```c
/*
Introduction
*/
#define MAX 255
#include <stdio.h>  /* standard from c */
#include <stdlib.h> /*constants like EXIT_SUCCESS or EXIT_FAILURE*/
#include <unistd.h> /* for arguments (getopt), R_OK */
#include <string.h> /* string copy and string length */
#include <time.h>   /* srand(time) */

#define DBG 1
#define DEBUG(fmt, ...)                                          \
    do                                                           \
    {                                                            \
        if (DBG)                                                 \
            fprintf(stderr, "\n%s:%d:%s(): " fmt "\n", __FILE__, \
                    __LINE__, __func__, __VA_ARGS__);            \
    } while (0)

enum
{
    OFF,
    ON
};

struct
{
    int l;
    int f;
} optflags = {OFF, OFF};

struct Liste_s
{
    char info;
    int value;
    struct Liste_s *next;
};
typedef struct Liste_s LISTE_T;

struct binaerer_baum
{
    char keyword[MAX];
    int maxchars;
    struct Liste_s *list;
    struct binaerer_baum *left;
    struct binaerer_baum *right;
};
typedef struct binaerer_baum BAUM_T;

LISTE_T *insert_Node(LISTE_T *aktuellenknoten, char zeichen);
```

```c
BAUM_T *create_newTreeNode(char key[], char zeichen, int s_optarg);
BAUM_T *search_Baum(BAUM_T *baum, char checkStr[]);
BAUM_T *einfuegen_Tree(BAUM_T *wurzel, char key[], int s_optarg);

void usage();
char randomChar(BAUM_T *head, char suchstr[]);

int main(int argc, char *argv[])
{
    int option;
    int count_zeichen = 0;
    int s_optarg = 1;
    int l_optarg = 0;
    int maxoutput = 0;
    FILE *fp = stdin;
    char z, c;
    char filename[MAX] = "";
    char suchstring[MAX] = "";
    char startstring[MAX] = "";

    // Startpunkt nur einmal vorne im Programm festlegen: Periode ist >= 2^^31
also ziemliich gross!
    srand((unsigned)time(NULL)); //start punkt festlegen

    while ((option = getopt(argc, argv, "hs:l:f:")) != -1)
    {

        switch (option)
        {
        case 'h':
            usage();
            break;
        case 's':
            s_optarg = atoi(optarg); // atoi = char* to int (ASCII to integer)
            break;
        case 'l':
            optflags.l = ON;
            l_optarg = atoi(optarg);
            maxoutput = l_optarg;
            break;
        case 'f':
            optflags.f = ON;
            strncpy(filename, optarg, MAX);
            break;
        case '?':
            usage();
            break;
        default:
            usage();
            break;
```

```
        }
    }
    if (optind < argc && optflags.f == OFF)
    {
        strncpy(filename, argv[optind], MAX);
    }
    if (strlen(filename) != 0)
    {
        if (access(filename, R_OK) == EXIT_SUCCESS) // check ob das file
readable ist und beim returnwert kein fehler auftritt (EXIT_SUCCESS = 0)
        {
            fp = fopen(filename, "r");
        }
        else
        {
            printf("Error Lesbarkeit\n");
            exit(EXIT_FAILURE);
        }
    }
    if (s_optarg >= 1)
    {
        BAUM_T *head = NULL;
        // Eingabephase
        while ((z = fgetc(fp)) && !feof(fp))
        {
            count_zeichen++;
            if (strlen(suchstring) == s_optarg + 1) // Länge + Folgezeichen
erreicht ?
            {
                head = einfuegen_Tree(head, suchstring, s_optarg); //
suchstring= suchstring + Folgezeichen !!
                for (int i = 0; i <= s_optarg + 1; i++)
                {
                    suchstring[i] = suchstring[i + 1];
                }
                suchstring[s_optarg] = z;
                suchstring[s_optarg + 1] = '\0';
            }
            else
            { // auffüllen auf Suchstringlänge
                strncat(suchstring, &z, 1);
            }
        } // Ende Eingabephase

        maxoutput = (optflags.l == ON) ? l_optarg : count_zeichen;

        strncpy(startstring, head->keyword, MAX);
        printf("\n----------------------------------------\n");
        printf("<%s>", startstring);
        maxoutput -= strlen(startstring); //erste ausgabe
```

```
        while (maxoutput != 0)
        {
            c = randomChar(head, startstring);
            printf("%c", c);
            maxoutput--;
            for (int i = 0; i < s_optarg; i++) // nur string nach links
schieben
            {
                startstring[i] = startstring[i + 1];
            }
            startstring[s_optarg - 1] = c;
            startstring[s_optarg] = '\0';
        }
        printf("\n----------------------------------------\n");
        free(head);
        fflush(stdout);
        return 0;
    }
}
void usage()
{
    printf("Usage : ./myprog [-h] [-s <n>] [-l <m>] [-f <inputfile>]\n");
    exit(EXIT_FAILURE);
}

BAUM_T *create_NewTreeNode(char key[], char zeichen, int s_optarg) //
CamelCase Schreibweise beachten
{
    BAUM_T *newNode;
    newNode = malloc(sizeof(BAUM_T)); // newNode speicherreservieren in größe
von BAUM_T
    if (newNode == NULL)              // falls kein speicher für newNode
reserviert werden kann
    {
        // Programm kann hier beendet werden
        printf("ERROR: Kein Speicher mehr!");
        exit(EXIT_FAILURE);
    }
    else
    {                                           // ansonsten definiere
newNode
        newNode->list = malloc(sizeof(LISTE_T));  // newNode schiefgeht machte
es keinen Sinn, deshalb hier
        strncpy(newNode->keyword, key, s_optarg); // information kopieren
        newNode->keyword[s_optarg] = '\0';        // Genau!
        newNode->maxchars = 1;
        newNode->right = NULL;
        newNode->left = NULL;          // zeigt auf ende der liste Ähhhh?????
        newNode->list->info = zeichen; //erstes zeichen nach suchstring
erstellt listenanfang
```

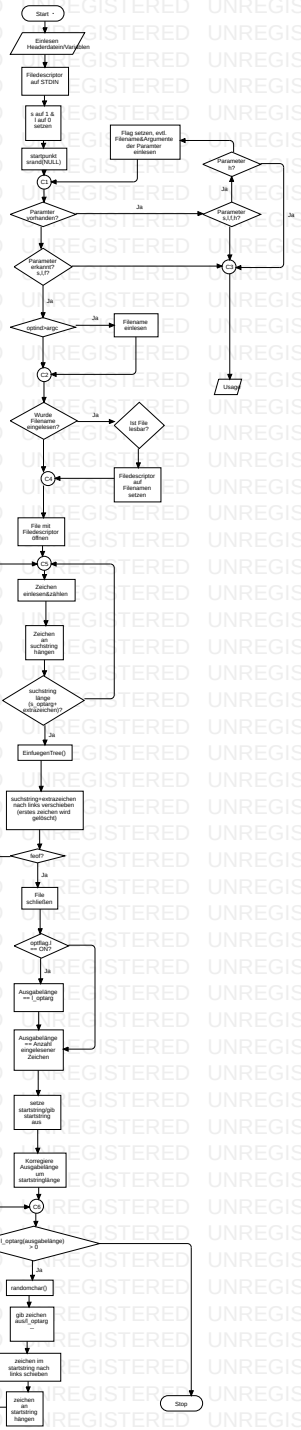
```
        newNode->list->value = 1;
        newNode->list->next = NULL;
    }
    return newNode;
}

// Namenskonventinen !! Methoden beginnen mit kleinem Buchstaben!
BAUM_T *einfuegen_Tree(BAUM_T *node, char key[], int s_optarg) // string ist
suchstring + Folgezeichen
{                                                              //übergibt
suchstring in dem baum
    char suchstring[s_optarg + 1];
    strncpy(suchstring, key, s_optarg);
    suchstring[s_optarg] = '\0';

    if (node == NULL)
    { // falls noch kein element am anfang ist
        node = create_NewTreeNode(suchstring, key[s_optarg], s_optarg);
    }
    else
    {
        int vergleichswert = strncmp(suchstring, node->keyword, s_optarg);
//vergleicht zeichenketten mit einander strcmp(cs, ct)
        if (vergleichswert > 0)
        { //>0 cs>ct
            /*folge knoten größer als vorheriger,rechter zeiger vom vorherigen
knoten war auf null somit wird im nächsten durch gang ein neuer knoten
erstellt(createnewTreeNode)*/
            node->right = einfuegen_Tree(node->right, key, s_optarg);
        }
        if (vergleichswert < 0)
        { //<0 cs<ct
            /*folge knoten kleiner als vorheriger,linker zeiger vom vorherigen
knoten war auf null somit wird im nächsten durch gang ein neuer knoten
erstellt(createnewTreeNode)*/
            node->left = einfuegen_Tree(node->left, key, s_optarg);
        }
        if (vergleichswert == 0)
        { //=0 cs==ct
            /*wenn der string aus dem übergebenen baumknoten einem bereits
bestehenden baumknoten gleich ist,
            soll das zeichen in der liste, die mit dem aktuellen baumknoten
verbunden ist, erst einmal gesucht werden und dann die value des zeichens
erhöhen
            oder falls das zeichen noch nicht in der liste ist hinzugefügt
werden*/
            node->maxchars += 1;
            insert_Node(node->list, key[s_optarg]);
        }
    }
```

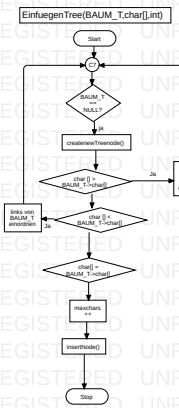
```
    return node;
}

LISTE_T *insert_Node(LISTE_T *aktuellenknoten, char zeichen) // im ersten fall
vorgänger_knoten = root
{
    LISTE_T *neuer_knoten;
    if (aktuellenknoten->info == zeichen)
    {                                   // gesuchtes zeichen gefunden
        aktuellenknoten->value += 1; //+= 1//erhöhte wertigkeit des zeichens
um 1
        return aktuellenknoten;
    }
    else
    { //
        if (aktuellenknoten->next == NULL)
        { // ende gefunden -> neuen Knoten einfügen
            neuer_knoten = malloc(sizeof(LISTE_T));
            if (neuer_knoten == NULL)
            { // ist speicher für den neuen knoten reserviert worden?
                // Programm kann hier beendet werden
                printf("ERROR: Kein Speicher mehr!");
                exit(EXIT_FAILURE);
            }
            neuer_knoten->info = zeichen;
            neuer_knoten->value = 1;
            neuer_knoten->next = NULL;
            aktuellenknoten->next = neuer_knoten; // vorgängerknoten zeigt auf
nachfolgeknoten
        }
        else
        {
            insert_Node(aktuellenknoten->next, zeichen); // wiederholung des
prozesses
        }
        return aktuellenknoten;
    }
}

// Eingabe Baumknoten
// Return:  Zufalls char zurück
char randomChar(BAUM_T *head, char suchStr[]) // CamelCase !!!
{
    char retVal = ' ';
    BAUM_T *myHead;
    LISTE_T *myListNode;

    myHead = search_Baum(head, suchStr); // setzt head neu auf gefundenen
Knoten !!!
```

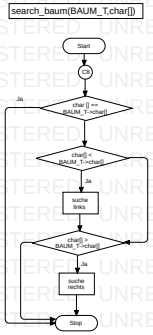
```
    myListNode = myHead->list;           // current Listnode. Verändert sich
während der Suche

    int intervall = myListNode->value; // erste obere Intervallgrenze
    int randomnumber;

    int a = 1;
    int e = myHead->maxchars;
    double range = e - a + 1.0;
    randomnumber = a + (int)(range * rand() / (RAND_MAX + 1.0));

    while (intervall < randomnumber)
    {                                  // head->list==NULL sollte nie
passieren ->break;
        myListNode = myListNode->next; // zuletzt == NULL, aber dann greift
auch intervall test
        if (myListNode == NULL)
        {
            printf(".\n\n");
            printf("Sollte (fast) niemals erreicht werden. (Nur beim
allerletzten Term ist das möglich)");
            exit(EXIT_SUCCESS);
        }
        intervall += myListNode->value; //intervall größe dem ersten wert in
der liste anpassen
    }
    // intervall >= randomnumber
    retVal = myListNode->info;
    return retVal;
}

// Besser hier gleich den Zufallschar ermitteln und den char zurückgeben)
BAUM_T *search_Baum(BAUM_T *baum, char checkStr[])
{
    // Besser mit nur einem Ausgang/return !!
    BAUM_T *retNode = NULL;
    int vergleich;
    vergleich = strcmp(checkStr, baum->keyword);
    if (baum == NULL)
    {
        printf("DEBUG: Warum wird hier der Knoten NULL ??");
    }
    if (vergleich == 0)
    { // Baum sollte niemals NULL werden
        retNode = baum;
    }
    else if (vergleich < 0)
    {
        retNode = search_Baum(baum->left, checkStr);
    }
```

```
    else if (vergleich > 0)
    {
        retNode = search_Baum(baum->right, checkStr);
    }
    return retNode;
}
```

Flowchart/Flowchart Diagram







Programmierübung Aufgabe 2

Es sollte ein Programm shakespeare.c geschrieben werden, dass:

1. Parameter einliest,
2. eine Text-Datei/Pfad durch Parameter einliest und diese Datei öffnet,
3. deren Inhalt einliest,
4. Operationen ausführt,
5. das Ergebnis in die Standard-Ausgabe(`stdout`) schreibt.

Parameter sollten aus einem Buchstaben mit einem vorangestellten – gekennzeichnet werden.

Diese bestimmen welche der Operationen ausgeführt werden soll:

1. Bei dem Operator -h soll eine `usage ()` –Meldung in die Standard-Ausgabe geschrieben werden.
2. Bei dem Operator -f soll als Argument eine Datei eingelesen werden.
3. Bei dem Operator -s soll als Argument eine Zahl eingelesen werden, die die Länge eines Suchstrings angibt und standardmäßig auf 1 initialisiert wird.
4. Bei dem Operator -l soll als Argument eine Zahl eingelesen werden, die die Länge der auszugebenen Zeichen angibt und standardmäßig in Höhe der eingelesenen Zeichen initialisiert wird.

Die Konzipierung des Programmes soll später das Hinzufügen von weiteren Parametern leicht gestalten. Außerdem soll das Programm als Filter arbeiten können, d.h falls keine Datei oder Pfadname als Argument angegeben wurde, soll das Programm seine Eingabe aus der Standard-Eingabe (`stdin`) einlesen.

Operationen der Anwendung

Das Programm soll mithilfe von Parametern verschiedene Operationen ausführen.

Parameter werden mit der `geopt ()` –Funktion, in `unistd.h` zu finden, als Option mit oder ohne Argumente an die Funktion übergeben. Die `geopt ()` –Funktion in eine `while`-Schleife einzubinden, ermöglicht dem Programm mehrere Parameter nacheinander einzulesen. Da in diesem Fall jeder Parameter eine andere Funktion in dem Programm haben soll, kann man die Parameter in Fälle aufteilen, um je nachdem welcher Parameter eingelesen wurde ein anderes Signal/Flag übergeben werden kann.

Der -h Parameter dient als Hilfestellung, beim Einlesen dieses Parameters soll eine `usage ()` – Meldung in die Standard-Ausgabe geschrieben werden, um dem Anwender die Vorgaben für das Programm mitzuteilen.

Der -f Parameter dient dazu eine Textdatei einzulesen. Der Parameter ist nicht zwingend notwendig, um eine Textdatei einlesen zu können.

Der -l Parameter dient dazu die Ausgabelänge des Zufallstext zu bestimmen. Standardmäßig wird die Ausgabelänge der eingelesenen Zeichen angepasst, falls kein Operator eingelesen wurde. Falls ein -l Operator eingelesen wurde soll das dazugehörige Argument als Länge der Ausgabe dienen.

Der -s Parameter dient dazu die Länge der Suchstrings anzugeben. Standardmäßig wird der Wert der Länge auf 1 initialisiert. Falls ein -s Operator mit einem Argument eingelesen wurde, gibt das Argument die Länge der Suchstrings an.

Falls eine Textdatei mit Parametern eingelesen wurde, muss dieses auf dessen Zugriffsrechte überprüft werden. Den Namen der Textdatei wird mit `strncpy ()`, aus der `string.h`, entweder durch das Argument des -f Parameters oder das zulässt eingelesenes Argument in die Variable filename kopiert. Dazu wird überprüft, ob der Name der Textdatei in die Variable gespeichert wurde und mit der `access ()` -Funktion und dem Makro `R_OK` wird das File auf Lesbarkeit geprüft. Sollte der Rückgabewert = 0 sein, so wird das File geöffnet andernfalls wird eine Fehlermeldung ausgeben. Da kein Pfad des Textfiles angegeben wurde muss das File im gleichen Ordner wie das C-Programm sein. Wenn das File erfolgreich geöffnet werden konnte, wird der Text zeichenweise eingelesen und in der Variablen Suchstring kopiert. Wenn die Variable Suchstring die Anzahl der Zeichen, die Anzahl wird mit dem übergebenen Argument des -s Parameters festgelegt (falls dieser nicht angeben wurde, ist die Anzahl der Zeichen 1), + ein Folgezeichen enthält wird dieser Suchstring in einem binären Suchbaum eingeordnet mit der `EinfuegenTree ()` -Funktion und das Folgezeichen in eine einfach verkettete Liste, welche an dem Baum-Knoten hängt, eingeordnet. Ein binärer Suchbaum ist eine Datenstruktur, die immer mindestens eine Daten-Variable und zwei Äste, einen linken und einen rechten Ast, enthält. Datenstrukturen sind dazu da Daten zu speichern, löschen, sortieren, anzuhängen oder zu suchen. Von der Wurzel des Baumes aus werden alle Knoten kleiner als/gleich die Wurzel links eingeordnet und größer als die Wurzel rechts eingeordnet. Falls keine Duplikate im Baum erwünscht sind, muss beim Einordnen im Baum ein Fall vorkommen der dieses berücksichtigt. In unserem Fall sind Duplikate nicht erwünscht. Falls ein Suchstring als Duplikat auftritt, wird das Folgezeichen in eine einfach verkettete Liste eingeordnet. Dabei wird überprüft, ob das Zeichen bereits in der Liste vorkommt oder nicht, falls das Zeichen noch nicht vorhanden ist, wird es an die

Liste angehängt und wenn das Zeichen bereits zu finden ist, wird die Anzahl des vorgekommenen Zeichens um 1 erhöht. Sobald der Knoten im Baum und das Folgezeichen in der Liste eingeordnet wurden, wird der Suchstring um ein Zeichen nach links verschoben, sodass das erste Zeichen wegfällt. An den daraus entstandenen Suchstring wird das nächste Zeichen im Text angehangen. Dieser Prozess läuft so lange bis das Ende des Files (`feof`) erreicht ist. Die Ausgabelänge wird, falls ein -l Operator eingelesen wurde mit dem dazugehörigen Argument angegeben, andernfalls mit der Anzahl der eingelesenen Zeichen aus dem File. Um den Zufallstext zu erstellen, wird als ersten String der String, der sich im Root-Knoten befindet, ausgegeben. Der Startstring und den Baum werden in die `randomChar ()` -Funktion übergeben. Der übergebene String wird im Baum gesucht und der gesuchte Knoten zurückgegeben. Die `rand ()` -Funktion generiert eine Zufallszahl. Die `srand ()` -Funktion setzt den Startpunkt der `rand ()` -Funktion fest, ohne diese Funktion wird die zurückgegebene Zufallszahl immer dieselbe bleiben. Die value -Variable in den Listen-Knoten gibt mir die Anzahl der dort gespeicherten Buchstaben an. Diese Anzahl kann man als Intervall betrachten. Sollte meine Zufallszahl größer als mein Intervall sein, wird mein Intervall um die value -Variablen erhöht und es wird zum nächsten Listen-Knoten gegangen. Wenn mein Intervall größer oder gleich als meine Zufallszahl ist, wird der Buchstabe zurückgegeben. Dieser Buchstabe wird ausgegeben und die Ausgabelänge angepasst. Nach jeder Ausgabe eines Zufallsbuchstaben wird dieser an den vorherigen String angehangen und der String um ein Zeichen nach links geschoben und der Vorgang wird wiederholt, bis die Ausgabelänge 0 erreicht hat. Da mit den Datenstrukturen Speicher belegt wurde, wird der Speicher am Ende des Programms mit der `free ()` –Funktion freigegeben.