U8 8. Übung

- Dynamische Speicherverwaltung
- Fehler finden mit valgrind
- Aufgabe 8: sortdir

U8-1 Dynamische Speicherverwaltung

- Rückblick: GDI 9.1

```
Element newElement = new Element();
newElement.next = head;
head = newElement;
```

  - In Java: Neues Listenelement wird mit Hilfe von `new` instanziiert
    - Reservieren eines Speicherbereichs für das Objekt
    - Initialisieren des Objektes durch Ausführen des Konstruktors

- In C: Anlegen eines Listenelements mittels `malloc()`

```
struct listelement *newElement;
newElement = malloc( sizeof( struct listelement ) );
if( newElement == NULL ) {
    // Fehlerbehandlung
}
```

  - Zurückgegebener Speicherbereich ist uninitialisiert
  - Initialisierung muss per Hand erfolgen

U8-1 Dynamische Speicherverwaltung

- Explizite Initialisierung mit `void *memset(void *s, int c, size_t n)`

```
memset(newElement, 0, sizeof( struct listelement ) );
```

- Mit 0 vorinitialisierter Speicher kann mit `calloc()` angefordert werden:
  `void *calloc(size_t nelem, size_t elsize)`

```
struct listelement *newElement;
newElement = calloc( 1, sizeof( struct listelement ) );
if( newElement == NULL ) { ... }
```

- Im Gegensatz zu Java gibt es in C keinen Garbage-Collection-Mechanismus
  - Speicherbereich muss von Hand freigegeben werden

```
free(newElement);
```

  - Nur Speicher, der mit einer der alloc-Funktionen zuvor angefordert wurde, darf mit `free()` freigegeben werden!

- Der Zugriff auf freigegebene Speicherbereiche hat undefiniertes Verhalten.

U8-2 Fehler finden mit valgrind

- Baukasten von Debugging- und Profiling-Werkzeugen (ausführbarer Code wird durch synthetische CPU auf Softwareebene interpretiert ➔ Ausführung erheblich langsamer!)
  - ◆ Memcheck: erkennt Speicherzugriff-Probleme
    - ➤ Nutzung von nicht-initialisiertem Speicher
    - ➤ Zugriff auf freigegebenen Speicher
    - ➤ Zugriff über das Ende von allokierten Speicherbereichen
    - ➤ Zugriff auf ungültige Stack-Bereiche
- Aufrufbeispiel: **`valgrind ./sortdir`**
- Programm muss mit der Compileroption **`-g`** übersetzt werden.

```
gcc [...] -g -std=c99 -D_BSD_SOURCE -o sortdir sortdir.c
```

  - ◆ Sogenannte Debug-Symbole werden vom Compiler in die ausführbare Datei eingebaut. Diese wertet valgrind aus, um z.B. die Zeilennummer eines Funktionsaufrufs zu erhalten.

1 Fehler finden mit valgrind

- Zugriffe auf nicht allokierten Speicher finden

```
==11711== Invalid read of size 4
==11711==    at 0x804841B: main (sortdir.c:19)
==11711==  Address 0x0 is not stack'd, malloc'd or (recently) free'd
==11711==
==11711==
==11711== Process terminating with default action of signal 11 (SIGSEGV)
==11711==  Access not within mapped region at address 0x0
==11711==    at 0x804841B: main (sortdir.c:19)
```

  - In Zeile 19 wurden 4 Byte von Adresse 0x0 gelesen
    - Auch schreibende Zugriffe können entdeckt werden
  - Darauf hin wurde der Prozess beendet

1 Fehler finden mit valgrind

- Auffinden von nicht freigegebenem Speicher

```
==5344== HEAP SUMMARY:
==5344==     in use at exit: 16 bytes in 2 blocks
==5344==   total heap usage: 4 allocs, 2 frees, 32 bytes allocated
```

  - Bei Programmende sind noch 2 Speicherbereiche (Blöcke) belegt
  - Während der Programmausführung wurde viermal `malloc()` und nur zweimal `free()` aufgerufen
  - Mit Hilfe der Option `--leak-check=full --show-reachable=yes` wird angezeigt, wo der Speicher angelegt wurde, der nicht freigegeben wurde.

```
==5865== 8 bytes in 1 blocks are still reachable in loss record 1 of 2
==5865==    at 0x48DAF50: malloc (vg_replace_malloc.c:236)
==5865==    by 0x80484B9: insertDir (sortdir.c:28)
==5865==    by 0x8048599: main (sortdir.c:61)
```

  - In der Zeile 28 wurde der Speicher angelegt
  - Nun im Quellcode Stellen identifizieren, an denen `free()`-Aufrufe fehlen

1 Fehler finden mit valgrind

- Auffinden uninitialisierten Speichers

```
==18108== Conditional jump or move depends on uninitialised value(s)
==18108==    at 0x80484AC: insertDir (sortdir.c:19)
==18108==    by 0x8048566: main (sortdir.c:59)
```

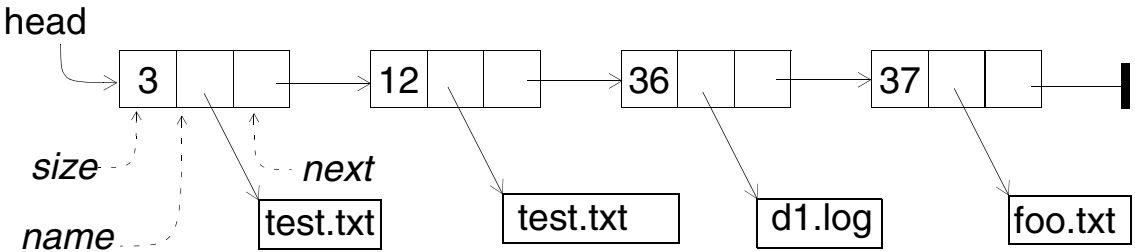
  - In der Funktion `insertElement()` in Zeile 19 wird auf uninitialisierten Speicher zugegriffen.
  - Mit Hilfe der Option `--track-origins=yes` wird angezeigt, wo der uninitialisierte Speicher angelegt wurde.

```
==18741== Conditional jump or move depends on uninitialised value(s)
==18741==    at 0x80484AC: insertDir (sortdir.c:19)
==18741==    by 0x8048566: main (sortdir.c:59)
==18741==  Uninitialised value was created by a heap allocation
==18741==    at 0x48DAF50: malloc (vg_replace_malloc.c:236)
==18741==    by 0x80484B9: insertDir (sortdir.c:28)
==18741==    by 0x8048549: main (sortdir.c:58)
```

U8-3 Aufgabe 8: sortdir

1 Einfach verkettete FIFO-Liste

- Zielsetzungen
  - ◆ Dynamische Speicherverwaltung und Umgang mit Zeigern



- Strukturdefinition:

```
struct dirList {
        struct dirList * next;
        char * name;
        off_t size;
};
```