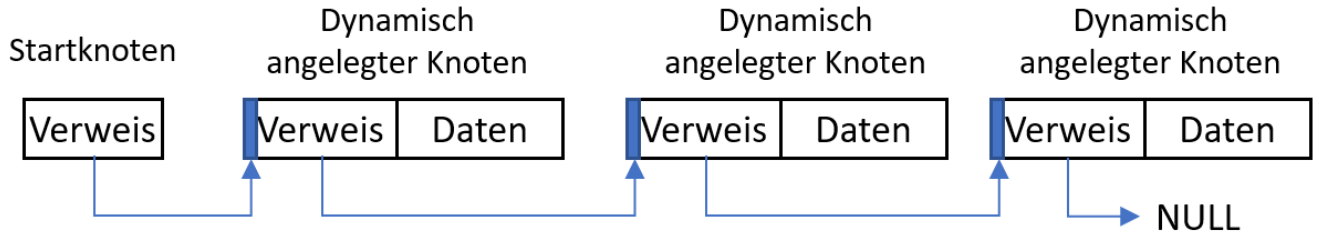
Dyn. Speicher (Verkette Liste)

Dienstag, 12. April 2022 10:29

Einfach verkette Listen

Wikipedia: Eine verkettete Liste ist eine dynamische Datenstruktur, in der Datenelemente geordnet gespeichert sind. Bei ihrer Erstellung braucht die maximale Anzahl der Elemente nicht festgelegt zu werden, und die Anzahl darf während der Laufzeit beliebig variieren.

Wikipedia: Die technische Realisierung erfolgt meistens durch einzelne Knoten, die aus den Nettodaten selbst und einen Verweis auf den Nachfolgeknoten besteht. Im letzten Knoten ist als Nachfolgeknoten der sogenannte Null-Zeiger angegeben. Der Beginn der Liste wird durch ein Startelement beschrieben welches nur ein Verweis (Pointer) auf den1. Knoten enthält.



Der Verweis auf den Nachfolgeknoten entspricht in C einer Zeigervariablen/Pointer, wobei der Inhalt des Zeigervariablen auf die Startadresse des nächsten Knoten zeigt oder einen NULL-Zeiger enthält. In der Programmiersprache C könnte ein Knoten der verketteten Liste wie folgt dargestellt werden:

```
typedef struct vl {
        struct vl *next;
        struct student {
            char vorname[100];
            char nachname[100];
            enum {deutsch,england,spanien,sonstiges}nationalität;
            short semester;
        } student;
} vl_t;
```

Das Strukturelement next dient zur Speicherung der Adresse des Folgeknotens. Ist kein Folgeknoten enthalten, so wird dies mit NULL kenntlich gemacht. Da der Startknoten nur die Adresse des ersten Knotens und keine Daten speichern soll, empfiehlt sich folgende Variable für den Startknoten. Auch hier zeigt NULL an, dass kein Folgeknoten enthalten ist, die verkette Liste also leer ist.

```
vl_t *head=NULL;
```

Wenn die Anzahl der zu speichernden Daten Variabel ist, bietet sich folgende Struktur an:

```
typedef struct vl {
        struct vl *next;
        char       data[];
} vl_t;
vl_t *head;
```

Das incomplete Array (hier data[]) am Ende der Struktur ermöglicht es, den Speicherbedarf der Struktur den tatsächlichen Bedürfnissen anzupassen und nur so viel Speicher zu reservieren, wie zur Speicherung der Daten notwendig sind. Im konkreten Anwendungsfall sollen Strings gespeichert werden, deren Größe sich erst zur Laufzeit ergibt. Die Speicherplatzreservierung sieht dann wie folgt aus:

```
vl_t *new=(vl_t*)malloc(sizeof(vl_t)+strlen(str)+1);
```

Aufgabe

a) Realisieren sie die Funktion:
   vl_addfront() //Zum Einhängen eines Strings an den Anfang der verketten Liste
   vl_addback() //Zum Einhängen eines Strings an das Ende der verketten Liste
   vl_getback(); //Zum Entnehmen (inkl. Löschen) des letzten Elementes der Liste

vl_close(); //Als Dekonstruktor zur Freigabe sämtlicher Elemente der Liste

b) Erzeugen sie eine verkette Liste mit 5 Elemente (wobei die Elemente abwechselnd vorne und hinten anzuhängen sind). Erstellen sie hiervon mit debug() eine 'Speicherabbildung' und fügen die Ausgabe in den SourceCode ein. Der Ausgabe entspricht einem Speicherdump der einzelnen Knoten. Erklären sie den Inhalt des Bereiches ab 'Speicherdump:'

Zum Testen der Funktion benutzen sie den 'integrierten' Kommandozeileninterpreter, welcher nachfolgende Befehle zur Verfügung stellt :

```
setf <value> - Fügt <value> am Anfang der Liste ein
setb <value> - Fügt <value> am Ende der Liste ein
getb         - Gibt das letzte Element zurück und
               entfernt dieses aus der Liste
debug        - Gibt die gesamte Lite aus
dump         - Gibt ein Speicherabbild der Liste aus
help         - Führt alle Operationen auf.
quit         - Beendet das Programm.
```

```c
#include <stdio.h>     //fuer printf() size_t stderr
#include <stdlib.h>    //fuer malloc() free()
#include <string.h>    //fuer strcpy() memcpy()
//Compilerschalter: -fsanitize=address -Wall -m32
//-m32 zur Umschaltung auf eine 32-Bit Architektur
#if 0
//Aufgabe b)
//Kopie der Standarausgabe von debug
#endif
typedef struct vl {
        struct vl *next;
        char       data[];
} vl_t;
vl_t *head;
void vl_init(void)
{
    head=NULL;
}
void vl_close(void)
{
    //Liste durchgehen und allen Speicher löschen
}
static void vl_debug(void)
{
    printf("------------\n");
    printf("head=%p\n",head);
    if(head!=NULL) {
        printf("Inhalt der Struktur:\n");
        for(vl_t *ptr=head;ptr!=NULL;ptr=ptr->next)
            printf("%p:{.next=%p: .data='%s'}\n",ptr,ptr->next,ptr->
data);
        printf("Speicherdump:\n");
        for(vl_t *ptr=head;ptr!=NULL;ptr=ptr->next) {
            printf("%p:",ptr);
            unsigned char *start=(unsigned char *)ptr;
            unsigned char *end  =start+sizeof(char *)+strlen(ptr->data)+
1;
            do {
                printf("%02x ",*start++);
            } while(start<end);
            printf("\n");
        }
    }
    //Größe des benötigten Speichers berechnen
}
int vl_addfront(const char *value)
{
```

```
    vl_t *new;
    new=malloc(sizeof(vl_t)+strlen(value+1) );
    if(new==NULL)
        return -1;
    return 0;
}
int vl_addback(const char *value)
{
    return 0;
}
char *vl_getback(void)
{
    return NULL;
}

const char help[]= "\n"
  "setf <value> - Fügt <value> am Anfang der Liste ein\n"
  "setb <value> - Fügt <value> am Ende der Liste ein\n"
  "getb         - Gibt das letzte Element zurück und \n"
  "               entfernt dieses aus der Liste\n"
  "\e[1mdebug\e[0m        - Gibt die gesamte Lite aus\n"
  "help         - Führt alle Operationen auf.\n"
  "\e[1mquit\e[0m         - Beendet das Programm.\n";
int main(int argc, char const *argv[]) {
    //Konstruktoren
    vl_init();
    printf("File:"__FILE__ " erstellt am "__DATE__" um "__TIME__ "
gestartet\n");
    printf(help);
    while(1) {
        char input[100];
        if(fgets(input, 100, stdin)==NULL)
            break;
        char *arg1 = strtok(input, " \n\r");
        char *arg2 = strtok(NULL, " \n\r");

        if(arg1==NULL) {
            continue;
        }
        else if((strcmp(arg1,"setf")==0) && (arg2!=NULL)) {
            if(vl_addfront(arg2) == -1)
                printf("Fehler: Element konnte nicht eingefügt werden");
        }
        else if((strcmp(arg1,"setb")==0) && (arg2!=NULL)) {
            if(vl_addback(arg2) == -1)
                printf("Fehler: Element konnte nicht eingefügt werden");
        }
        else if(strcmp(arg1,"getb")==0)  {
            char *ret=vl_getback();
            if(ret!=NULL) {
                printf("'%s'",ret);
                free(ret);
            }
        }
        else if(strcmp(arg1,"debug")==0) {
            vl_debug();
        }
        else if(strcmp(arg1,"help")==0) {
            printf(help);
        }
        else if(strcmp(arg1,"quit")==0) {
            break;
        }
        else {
            printf("unbekanntes Kommando\n");
        }
```

```
    }
    vl_close();
    return 0;
}
```