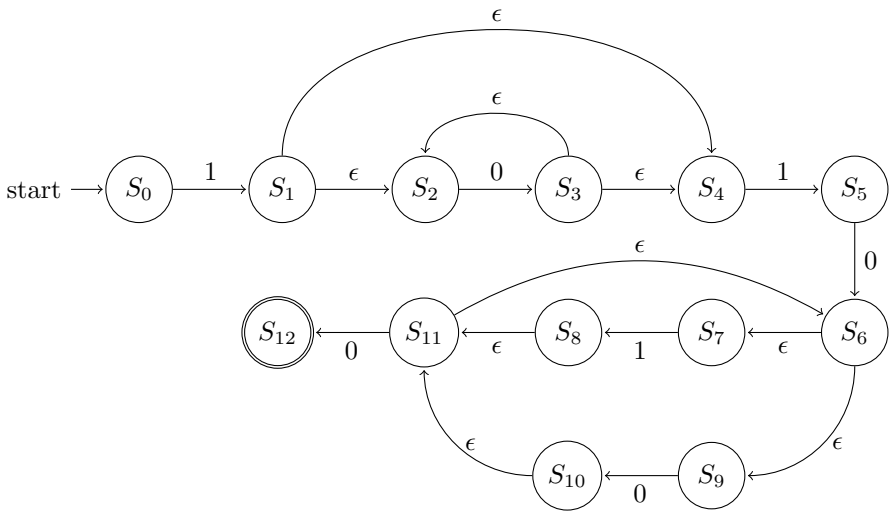
```
gcc --version
flex --version
bison --version
```

Problem 2

Interpreter: Kjører programkode uten å oversette sammen med input fra bruker for å returnere et svar. Kompilator: Oversetter programkoden til maskinkode som kan kjøres.

Problem 3

a)



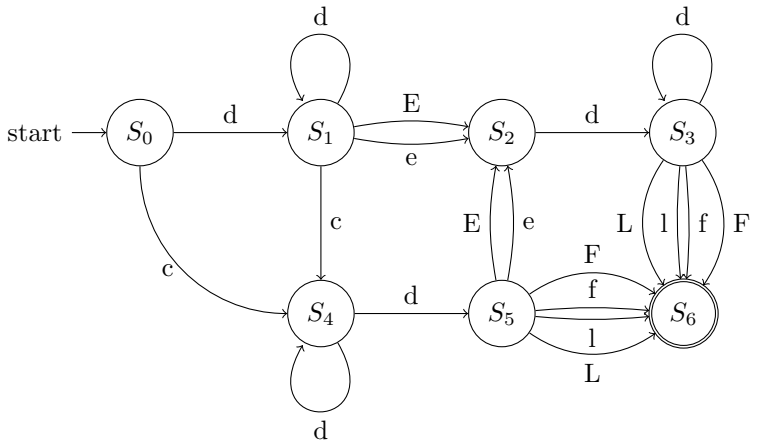
b) Alle ord som begynner på 1 etterfulgt av ingen eller flere 0 etterfulgt av 1 etterfulgt av 0 etterfulgt av en ubegrenset kombinasjon av 0 og 1 avsluttet med 0.

c)

state	1	0	ϵ
0	1	$\emptyset$	$\emptyset$
1	$\emptyset$	$\emptyset$	2, 4
2	$\emptyset$	3	$\emptyset$
3	$\emptyset$	$\emptyset$	2, 4
4	5	$\emptyset$	$\emptyset$
5	$\emptyset$	6	$\emptyset$
6	$\emptyset$	$\emptyset$	7, 9
7	8	$\emptyset$	$\emptyset$
8	$\emptyset$	$\emptyset$	11
9	$\emptyset$	10	$\emptyset$
10	$\emptyset$	$\emptyset$	11
11	$\emptyset$	12	$\emptyset$

Problem 4

d = siffer
c = komma



Code

```
#include <stdio.h>
#include <stdlib.h>
// A struct for tree nodes, with pointers to the parent and child nodes
// and a int to store the value at the node.
typedef struct Node{
    struct Node* left;
```

```
    struct Node* right;
    int value;
} Node;


void swap(int v[], int i, int j);
Node* create_blank_node();

// Returns a random number between 0 and n
// For simplicity, the random number generator is not seeded,
// hence the same random numbers will be generated on every execution
int get_random_number(int n){
    return rand() % n;
}


// Return a (dynamically allocated) array of length size,
// filled with random numbers between 0 and n
int* create_random_array(int size, int n){
        int* ptr = malloc(size * sizeof(int));
        for(int i = 0; i < size; i++)
                ptr[i] = get_random_number(n);
        return ptr;
}


// Should print the contents of array of lenght size
void print_array(int* array, int size){
        for(int i = 0; i < size; i++)
                printf("%d ", array[i]);
        printf("\n");

}


// Should sort the numbers in array in increasing order
void sort(int* array, int size){
        //Basically selectionsort
        for(int i = 0; i < size; i++){
                int index_of_min = i;
                int j = i;

                for(j = i; j < size; j++)
                        if(array[j] < array[index_of_min])
                                index_of_min = j;
                swap(array, i, index_of_min);
        }
}

//swap as taken from Kernighan & Ritchie. Call by reference because arrays are
```

```
void swap(int v[], int i, int j)
{
        int temp = v[i];
        v[i] = v[j];
        v[j] = temp;
}

// Inserts the node into the tree rooted at the node pointed to by root
void insert_node(Node** root, Node* node){
        Node* cur_root = *root;
        if(cur_root->value == -1)
        {
                *root = node;
                (*root)->right = create_blank_node();
                (*root)->left  = create_blank_node();
        }

        else if(node->value >= cur_root->value)
        {
                //if(cur_root->right == NULL)
                //      cur_root->right = create_blank_node();
                insert_node( &cur_root->right, node );
        }
        else
        {
                //if(cur_root->left == NULL)
                        //cur_root->left = create_blank_node();
                insert_node( &cur_root->left, node);
        }
}


// Searches for the number n in the tree rooted at root.
// Should return 1 if the number is present, and 0 if not.
int search(Node* root, int n){
        if(root->value == n)
                return 1;
        else if(root->value < n && root->right != NULL)
                return search(root->right, n);
        else if(root->value > n && root->left != NULL)
                return search(root->left, n);
        else return 0;
}


// Returns a dynamically allocated node, with all fields set to NULL/0
Node* create_blank_node(){
        Node* new_node = (Node*)malloc(sizeof(Node));
        new_node->left = NULL;
        new_node->right = NULL;
```

```
        new_node->value = -1;

        return new_node;
}


// Builds a tree of all the numbers in an array
Node* create_tree(int* array, int size){
        Node* root = create_blank_node();
        for(int i = 0; i < size; i++)
        {
                Node* temp = create_blank_node();
                temp->value = array[i];
                insert_node(&root, temp);
        }
        return root;
}

// Prints all the nodes of the tree.
void print_tree(Node* n, int offset){

        if(offset <= 0 && n->value != -1)
                printf("%d ", n->value);
        if(n->left != NULL)
                print_tree(n->left, offset--);
        if(n->right != NULL)
                print_tree(n->right, offset--);

}


// Computes x^2
double x_squared(double x){
    return x*x;
}


// Computes x^3
double x_cubed(double x){
    return x*x*x;
}

//http://en.wikipedia.org/wiki/Higher-order_function
// Computes the definite integral of the function using the rectangle method
double integrate(double (*function)(double), double start, double end, double s
        double d = (end - start) * stepsize;
        int steps = (int)(end-start) / stepsize;
        double sum = 0.0;

        for(int i = 0; i < steps; i++)
```

```
            sum += (*function)(i * d + start) *d;

        return sum;
}


int main(int argc, char** argv){
    // Creates an array with random values
    int* array = create_random_array(10, 10);

    // Prints the values of the array, e.g:
    // 3 6 7 5 3 5 6 2 9 1
    print_array(array, 10);

    // Sorts the array
    sort(array, 10);

    // Prints the sorted array, e.g:
    // 1 2 3 3 5 5 6 6 7 9
    print_array(array, 10);

    // Create another random array
    int* new_array = create_random_array(10,10);

    // Print the second array
    print_array(new_array, 10);

    // Create a tree with the values in the new array
    Node* root = create_tree(new_array, 10);

    // Print the tree
    print_tree(root, 0);
    printf("\n");
    // Search for the values 3 and 11 in the tree
    // and print the results
    int found_3 = search(root, 3);
    int found_11 = search(root, 11);
    printf("%d, %d\n", found_3, found_11);

    // Integrate x^2 and x^3 from 0 to 1.
    // Should be approx 1/3 and 1/4.
    printf("%f\n", integrate(&x_squared, 0, 1, 0.001));
    printf("%f\n", integrate(&x_cubed, 0, 1, 0.001));
}
```