Szöveges kalandjáték értelmező

Készítette Doxygen 1.9.5

1. fejezet

Adatszerkezet-mutató

1.1. Adatszerkezetek

Az összes adatszerkezet listája rövid leírásokkal:

2. fejezet

Fájlmutató

2.1. Fájllista

Az összes fájl listája rövid leírásokkal:

3. fejezet

Adatszerkezetek dokumentációja

3.1. DebugmallocData struktúrareferencia

`#include <debugmalloc.h>`

Adatmezők

- char logfile [256]
- long max_block_size
- long alloc_count
- long long alloc_bytes
- long all_alloc_count
- long long all_alloc_bytes
- DebugmallocEntry head [debugmalloc_tablesize]
- DebugmallocEntry tail [debugmalloc_tablesize]

3.1.1. Adatmezők dokumentációja

3.1.1.1. all_alloc_bytes

`long long all_alloc_bytes`

3.1.1.2. all_alloc_count

`long all_alloc_count`

3.1.1.3. alloc_bytes

```
long long alloc_bytes
```

3.1.1.4. alloc_count

```
long alloc_count
```

3.1.1.5. head

```
DebugmallocEntry head[debugmalloc_tablesize]
```

3.1.1.6. logfile

```
char logfile[256]
```

3.1.1.7. max_block_size

```
long max_block_size
```

3.1.1.8. tail

```
DebugmallocEntry tail[debugmalloc_tablesize]
```

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- debugmalloc.h

3.2. DebugmallocEntry struktúrareferencia

```
#include <debugmalloc.h>
```

Adatmezők

- void * real_mem
- void * user_mem
- size_t size
- char file [64]
- unsigned line
- char func [32]
- char expr [128]
- struct DebugmallocEntry * prev
- struct DebugmallocEntry * next

3.2.1. Adatmezők dokumentációja

3.2.1.1. expr

```
char expr[128]
```

3.2.1.2. file

```
char file[64]
```

3.2.1.3. func

```
char func[32]
```

3.2.1.4. line

```
unsigned line
```

3.2.1.5. next

```
struct DebugmallocEntry * next
```

3.2.1.6. prev

```
struct DebugmallocEntry* prev
```

3.2.1.7. real_mem

```
void* real_mem
```

3.2.1.8. size

```
size_t size
```

3.2.1.9. user_mem

```
void* user_mem
```

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- debugmalloc.h

3.3. List struktúrareferencia

Int lista.

```
#include <essentials.h>
```

Adatmezők

- Node * startNode
- int len

3.3.1. Részletes leírás

Int lista.

3.3.2. Adatmezők dokumentációja

3.3.2.1. len

```
int len
```

3.3.2.2. startNode

```
Node* startNode
```

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- essentials.h

3.4. Menultem struktúrareferencia

Egy menü element definiál.

Adatmezők

- int x
- int y
- char * text

3.4.1. Részletes leírás

Egy menü element definiál.

3.4.2. Adatmezők dokumentációja

3.4.2.1. text

```
char* text
```

3.4.2.2. x

```
int x
```

3.4.2.3. y

```
int y
```

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- graphics.c

3.5. Node struktúrareferencia

Egy lista elem.

```
#include <essentials.h>
```

Adatmezők

- struct Node * nextNode
- int value

3.5.1. Részletes leírás

Egy lista elem.

3.5.2. Adatmezők dokumentációja

3.5.2.1. nextNode

```
struct Node* nextNode
```

3.5.2.2. value

```
int value
```

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- essentials.h

3.6. Object struktúrareferencia

Objectet definiáló struct.

`#include <gameSpace.h>`

Adatmezők

- int id
- int state
- char * name
- char * lookAtText
- bool collectible
- StringList * modifyEvent
- StringList * modify
- StringList * modifyString

3.6.1. Részletes leírás

Objectet definiáló struct.

3.6.2. Adatmezők dokumentációja

3.6.2.1. collectible

```
bool collectible
```

3.6.2.2. id

```
int id
```

3.6.2.3. lookAtText

```
char* lookAtText
```

3.6.2.4. modify

```
StringList* modify
```

3.6.2.5. modifyEvent

```
StringList* modifyEvent
```

3.6.2.6. modifyString

```
StringList* modifyString
```

3.6.2.7. name

```
char* name
```

3.6.2.8. state

```
int state
```

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- gameSpace.h

3.7. Room struktúrareferencia

Szobát definiáló struct.

```
#include <gameSpace.h>
```

Adatmezők

- int id
- int state
- char * name
- char * lookAroundText
- char * entryText
- List * connectedItems
- List * connectedRooms

3.7.1. Részletes leírás

Szobát definiáló struct.

3.7.2. Adatmezők dokumentációja

3.7.2.1. connectedItems

```
List* connectedItems
```

3.7.2.2. connectedRooms

```
List* connectedRooms
```

3.7.2.3. entryText

```
char* entryText
```

3.7.2.4. id

```
int id
```

3.7.2.5. lookAroundText

```
char* lookAroundText
```

3.7.2.6. name

```
char* name
```

3.7.2.7. state

```
int state
```

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- gameSpace.h

3.8. State struktúrareferencia

Állapotot definiáló struct.

```
#include <gameSpace.h>
```

Adatmezők

- bool room
- int id
- int state

3.8.1. Részletes leírás

Állapotot definiáló struct.

3.8.2. Adatmezők dokumentációja

3.8.2.1. id

```
int id
```

3.8.2.2. room

```
bool room
```

3.8.2.3. state

```
int state
```

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- gameSpace.h

3.9. StringList struktúrareferencia

String lista.

```
#include <essentials.h>
```

Adatmezők

- StringNode * startNode
- int len

3.9.1. Részletes leírás

String lista.

3.9.2. Adatmezők dokumentációja

3.9.2.1. len

```
int len
```

3.9.2.2. startNode

```
StringNode* startNode
```

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- essentials.h

3.10. StringNode struktúrareferencia

String lista elem.

`#include <essentials.h>`

Adatmezők

- struct StringNode * nextNode
- char * value

3.10.1. Részletes leírás

String lista elem.

3.10.2. Adatmezők dokumentációja

3.10.2.1. nextNode

```
struct StringNode* nextNode
```

3.10.2.2. value

```
char* value
```

Ez a dokumentáció a struktúráról a következő fájl alapján készült:

- essentials.h

4. fejezet

Fájlok dokumentációja

4.1. debugmalloc.h fájlreferencia

```
#include <stdbool.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdarg.h>
#include <unistd.h>
```

Adatszerkezetek

- struct DebugmallocEntry
- struct DebugmallocData

Makródefiníciók

- #define malloc(S) debugmalloc_malloc_full((S), "malloc", #S, __FILE__, __LINE__, false)
- #define calloc(N, S) debugmalloc_malloc_full((N)∗(S), "calloc", #N ", " #S, __FILE__, __LINE__, true)
- #define realloc(P, S) debugmalloc_realloc_full((P), (S), "realloc", #S, __FILE__, __LINE__)
- #define free(P) debugmalloc_free_full((P), "free", __FILE__, __LINE__)

Típusdefiníciók

- typedef struct DebugmallocEntry DebugmallocEntry
- typedef struct DebugmallocData DebugmallocData

Enumerációk

- enum { debugmalloc_canary_size = 64 , debugmalloc_canary_char = 'K' , debugmalloc_tablesize = 256 , debugmalloc_max_block_size_default = 1048576 }

4.1.1. Makródefiníciók dokumentációja

4.1.1.1. calloc

```
#define calloc(
              N,
              S ) debugmalloc_malloc_full((N)*(S), "calloc", #N ", " #S, __FILE__, __LINE__↩
, true)
```

4.1.1.2. free

```
#define free(
              P ) debugmalloc_free_full((P), "free", __FILE__, __LINE__)
```

4.1.1.3. malloc

```
#define malloc(
              S ) debugmalloc_malloc_full((S), "malloc", #S, __FILE__, __LINE__, false)
```

4.1.1.4. realloc

```
#define realloc(
              P,
              S ) debugmalloc_realloc_full((P), (S), "realloc", #S, __FILE__, __LINE__)
```

4.1.2. Típusdefiníciók dokumentációja

4.1.2.1. DebugmallocData

```
typedef struct DebugmallocData DebugmallocData
```

4.1.2.2. DebugmallocEntry

```
typedef struct DebugmallocEntry DebugmallocEntry
```

4.1.3. Enumerációk dokumentációja

4.1.3.1. anonymous enum

```
anonymous enum
```

Enumeráció-értékek

debugmalloc_canary_size	
debugmalloc_canary_char	
debugmalloc_tablesize	
debugmalloc_max_block_size_default	

4.2. debugmalloc.h

Ugrás a fájl dokumentációjához.

```
1 #ifndef DEBUGMALLOC_H
2 #define DEBUGMALLOC_H
3
4 #include <stdbool.h>
5 #include <stddef.h>
6 #include <stdlib.h>
7 #include <stdio.h>
8 #include <ctype.h>
9 #include <string.h>
10 #include <stdarg.h>
11
12
13 enum {
14     /* size of canary in bytes.  should be multiple of largest alignment
15 * required by any data type (usually 8 or 16) */
16     debugmalloc_canary_size = 64,
17
18     /* canary byte */
19     debugmalloc_canary_char = 'K',
20
21     /* hash table size for allocated entries */
22     debugmalloc_tablesize = 256,
23
24     /* max block size for allocation, can be modified with debugmalloc_max_block_size() */
25     debugmalloc_max_block_size_default = 1048576
26 };
27
28
29 /* make getpid and putenv "crossplatform".  deprecated on windows but they work just fine,
30 * however not declared.  */
31 #ifdef _WIN32
32     /* windows */
33 #include <process.h>
34 #ifdef _MSC_VER
35         /* visual studio, getenv/getpid deprecated warning */
36 #pragma warning(disable:  4996)
37 #else
38         /* other windows.  the declaration is unfortunately hidden
39 * in mingw header files by ifdefs.  */
40         int putenv(const char *);
41 #endif
42 #else
43     /* posix */
44 #include <unistd.h>
45 #endif
46
47
48 /* linked list entry for allocated blocks */
49 typedef struct DebugmallocEntry {
50     void *real_mem;     /* the address of the real allocation */
51     void *user_mem;     /* address shown to the user */
52     size_t size;        /* size of block requested by user */
53
54     char file[64];      /* malloc called in this file */
55     unsigned line;      /* malloc called at this line in file */
56     char func[32];      /* allocation function called (malloc, calloc, realloc) */
57     char expr[128];     /* expression calculating the size of allocation */
58
59     struct DebugmallocEntry *prev, *next;  /* for doubly linked list */
60 } DebugmallocEntry;
61
62
63 /* debugmalloc singleton, storing all state */
64 typedef struct DebugmallocData {
65     char logfile[256];    /* log file name or empty string */
66     long max_block_size;  /* max size of a single block allocated */
```

```
67     long alloc_count;     /* currently allocated; decreased with free */
68     long long alloc_bytes;
69     long all_alloc_count; /* all allocations, never decreased */
70     long long all_alloc_bytes;
71     DebugmallocEntry head[debugmalloc_tablesize], tail[debugmalloc_tablesize];  /* head and tail elements
       of allocation lists */
72 } DebugmallocData;
73
74
75 /* this forward declaration is required by the singleton manager function */
76 static DebugmallocData * debugmalloc_create(void);
77
78
79 /* creates singleton instance.  as this function is static included to different
80 * translation units, multiple instances of the static variables are created.
81 * to make sure it is really a singleton, these instances must know each other
82 * somethow.  an environment variable is used for that purpose, ie.  the address
83 * of the singleton allocated is stored by the operating system.
84 * this implementation is not thread-safe.  */
85 static DebugmallocData * debugmalloc_singleton(void) {
86     static char envstr[100];
87     static void *instance = NULL;
88
89     /* if we do not know the address of the singleton:
90 * - maybe we are the one to create it (env variable also does not exist)
91 * - or it is already created, and stored in the env variable.  */
92     if (instance == NULL) {
93         char envvarname[100] = "";
94         sprintf(envvarname, "%s%d", "debugmallocsingleton", (int) getpid());
95         char *envptr = getenv(envvarname);
96         if (envptr == NULL) {
97             /* no env variable:  create singleton.  */
98             instance = debugmalloc_create();
99             sprintf(envstr, "%s=%p", envvarname, instance);
100             putenv(envstr);
101         } else {
102             /* another copy of this function already created it.  */
103             int ok = sscanf(envptr, "%p", &instance);
104             if (ok != 1) {
105                 fprintf(stderr, "debugmalloc:  nem lehet ertelmezni:  %s!\n", envptr);
106                 abort();
107             }
108         }
109     }
110
111     return (DebugmallocData *) instance;
112 }
113
114
115 /* better version of strncpy, always terminates string with \0.  */
116 static void debugmalloc_strlcpy(char *dest, char const *src, size_t destsize) {
117     strncpy(dest, src, destsize);
118     dest[destsize - 1] = '\0';
119 }
120
121
122 /* set the name of the log file for debugmalloc.  empty filename
123 * means logging to stderr.  */
124 static void debugmalloc_log_file(char const *logfilename) {
125     if (logfilename == NULL)
126         logfilename = "";
127     DebugmallocData *instance = debugmalloc_singleton();
128     debugmalloc_strlcpy(instance->logfile, logfilename, sizeof(instance->logfile));
129 }
130
131
132 /* set the maximum size of one block.  useful for debugging purposes.  */
133 static void debugmalloc_max_block_size(long max_block_size) {
134     DebugmallocData *instance = debugmalloc_singleton();
135     instance->max_block_size = max_block_size;
136 }
137
138
139
140 /* printf to the log file, or stderr.  */
141 static void debugmalloc_log(char const *format, ...)  {
142     DebugmallocData *instance = debugmalloc_singleton();
143     FILE *f = stderr;
144     if (instance->logfile[0] != '\0') {
145         f = fopen(instance->logfile, "at");
146         if (f == NULL) {
147             f = stderr;
148             fprintf(stderr, "debugmalloc:  nem tudom megnyitni a %s fajlt irasra!\n",
    instance->logfile);
149             debugmalloc_strlcpy(instance->logfile, "", sizeof(instance->logfile));
150         }
151     }
```

```
152
153     va_list ap;
154     va_start(ap, format);
155     vfprintf(f, format, ap);
156     va_end(ap);
157
158     if (f != stderr)
159         fclose(f);
160 }
161
162
163 /* initialize a memory block allocated for the user.  the start and the end
164 * of the block is initialized with the canary characters.  if 'zero' is
165 * true, the user memory area is zero-initialized, otherwise it is also
166 * filled with the canary character to simulate garbage in memory.  */
167 static void debugmalloc_memory_init(DebugmallocEntry *elem, bool zero) {
168     unsigned char *real_mem = (unsigned char *) elem->real_mem;
169     unsigned char *user_mem = (unsigned char *) elem->user_mem;
170     unsigned char *canary1 = real_mem;
171     unsigned char *canary2 = real_mem + debugmalloc_canary_size + elem->size;
172     memset(canary1, debugmalloc_canary_char, debugmalloc_canary_size);
173     memset(canary2, debugmalloc_canary_char, debugmalloc_canary_size);
174     memset(user_mem, zero ?  0 :  debugmalloc_canary_char, elem->size);
175 }
176
177 /* check canary, return true if ok, false if corrupted.  */
178 static bool debugmalloc_canary_ok(DebugmallocEntry const *elem) {
179     unsigned char *real_mem = (unsigned char *) elem->real_mem;
180     unsigned char *canary1 = real_mem;
181     unsigned char *canary2 = real_mem + debugmalloc_canary_size + elem->size;
182     for (size_t i = 0; i < debugmalloc_canary_size; ++i) {
183         if (canary1[i] != debugmalloc_canary_char)
184             return false;
185         if (canary2[i] != debugmalloc_canary_char)
186             return false;
187     }
188     return true;
189 }
190
191
192 /* dump memory contents to log file.  */
193 static void debugmalloc_dump_memory(char const *mem, size_t size) {
194     for (unsigned y = 0; y < (size + 15) / 16; y++) {
195         char line[80];
196         int pos = 0;
197         pos += sprintf(line + pos, "      %04x  ", y * 16);
198         for (unsigned x = 0; x < 16; x++) {
199             if (y * 16 + x < size)
200                 pos += sprintf(line + pos, "%02x ", mem[y * 16 + x]);
201             else
202                 pos += sprintf(line + pos, "   ");
203         }
204         pos += sprintf(line + pos, "  ");
205         for (unsigned x = 0; x < 16; x++) {
206             if (y * 16 + x < size) {
207                 unsigned char c = mem[y * 16 + x];
208                 pos += sprintf(line + pos, "%c", isprint(c) ?  c :  '.');
209             }
210             else {
211                 pos += sprintf(line + pos, " ");
212             }
213         }
214         debugmalloc_log("%s\n", line);
215     }
216 }
217
218
219 /* dump data of allocated memory block.
220 * if the canary is corrupted, it is also written to the log.  */
221 static void debugmalloc_dump_elem(DebugmallocEntry const *elem) {
222     bool canary_ok = debugmalloc_canary_ok(elem);
223
224     debugmalloc_log("  %p, %u bajt, kanari:  %s\n"
225                     "  %s:%u, %s(%s)\n",
226                        elem->user_mem, (unsigned) elem->size, canary_ok ?  "ok" :  "**SERULT**",
227                        elem->file, elem->line,
228                        elem->func, elem->expr);
229
230     if (!canary_ok) {
231         debugmalloc_log("    ELOTTE kanari:  \n");
232         debugmalloc_dump_memory((char const *) elem->real_mem, debugmalloc_canary_size);
233     }
234
235     debugmalloc_dump_memory((char const *) elem->user_mem, elem->size > 64 ?  64 :  elem->size);
236
237     if (!canary_ok) {
238         debugmalloc_log("    UTANA kanari:  \n");
```

```
239         debugmalloc_dump_memory((char const *) elem->real_mem + debugmalloc_canary_size + elem->size,
    debugmalloc_canary_size);
240     }
241 }
242
243
244 /* dump data of all memory blocks allocated.  */
245 static void debugmalloc_dump(void) {
246     DebugmallocData *instance = debugmalloc_singleton();
247     debugmalloc_log("** DEBUGMALLOC DUMP **********************************\n");
248     int cnt = 0;
249     for (size_t i = 0; i < debugmalloc_tablesize; i++) {
250         DebugmallocEntry *head = &instance->head[i];
251         for (DebugmallocEntry *iter = head->next; iter->next != NULL; iter = iter->next) {
252             ++cnt;
253             debugmalloc_log("** %d/%d.  rekord:\n", cnt, instance->alloc_count);
254             debugmalloc_dump_elem(iter);
255         }
256     }
257     debugmalloc_log("** DEBUGMALLOC DUMP VEGE ******************************\n");
258 }
259
260
261 /* called at program exit to dump data if there is a leak,
262 * ie.  allocated block remained.  */
263 static void debugmalloc_atexit_dump(void) {
264     DebugmallocData *instance = debugmalloc_singleton();
265
266     if (instance->alloc_count > 0) {
267         debugmalloc_log("\n"
268                         "*********************************************************\n"
269                         "* MEMORIASZIVARGAS VAN A PROGRAMBAN!!!\n"
270                         "*********************************************************\n"
271                         "\n");
272         debugmalloc_dump();
273     } else {
274         debugmalloc_log("*************************************************\n"
275                         "* Debugmalloc:  nincs memoriaszivargas a programban.\n"
276                         "* Osszes foglalas:  %d blokk, %d bajt.\n"
277                         "*************************************************\n",
278                         instance->all_alloc_count, instance->all_alloc_bytes);
279     }
280 }
281
282
283 /* hash function for bucket hash.  */
284 static size_t debugmalloc_hash(void *address) {
285     /* the last few bits are ignored, as they are usually zero for
286 * alignment purposes.  all tested architectures used 16 byte allocation.  */
287     size_t cut = (size_t)address >> 4;
288     return cut % debugmalloc_tablesize;
289 }
290
291
292 /* insert element to hash table.  */
293 static void debugmalloc_insert(DebugmallocEntry *entry) {
294     DebugmallocData *instance = debugmalloc_singleton();
295     size_t idx = debugmalloc_hash(entry->user_mem);
296     DebugmallocEntry *head = &instance->head[idx];
297     entry->prev = head;
298     entry->next = head->next;
299     head->next->prev = entry;
300     head->next = entry;
301     instance->alloc_count += 1;
302     instance->alloc_bytes += entry->size;
303     instance->all_alloc_count += 1;
304     instance->all_alloc_bytes += entry->size;
305 }
306
307
308 /* remove element from hash table */
309 static void debugmalloc_remove(DebugmallocEntry *entry) {
310     DebugmallocData *instance = debugmalloc_singleton();
311     entry->next->prev = entry->prev;
312     entry->prev->next = entry->next;
313     instance->alloc_count -= 1;
314     instance->alloc_bytes -= entry->size;
315 }
316
317
318 /* find element in hash table, given with the memory address that the user sees.
319 * @return the linked list entry, or null if not found.  */
320 static DebugmallocEntry *debugmalloc_find(void *mem) {
321     DebugmallocData *instance = debugmalloc_singleton();
322     size_t idx = debugmalloc_hash(mem);
323     DebugmallocEntry *head = &instance->head[idx];
324     for (DebugmallocEntry *iter = head->next; iter->next != NULL; iter = iter->next)
```

```
325         if (iter->user_mem == mem)
326             return iter;
327     return NULL;
328 }
329 
330 
331 /* allocate memory.  this function is called via the macro.  */
332 static void *debugmalloc_malloc_full(size_t size, char const *func, char const *expr, char const *file,
      unsigned line, bool zero) {
333     /* imitate standard malloc:  return null if size is zero */
334     if (size == 0)
335         return NULL;
336 
337     /* check max size */
338     DebugmallocData *instance = debugmalloc_singleton();
339     if (size > instance->max_block_size) {
340         debugmalloc_log("debugmalloc:  %s @ %s:%u:  a blokk merete tul nagy, %u bajt;
    debugmalloc_max_block_size() fuggvennyel novelheto.\n", func, file, line, (unsigned) size);
341         abort();
342     }
343 
344     /* allocate more memory, make room for canary */
345     void *real_mem = malloc(size + 2 * debugmalloc_canary_size);
346     if (real_mem == NULL) {
347         debugmalloc_log("debugmalloc:  %s @ %s:%u:  nem sikerult %u meretu memoriat foglalni!\n", func,
    file, line, (unsigned) size);
348         return NULL;
349     }
350 
351     /* allocate memory for linked list element */
352     DebugmallocEntry *newentry = (DebugmallocEntry *) malloc(sizeof(DebugmallocEntry));
353     if (newentry == NULL) {
354         free(real_mem);
355         debugmalloc_log("debugmalloc:  %s @ %s:%u:  le tudtam foglalni %u memoriat, de utana a sajatnak
    nem, sry\n", func, file, line, (unsigned) size);
356         abort();
357     }
358 
359     /* metadata of allocation:  caller function, code line etc.  */
360     debugmalloc_strlcpy(newentry->func, func, sizeof(newentry->func));
361     debugmalloc_strlcpy(newentry->expr, expr, sizeof(newentry->expr));
362     debugmalloc_strlcpy(newentry->file, file, sizeof(newentry->file));
363     newentry->line = line;
364 
365     /* address of allocated memory chunk */
366     newentry->real_mem = real_mem;
367     newentry->user_mem = (unsigned char *) real_mem + debugmalloc_canary_size;
368     newentry->size = size;
369     debugmalloc_memory_init(newentry, zero);
370 
371     /* store in list and return pointer to user area */
372     debugmalloc_insert(newentry);
373     return newentry->user_mem;
374 }
375 
376 
377 /* free memory and remove list item.  before deleting, the chuck is filled with
378 * the canary byte to make sure that the user will see garbage if the memory
379 * is accessed after freeing.  */
380 static void debugmalloc_free_inner(DebugmallocEntry *deleted) {
381     debugmalloc_remove(deleted);
382 
383     /* fill with garbage, then remove from linked list */
384     memset(deleted->real_mem, debugmalloc_canary_char, deleted->size + 2 * debugmalloc_canary_size);
385     free(deleted->real_mem);
386     free(deleted);
387 }
388 
389 
390 /* free memory - called via the macro.
391 * as all allocations are tracked in the list, this function can terminate the program
392 * if a block is freed twice or the free function is called with an invalid address.  */
393 static void debugmalloc_free_full(void *mem, char const *func, char const *file, unsigned line) {
394     /* imitate standard free function:  if ptr is null, no operation is performed */
395     if (mem == NULL)
396         return;
397 
398     /* find allocation, abort if not found */
399     DebugmallocEntry *deleted = debugmalloc_find(mem);
400     if (deleted == NULL) {
401         debugmalloc_log("debugmalloc:  %s @ %s:%u:  olyan teruletet probalsz felszabaditani, ami nincs
    lefoglalva!\n", func, file, line);
402         abort();
403     }
404 
405     /* check canary and then free memory */
406     if (!debugmalloc_canary_ok(deleted)) {
```

```
407         debugmalloc_log("debugmalloc:  %s @ %s:%u:  a %p memoriateruletet tulindexelted!\n", func, file,
    line, mem);
408         debugmalloc_dump_elem(deleted);
409     }
410     debugmalloc_free_inner(deleted);
411 }
412
413
414 /* realloc-like function.  */
415 static void *debugmalloc_realloc_full(void *oldmem, size_t newsize, char const *func, char const *expr,
    char const *file, unsigned line) {
416     /* imitate standard realloc:  equivalent to free if size is null.  */
417     if (newsize == 0) {
418         debugmalloc_free_full(oldmem, func, file, line);
419         return NULL;
420     }
421     /* imitate standard realloc:  equivalent to malloc if first param is NULL */
422     if (oldmem == NULL)
423         return debugmalloc_malloc_full(newsize, func, expr, file, line, 0);
424
425     /* find old allocation.  abort if not found.  */
426     DebugmallocEntry *oldentry = debugmalloc_find(oldmem);
427     if (oldentry == NULL) {
428         debugmalloc_log("debugmalloc:  %s @ %s:%u:  olyan teruletet probalsz atmeretezni, ami nincs
    lefoglalva!\n", func, file, line);
429         abort();
430     }
431
432     /* create new allocation, copy & free old data */
433     void *newmem = debugmalloc_malloc_full(newsize, func, expr, file, line, false);
434     if (newmem == NULL) {
435         debugmalloc_log("debugmalloc:  %s @ %s:%u:  nem sikerult uj memoriat foglalni az
    atmeretezeshez!\n", func, file, line);
436         /* imitate standard realloc:  original block is untouched, but return NULL */
437         return NULL;
438     }
439     size_t smaller = oldentry->size < newsize ?  oldentry->size :  newsize;
440     memcpy(newmem, oldmem, smaller);
441     debugmalloc_free_inner(oldentry);
442
443     return newmem;
444 }
445
446
447 /* initialize debugmalloc singleton.  returns the newly allocated instance */
448 static DebugmallocData * debugmalloc_create(void) {
449     /* config check */
450     if (debugmalloc_canary_size % 16 != 0) {
451         debugmalloc_log("debugmalloc:  a kanari merete legyen 16-tal oszthato\n");
452         abort();
453     }
454     if (debugmalloc_canary_char == 0) {
455         debugmalloc_log("debugmalloc:  a kanari legyen 0-tol kulonbozo\n");
456         abort();
457     }
458     /* avoid compiler warning if these functions are not used */
459     (void) debugmalloc_realloc_full;
460     (void) debugmalloc_log_file;
461     (void) debugmalloc_max_block_size;
462
463     /* create and initialize instance */
464     DebugmallocData *instance = (DebugmallocData *) malloc(sizeof(DebugmallocData));
465     if (instance == NULL) {
466         debugmalloc_log("debugmalloc:  nem sikerult elinditani a memoriakezelest\n");
467         abort();
468     }
469     debugmalloc_strlcpy(instance->logfile, "", sizeof(instance->logfile));
470     instance->max_block_size = debugmalloc_max_block_size_default;
471     instance->alloc_count = 0;
472     instance->alloc_bytes = 0;
473     instance->all_alloc_count = 0;
474     instance->all_alloc_bytes = 0;
475     for (size_t i = 0; i < debugmalloc_tablesize; i++) {
476         instance->head[i].prev = NULL;
477         instance->head[i].next = &instance->tail[i];
478         instance->tail[i].next = NULL;
479         instance->tail[i].prev = &instance->head[i];
480     }
481
482     atexit(debugmalloc_atexit_dump);
483     return instance;
484 }
485
486
487 /* These macro-like functions forward all allocation/free
488 * calls to debugmalloc.  Usage is the same, malloc(size)
489 * gives the address of a new memory block, free(ptr)
```

```
490 * deallocates etc.
491 *
492 * If you use this file, make sure that you include this
493 * in *ALL* translation units (*.c) of your source.  The
494 * builtin free() function cannot deallocate a memory block
495 * that was allocated via debugmalloc, yet the name of
496 * the function is the same!  */
497
498 #define malloc(S) debugmalloc_malloc_full((S), "malloc", #S, __FILE__, __LINE__, false)
499 #define calloc(N,S) debugmalloc_malloc_full((N)*(S), "calloc", #N ", " #S, __FILE__, __LINE__, true)
500 #define realloc(P,S) debugmalloc_realloc_full((P), (S), "realloc", #S, __FILE__, __LINE__)
501 #define free(P) debugmalloc_free_full((P), "free", __FILE__, __LINE__)
502
503 #endif
```

4.3. econio.c fájlreferencia

```
#include "econio.h"
#include <assert.h>
#include <stdio.h>
#include <termios.h>
#include <unistd.h>
#include <stdbool.h>
#include <sys/time.h>
#include <sys/types.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
```

Függvények

- void econio_textcolor (int color)
- void econio_textbackground (int color)
- void econio_gotoxy (int x, int y)
- void econio_clrscr ()
- void econio_flush ()
- void econio_set_title (char const *title)
- void econio_rawmode ()
- void econio_normalmode ()
- bool econio_kbhit ()
- int econio_getch ()
- void econio_sleep (double sec)

4.3.1. Függvények dokumentációja

4.3.1.1. econio_clrscr()

```
void econio_clrscr ( )
```

Clear the screen and return the cursor to the upper left position.

4.3.1.2. econio_flush()

```
void econio_flush ( )
```

Send output to the terminal. To be called if many characters were drawn to the terminal and there was no at the end.

4.3.1.3. econio_getch()

```
int econio_getch ( )
```

Get one raw character from terminal. This can detect F1-F10, cursor keys, backspace and other controlling keys: see the keyboard constants.DIW ASCII code is returned for other keys. Non-ASCII keys probably won't work. Characters are not echoed to the screen when in raw mode. Only to be used after calling econio_rawmode(). Note that backspace will be code 8, regardless of terminal settings (whether it sent BS or DEL char). Enter will always be 10, even on Windows. On Windows, this function sometimes returns 0's, so just ignore them. Also function keys are not supported on Windows.

4.3.1.4. econio_gotoxy()

```
void econio_gotoxy (
            int x,
            int y )
```

Jump to position (x, y) with the cursor. Upper left corner is (0, 0).

4.3.1.5. econio_kbhit()

```
bool econio_kbhit ( )
```

Detect if a key is pressed. If so, it can be read with econio_getch(). Only to be used after calling econio_rawmode().

4.3.1.6. econio_normalmode()

```
void econio_normalmode ( )
```

Switch the terminal back to normal, line-oriented mode. Characters are echoed to the screen when in normal mode.

4.3.1.7. econio_rawmode()

```
void econio_rawmode ( )
```

Switch the terminal to raw mode, to detect F1-F10, cursor keys and other controlling keys. Use econio_getch() and econio_kbhit() afterwards. Characters are not echoed to the screen when in raw mode. Switching back to line-oriented mode is possible using econio_normalmode().

4.3.1.8. econio_set_title()

```
void econio_set_title (
            char const * title )
```

Set the title of the terminal window.

4.3.1.9. econio_sleep()

```
void econio_sleep (
            double sec )
```

Delay for the specified amount of time (sec can be an arbitrary floating point number, not just integer).

4.3.1.10. econio_textbackground()

```
void econio_textbackground (
            int color )
```

Change background color to the one specified. See the color constants in the colors header.

4.3.1.11. econio_textcolor()

```
void econio_textcolor (
            int color )
```

Change text color to the one specified. See the color constants in the colors header.

4.4. econio.h fájlreferencia

```
#include <stdbool.h>
```

Típusdefiníciók

- typedef enum EconioColor EconioColor
- typedef enum EconioKey EconioKey

Enumerációk

- enum EconioColor {
  COL_BLACK = 0 , COL_BLUE = 1 , COL_GREEN = 2 , COL_CYAN = 3 ,
  COL_RED = 4 , COL_MAGENTA = 5 , COL_BROWN = 6 , COL_LIGHTGRAY = 7 ,
  COL_DARKGRAY = 8 , COL_LIGHTBLUE = 9 , COL_LIGHTGREEN = 10 , COL_LIGHTCYAN = 11 ,
  COL_LIGHTRED = 12 , COL_LIGHTMAGENTA = 13 , COL_YELLOW = 14 , COL_LIGHTYELLOW = COL↩
  _YELLOW ,
  COL_WHITE = 15 , COL_RESET = 16 }
- enum EconioKey {
  KEY_F1 = -1 , KEY_F2 = -2 , KEY_F3 = -3 , KEY_F4 = -4 ,
  KEY_F5 = -5 , KEY_F6 = -6 , KEY_F7 = -7 , KEY_F8 = -8 ,
  KEY_F9 = -9 , KEY_F10 = -10 , KEY_F11 = -11 , KEY_F12 = -12 ,
  KEY_UP = -20 , KEY_DOWN = -21 , KEY_LEFT = -22 , KEY_RIGHT = -23 ,
  KEY_PAGEUP = -24 , KEY_PAGEDOWN = -25 , KEY_HOME = -26 , KEY_END = -27 ,
  KEY_INSERT = -28 , KEY_DELETE = -29 , KEY_CTRLUP = -30 , KEY_CTRLDOWN = -31 ,
  KEY_CTRLLEFT = -32 , KEY_CTRLRIGHT = -33 , KEY_CTRLPAGEUP = -34 , KEY_CTRLPAGEDOWN =
  -35 ,
  KEY_CTRLHOME = -36 , KEY_CTRLEND = -37 , KEY_CTRLINSERT = -38 , KEY_CTRLDELETE = -39 ,
  KEY_UNKNOWNKEY = -255 , KEY_BACKSPACE = 8 , KEY_ENTER = 10 , KEY_ESCAPE = 27 ,
  KEY_TAB = 9 }

Függvények

- void econio_textcolor (int color)
- void econio_textbackground (int color)
- void econio_gotoxy (int x, int y)
- void econio_clrscr ()
- void econio_flush ()
- void econio_set_title (char const *title)
- void econio_rawmode ()
- void econio_normalmode ()
- bool econio_kbhit ()
- int econio_getch ()
- void econio_sleep (double sec)

4.4.1. Típusdefiníciók dokumentációja

4.4.1.1. EconioColor

```
typedef enum EconioColor EconioColor
```

4.4.1.2. EconioKey

```
typedef enum EconioKey EconioKey
```

4.4.2. Enumerációk dokumentációja

4.4.2.1. EconioColor

```
enum EconioColor
```

Enumeráció-értékek

COL_BLACK	
COL_BLUE	
COL_GREEN	
COL_CYAN	
COL_RED	
COL_MAGENTA	
COL_BROWN	
COL_LIGHTGRAY	
COL_DARKGRAY	
COL_LIGHTBLUE	
COL_LIGHTGREEN	
COL_LIGHTCYAN	
COL_LIGHTRED	
COL_LIGHTMAGENTA	
COL_YELLOW	
COL_LIGHTYELLOW	
COL_WHITE	
COL_RESET	

4.4.2.2. EconioKey

```
enum EconioKey
```

Enumeráció-értékek

KEY_F1	
KEY_F2	
KEY_F3	
KEY_F4	
KEY_F5	
KEY_F6	
KEY_F7	
KEY_F8	
KEY_F9	
KEY_F10	
KEY_F11	
KEY_F12	
KEY_UP	
KEY_DOWN	
KEY_LEFT	
KEY_RIGHT	
KEY_PAGEUP	
KEY_PAGEDOWN	
KEY_HOME	
KEY_END	
KEY_INSERT	

Enumeráció-értékek

KEY_DELETE	
KEY_CTRLUP	
KEY_CTRLDOWN	
KEY_CTRLLEFT	
KEY_CTRLRIGHT	
KEY_CTRLPAGEUP	
KEY_CTRLPAGEDOWN	
KEY_CTRLHOME	
KEY_CTRLEND	
KEY_CTRLINSERT	
KEY_CTRLDELETE	
KEY_UNKNOWNKEY	
KEY_BACKSPACE	
KEY_ENTER	
KEY_ESCAPE	
KEY_TAB	

4.4.3. Függvények dokumentációja

4.4.3.1. econio_clrscr()

```
void econio_clrscr ( )
```

Clear the screen and return the cursor to the upper left position.

4.4.3.2. econio_flush()

```
void econio_flush ( )
```

Send output to the terminal. To be called if many characters were drawn to the terminal and there was no at the end.

4.4.3.3. econio_getch()

```
int econio_getch ( )
```

Get one raw character from terminal. This can detect F1-F10, cursor keys, backspace and other controlling keys↩ : see the keyboard constants.DIW ASCII code is returned for other keys. Non-ASCII keys probably won't work. Characters are not echoed to the screen when in raw mode. Only to be used after calling econio_rawmode(). Note that backspace will be code 8, regardless of terminal settings (whether it sent BS or DEL char). Enter will always be 10, even on Windows. On Windows, this function sometimes returns 0's, so just ignore them. Also function keys are not supported on Windows.

4.4.3.4. econio_gotoxy()

```
void econio_gotoxy (
            int x,
            int y )
```

Jump to position (x, y) with the cursor. Upper left corner is (0, 0).

4.4.3.5. econio_kbhit()

```
bool econio_kbhit ( )
```

Detect if a key is pressed. If so, it can be read with econio_getch(). Only to be used after calling econio_rawmode().

4.4.3.6. econio_normalmode()

```
void econio_normalmode ( )
```

Switch the terminal back to normal, line-oriented mode. Characters are echoed to the screen when in normal mode.

4.4.3.7. econio_rawmode()

```
void econio_rawmode ( )
```

Switch the terminal to raw mode, to detect F1-F10, cursor keys and other controlling keys. Use econio_getch() and econio_kbhit() afterwards. Characters are not echoed to the screen when in raw mode. Switching back to line-oriented mode is possible using econio_normalmode().

4.4.3.8. econio_set_title()

```
void econio_set_title (
            char const * title )
```

Set the title of the terminal window.

4.4.3.9. econio_sleep()

```
void econio_sleep (
            double sec )
```

Delay for the specified amount of time (sec can be an arbitrary floating point number, not just integer).

4.4.3.10. econio_textbackground()

```
void econio_textbackground (
            int color )
```

Change background color to the one specified. See the color constants in the colors header.

4.4.3.11. econio_textcolor()

```
void econio_textcolor (
             int color )
```

Change text color to the one specified. See the color constants in the colors header.

4.5. econio.h

Ugrás a fájl dokumentációjához.

```
1 #ifndef ECONIO_H
2 #define ECONIO_H
3
4 #include <stdbool.h>
5
6 #ifdef __cplusplus
7 extern "C" {
8 #endif
9
10 typedef enum EconioColor {
11     COL_BLACK = 0,
12     COL_BLUE = 1,
13     COL_GREEN = 2,
14     COL_CYAN = 3,
15     COL_RED = 4,
16     COL_MAGENTA = 5,
17     COL_BROWN = 6,
18     COL_LIGHTGRAY = 7,
19     COL_DARKGRAY = 8,
20     COL_LIGHTBLUE = 9,
21     COL_LIGHTGREEN = 10,
22     COL_LIGHTCYAN = 11,
23     COL_LIGHTRED = 12,
24     COL_LIGHTMAGENTA = 13,
25     COL_YELLOW = 14,    COL_LIGHTYELLOW = COL_YELLOW,
26     COL_WHITE = 15,
27     COL_RESET = 16,
28 } EconioColor;
29
30
31 typedef enum EconioKey {
32     // function keys are not supported on Windows
33     KEY_F1 = -1,
34     KEY_F2 = -2,
35     KEY_F3 = -3,
36     KEY_F4 = -4,
37     KEY_F5 = -5,
38     KEY_F6 = -6,
39     KEY_F7 = -7,
40     KEY_F8 = -8,
41     KEY_F9 = -9,
42     KEY_F10 = -10,
43     KEY_F11 = -11,
44     KEY_F12 = -12,
45
46     KEY_UP = -20,
47     KEY_DOWN = -21,
48     KEY_LEFT = -22,
49     KEY_RIGHT = -23,
50     KEY_PAGEUP = -24,
51     KEY_PAGEDOWN = -25,
52     KEY_HOME = -26,
53     KEY_END = -27,
54     KEY_INSERT = -28,
55     KEY_DELETE = -29,
56
57     KEY_CTRLUP = -30,
58     KEY_CTRLDOWN = -31,
59     KEY_CTRLLEFT = -32,
60     KEY_CTRLRIGHT = -33,
61     KEY_CTRLPAGEUP = -34,
62     KEY_CTRLPAGEDOWN = -35,
63     KEY_CTRLHOME = -36,
64     KEY_CTRLEND = -37,
65     KEY_CTRLINSERT = -38,
66     KEY_CTRLDELETE = -39,
67
68     KEY_UNKNOWNKEY = -255,
```

```
69
70      KEY_BACKSPACE = 8,
71      KEY_ENTER = 10,
72      KEY_ESCAPE = 27,
73      KEY_TAB = 9,
74 } EconioKey;
75
76
80 void econio_textcolor(int color);
81
82
87 void econio_textbackground(int color);
88
89
91 void econio_gotoxy(int x, int y);
92
93
95 void econio_clrscr();
96
97
102 void econio_flush();
103
104
106 void econio_set_title(char const *title);
107
108
115 void econio_rawmode();
116
117
122 void econio_normalmode();
123
124
129 bool econio_kbhit();
130
131
143 int econio_getch();
144
145
150 void econio_sleep(double sec);
151
152 #ifdef __cplusplus
153 } /* extern "C" */
154 #endif
155
156 #endif
```

4.6. essentials.c fájlreferencia

Az összes máshová nem illő de szükséges függvények. Nagyrészt lista kezelő függvények.

```
#include "debugmalloc.h"
#include "essentials.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
```

Függvények

- List * CreateList (int startValue)

  Létrehoz egy int listát egy kezdőértékkel. A hívó felelőssége meghívni a listára a DestroyList() függvényt.
- StringList * CreateStringList (char *start)

  Létrehoz egy String listát egy kezdőértékkel. A hívó felelőssége meghívni a listára a DestroyStringList() függvényt.
- void DestroyList (List *list)

  Felszabadítja a teljes int listát.
- void DestroyStringList (StringList *list)

  Felszabadítja a teljes String listát.

- void Add (List *list, int value)

  Hozzáad egy elemet egy int listához.
- void AddString (StringList *list, char *item)

  Hozzáad egy stringet elemet egy string listához.
- void AddStringPointer (StringList *list, char *item)

  Hozzáfűz egy String listához egy elemet aminek már le van foglalva a memória (csupán a kód gyorsításáért kell)
- int At (List *list, int index)

  Visszaadja a tömb bizonyos indexénél található értéket.
- char * AtString (StringList *list, int index)

  Visszaadja a tömb bizonyos indexénél található értéket.
- void Set (List *list, int index, int value)

  Beállítja a tömb bizonyos indexénél található értéket.
- void SetString (StringList *list, int index, char *value)

  Beállítja a tömb bizonyos indexénél található értéket.
- void RemoveFirst (List *list)

  Kitörli a lista első elemét.
- void RemoveFirstString (StringList *list)

  Kitörli a lista első elemét.
- void RemoveAt (StringList *list, int index)

  Kitörli a lista megadott elemét.
- void RemoveAtInt (List *list, int index)

  Kitörli a lista megadott elemét.
- int Find (List *list, int value)

  Megkeresi, hogy az érték hanyadik indexen található meg először. -1 ha az érték nem található
- int FindString (StringList *list, char *value)

  Megkeresi, hogy az érték hanyadik indexen található meg először. -1 ha az érték nem található
- bool Contains (List *list, int value)

  Megnézi, hogy tartalmazza-e a lista az elemet.
- bool ContainsString (StringList *list, void *value)

  Megnézi, hogy tartalmazza-e a lista az elemet.
- StringList * Split (char *string, char split)

  Az elválasztó karakter alapján több elemre szed szét egy stringet. A hívó felelőssége meghívni a DestroyStringList()-et.
- List * SplitInt (char *string, char split)

  Egy stringben megadott integereket szétszed az elválasztó karakter alapján. A hívó felelőssée meghívni a DestroyList()-et.
- char * Alloc (int n, FILE *fp)

  SortBeolvas segédfüggvénye.
- char * AllocStd (int n)

  SortBeolvasStd segédfüggvénye.
- char * SortBeolvas (FILE *fp)

  A függvény beolvas egy teljes sort (enterig vagy fájl vége jelig) a megadott fájlból, és visszaadja egy dinamikusan foglalt sztringben. A sztring nullával van lezárva, az entert nem tartalmazza. A hívó felelőssége a free()-t meghívni a kapott pointerre.
- char * SortBeolvasStd ()

  A függvény beolvas egy teljes sort (enterig vagy fájl vége jelig) , és visszaadja egy dinamikusan foglalt sztringben. A sztring nullával van lezárva, az entert nem tartalmazza. A hívó felelőssége a free()-t meghívni a kapott pointerre.
- int ConvertToInt (char *str)

  Egy stringet int-é alakít.
- char * CpyStrPtr (char *str)

  Készít egy másolatot egy stringből. A hívó felelőssége meghívni a free-t.
- char * Merge (StringList *list, int start, int end)

  Lista elemeit összeolvasztja egy stringbe, whitespace karakterrel elválasztva. A hívó felelőssége meghívni a free-t.

4.6.1. Részletes leírás

Az összes máshová nem illő de szükséges függvények. Nagyrészt lista kezelő függvények.

4.6.2. Függvények dokumentációja

4.6.2.1. Add()

```
void Add (
            List * list,
            int value )
```

Hozzáad egy elemet egy int listához.

Paraméterek

list	Lista
value	Érték amit hozzáad a listához

4.6.2.2. AddString()

```
void AddString (
            StringList * list,
            char * item )
```

Hozzáad egy stringet elemet egy string listához.

Paraméterek

list	Lista
value	Érték amit hozzáad a listához

4.6.2.3. AddStringPointer()

```
void AddStringPointer (
            StringList * list,
            char * item )
```

Hozzáfűz egy String listához egy elemet aminek már le van foglalva a memória (csupán a kód gyorsításáért kell)

Paraméterek

list	Lista
item	Pointer amit hozzá fog fűzni a listához

4.6.2.4. Alloc()

```
char * Alloc (
            int n,
            FILE * fp )
```

SortBeolvas segédfüggvénye.

4.6.2.5. AllocStd()

```
char * AllocStd (
            int n )
```

SortBeolvasStd segédfüggvénye.

4.6.2.6. At()

```
int At (
            List * list,
            int index )
```

Visszaadja a tömb bizonyos indexénél található értéket.

Paraméterek

list	Lista
index	index

Visszatérési érték

int Indexen található érték

4.6.2.7. AtString()

```
char * AtString (
            StringList * list,
            int index )
```

Visszaadja a tömb bizonyos indexénél található értéket.

Paraméterek

list	Lista
index	index

Visszatérési érték

char∗ Indexen található érték

4.6.2.8. Contains()

```
bool Contains (
            List * list,
            int value )
```

Megnézi, hogy tartalmazza-e a lista az elemet.

Paraméterek

list	Lista
value	Érték

Visszatérési érték

true Az elem benne van a listában

false Az elem nincs benne a listában

4.6.2.9. ContainsString()

```
bool ContainsString (
            StringList * list,
            void * value )
```

Megnézi, hogy tartalmazza-e a lista az elemet.

Paraméterek

list	Lista
value	Érték

Visszatérési érték

true Az elem benne van a listában

false Az elem nincs benne a listában

4.6.2.10. ConvertToInt()

```
int ConvertToInt (
            char * str )
```

Egy stringet int-é alakít.

Paraméterek

str	Átalakítandó string

Visszatérési érték

int Átalakított int

4.6.2.11. CpyStrPtr()

```
char * CpyStrPtr (
            char * str )
```

Készít egy másolatot egy stringből. A hívó felelőssége meghívni a free-t.

Paraméterek

str	Másolandó string

Visszatérési érték

char∗ Másolt string

4.6.2.12. CreateList()

```
List * CreateList (
            int startValue )
```

Létrehoz egy int listát egy kezdőértékkel. A hívó felelőssége meghívni a listára a DestroyList() függvényt.

Paraméterek

startValue	Kezdőérték

Visszatérési érték

List∗ Létrehozott lista

4.6.2.13. CreateStringList()

```
StringList * CreateStringList (
             char * start )
```

Létrehoz egy String listát egy kezdőértékkel. A hívó felelőssége meghívni a listára a DestroyStringList() függvényt.

Paraméterek

start	kezdő érték

Visszatérési érték

StringList∗ Létrehozott lista

4.6.2.14. DestroyList()

```
void DestroyList (
             List * list )
```

Felszabadítja a teljes int listát.

Paraméterek

list	Felszabadítandó lista

4.6.2.15. DestroyStringList()

```
void DestroyStringList (
             StringList * list )
```

Felszabadítja a teljes String listát.

Paraméterek

list	Felszabadítandó lista

4.6.2.16. Find()

```
int Find (
            List * list,
            int value )
```

Megkeresi, hogy az érték hanyadik indexen található meg először. -1 ha az érték nem található

Paraméterek

list	Lista
value	Keresendő érték

Visszatérési érték

int index

4.6.2.17. FindString()

```
int FindString (
            StringList * list,
            char * value )
```

Megkeresi, hogy az érték hanyadik indexen található meg először. -1 ha az érték nem található

Paraméterek

list	Lista
value	Keresendő érték

Visszatérési érték

int index

4.6.2.18. Merge()

```
char * Merge (
            StringList * list,
```

```
            int start,
            int end )
```

Lista elemeit összeolvasztja egy stringbe, whitespace karakterrel elválasztva. A hívó felelőssége meghívni a free-t.

Paraméterek

list	Lista
start	Kezdő index (inclusive)
end	Végső index (inclusive)

Visszatérési érték

char∗ Összeolvasztott string

4.6.2.19. RemoveAt()

```
void RemoveAt (
            StringList * list,
            int index )
```

Kitörli a lista megadott elemét.

Paraméterek

list	Lista
index	Index

4.6.2.20. RemoveAtInt()

```
void RemoveAtInt (
            List * list,
            int index )
```

Kitörli a lista megadott elemét.

Paraméterek

list	Lista
index	Index

4.6.2.21. RemoveFirst()

```
void RemoveFirst (
            List * list )
```

Kitörli a lista első elemét.

Paraméterek

list	Lista

4.6.2.22. RemoveFirstString()

```
void RemoveFirstString (
            StringList * list )
```

Kitörli a lista első elemét.

Paraméterek

list	Lista

4.6.2.23. Set()

```
void Set (
            List * list,
            int index,
            int value )
```

Beállítja a tömb bizonyos indexénél található értéket.

Paraméterek

list	Lista
index	Index
value	Beállítandó érték

4.6.2.24. SetString()

```
void SetString (
            StringList * list,
```

```
            int index,
            char * value )
```

Beállítja a tömb bizonyos indexénél található értéket.

Paraméterek

list	Lista
index	Index
value	Beállítandó érték

4.6.2.25. SortBeolvas()

```
char * SortBeolvas (
            FILE * fp )
```

A függvény beolvas egy teljes sort (enterig vagy fájl vége jelig) a megadott fájlból, és visszaadja egy dinamikusan foglalt sztringben. A sztring nullával van lezárva, az entert nem tartalmazza. A hívó felelőssége a free()-t meghívni a kapott pointerre.

Paraméterek

fp	Fájl pointer

Visszatérési érték

char* Beolvasott string

4.6.2.26. SortBeolvasStd()

```
char * SortBeolvasStd ( )
```

A függvény beolvas egy teljes sort (enterig vagy fájl vége jelig) , és visszaadja egy dinamikusan foglalt sztringben. A sztring nullával van lezárva, az entert nem tartalmazza. A hívó felelőssége a free()-t meghívni a kapott pointerre.

Visszatérési érték

char* Beolvasott string

4.6.2.27. Split()

```
StringList * Split (
            char * string,
            char split )
```

Az elválasztó karakter alapján több elemre szed szét egy stringet. A hívó felelőssége meghívni a DestroyStringList()-et.

Paraméterek

string	Felbontandó string
split	Karakter ami alapján felbontja

Visszatérési érték

StringList* Felbontott string lista

4.6.2.28. SplitInt()

```
List * SplitInt (
            char * string,
            char split )
```

Egy stringben megadott integereket szétszed az elválasztó karakter alapján. A hívó felelőssée meghívni a DestroyList()-et.

Paraméterek

string	Felbontandó string
split	Karakter ami alapján felbontja

Visszatérési érték

List* Felbontott int lista

4.7. essentials.h fájlreferencia

```
#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>
```

Adatszerkezetek

- struct Node

  Egy lista elem.
- struct List

  Int lista.
- struct StringNode

  String lista elem.
- struct StringList

  String lista.

Típusdefiníciók

- typedef struct Node Node

  Egy lista elem.
- typedef struct List List

  Int lista.
- typedef struct StringNode StringNode

  String lista elem.
- typedef struct StringList StringList

  String lista.

Függvények

- List ∗ CreateList (int startValue)

  Létrehoz egy int listát egy kezdőértékkel. A hívó felelőssége meghívni a listára a DestroyList() függvényt.
- StringList ∗ CreateStringList (char ∗start)

  Létrehoz egy String listát egy kezdőértékkel. A hívó felelőssége meghívni a listára a DestroyStringList() függvényt.
- void DestroyList (List ∗list)

  Felszabadítja a teljes int listát.
- void DestroyStringList (StringList ∗list)

  Felszabadítja a teljes String listát.
- void Add (List ∗list, int value)

  Hozzáad egy elemet egy int listához.
- void AddString (StringList ∗list, char ∗item)

  Hozzáad egy stringet elemet egy string listához.
- void AddStringPointer (StringList ∗list, char ∗item)

  Hozzáfűz egy String listához egy elemet aminek már le van foglalva a memória (csupán a kód gyorsításáért kell)
- int At (List ∗list, int index)

  Visszaadja a tömb bizonyos indexénél található értéket.
- char ∗ AtString (StringList ∗list, int index)

  Visszaadja a tömb bizonyos indexénél található értéket.
- void Set (List ∗list, int index, int value)

  Beállítja a tömb bizonyos indexénél található értéket.
- void SetString (StringList ∗list, int index, char ∗value)

  Beállítja a tömb bizonyos indexénél található értéket.
- void RemoveFirst (List ∗list)

  Kitörli a lista első elemét.
- void RemoveFirstString (StringList ∗list)

  Kitörli a lista első elemét.
- void RemoveAt (StringList ∗list, int index)

  Kitörli a lista megadott elemét.
- void RemoveAtInt (List ∗list, int index)

  Kitörli a lista megadott elemét.
- int Find (List ∗list, int value)

  Megkeresi, hogy az érték hanyadik indexen található meg először. -1 ha az érték nem található
- int FindString (StringList ∗list, char ∗value)

  Megkeresi, hogy az érték hanyadik indexen található meg először. -1 ha az érték nem található
- bool Contains (List ∗list, int value)

  Megnézi, hogy tartalmazza-e a lista az elemet.
- bool ContainsString (StringList ∗list, void ∗value)

Megnézi, hogy tartalmazza-e a lista az elemet.

- StringList * Split (char *string, char split)

  Az elválasztó karakter alapján több elemre szed szét egy stringet. A hívó felelőssége meghívni a DestroyStringList()-et.

- List * SplitInt (char *string, char split)

  Egy stringben megadott integereket szétszed az elválasztó karakter alapján. A hívó felelőssée meghívni a DestroyList()-et.

- char * Alloc (int n, FILE *fp)

  SortBeolvas segédfüggvénye.

- char * SortBeolvas (FILE *fp)

  A függvény beolvas egy teljes sort (enterig vagy fájl vége jelig) a megadott fájlból, és visszaadja egy dinamikusan foglalt sztringben. A sztring nullával van lezárva, az entert nem tartalmazza. A hívó felelőssége a free()-t meghívni a kapott pointerre.

- int ConvertToInt (char *str)

  Egy stringet int-é alakít.

- char * CpyStrPtr (char *str)

  Készít egy másolatot egy stringből. A hívó felelőssége meghívni a free-t.

- char * Merge (StringList *list, int start, int end)

  Lista elemeit összeolvasztja egy stringbe, whitespace karakterrel elválasztva. A hívó felelőssége meghívni a free-t.

- char * SortBeolvasStd ()

  A függvény beolvas egy teljes sort (enterig vagy fájl vége jelig) , és visszaadja egy dinamikusan foglalt sztringben. A sztring nullával van lezárva, az entert nem tartalmazza. A hívó felelőssége a free()-t meghívni a kapott pointerre.

4.7.1. Típusdefiníciók dokumentációja

4.7.1.1. List

```
typedef struct List List
```

Int lista.

4.7.1.2. Node

```
typedef struct Node Node
```

Egy lista elem.

4.7.1.3. StringList

```
typedef struct StringList StringList
```

String lista.

4.7.1.4. StringNode

```
typedef struct StringNode StringNode
```

String lista elem.

4.7.2. Függvények dokumentációja

4.7.2.1. Add()

```
void Add (
            List * list,
            int value )
```

Hozzáad egy elemet egy int listához.

Paraméterek

list	Lista
value	Érték amit hozzáad a listához

4.7.2.2. AddString()

```
void AddString (
            StringList * list,
            char * item )
```

Hozzáad egy stringet elemet egy string listához.

Paraméterek

list	Lista
value	Érték amit hozzáad a listához

4.7.2.3. AddStringPointer()

```
void AddStringPointer (
            StringList * list,
            char * item )
```

Hozzáfűz egy String listához egy elemet aminek már le van foglalva a memória (csupán a kód gyorsításáért kell)

Paraméterek

list	Lista
item	Pointer amit hozzá fog fűzni a listához

4.7.2.4. Alloc()

```
char * Alloc (
            int n,
            FILE * fp )
```

SortBeolvas segédfüggvénye.

4.7.2.5. At()

```
int At (
            List * list,
            int index )
```

Visszaadja a tömb bizonyos indexénél található értéket.

Paraméterek

list	Lista
index	index

Visszatérési érték

int Indexen található érték

4.7.2.6. AtString()

```
char * AtString (
            StringList * list,
            int index )
```

Visszaadja a tömb bizonyos indexénél található értéket.

Paraméterek

list	Lista
index	index

Visszatérési érték

char∗ Indexen található érték

4.7.2.7. Contains()

```
bool Contains (
            List * list,
            int value )
```

Megnézi, hogy tartalmazza-e a lista az elemet.

Paraméterek

list	Lista
value	Érték

Visszatérési érték

true Az elem benne van a listában

false Az elem nincs benne a listában

4.7.2.8. ContainsString()

```
bool ContainsString (
            StringList * list,
            void * value )
```

Megnézi, hogy tartalmazza-e a lista az elemet.

Paraméterek

list	Lista
value	Érték

Visszatérési érték

true Az elem benne van a listában

false Az elem nincs benne a listában

4.7.2.9. ConvertToInt()

```
int ConvertToInt (
            char * str )
```

Egy stringet int-é alakít.

Paraméterek

str	Átalakítandó string

Visszatérési érték

int Átalakított int

4.7.2.10. CpyStrPtr()

```
char * CpyStrPtr (
            char * str )
```

Készít egy másolatot egy stringből. A hívó felelőssége meghívni a free-t.

Paraméterek

str	Másolandó string

Visszatérési érték

char∗ Másolt string

4.7.2.11. CreateList()

```
List * CreateList (
            int startValue )
```

Létrehoz egy int listát egy kezdőértékkel. A hívó felelőssége meghívni a listára a DestroyList() függvényt.

Paraméterek

startValue	Kezdőérték

Visszatérési érték

List∗ Létrehozott lista

4.7.2.12. CreateStringList()

```
StringList * CreateStringList (
            char * start )
```

Létrehoz egy String listát egy kezdőértékkel. A hívó felelőssége meghívni a listára a DestroyStringList() függvényt.

Paraméterek

start	kezdő érték

Visszatérési érték

StringList∗ Létrehozott lista

4.7.2.13. DestroyList()

```
void DestroyList (
            List * list )
```

Felszabadítja a teljes int listát.

Paraméterek

list	Felszabadítandó lista

4.7.2.14. DestroyStringList()

```
void DestroyStringList (
            StringList * list )
```

Felszabadítja a teljes String listát.

Paraméterek

list	Felszabadítandó lista

4.7.2.15. Find()

```
int Find (
              List * list,
              int value )
```

Megkeresi, hogy az érték hanyadik indexen található meg először. -1 ha az érték nem található

Paraméterek

list	Lista
value	Keresendő érték

Visszatérési érték

int index

4.7.2.16. FindString()

```
int FindString (
              StringList * list,
              char * value )
```

Megkeresi, hogy az érték hanyadik indexen található meg először. -1 ha az érték nem található

Paraméterek

list	Lista
value	Keresendő érték

Visszatérési érték

int index

4.7.2.17. Merge()

```
char * Merge (
              StringList * list,
              int start,
              int end )
```

Lista elemeit összeolvasztja egy stringbe, whitespace karakterrel elválasztva. A hívó felelőssége meghívni a free-t.

Paraméterek

list	Lista
start	Kezdő index (inclusive)
end	Végső index (inclusive)

Visszatérési érték

char* Összeolvasztott string

4.7.2.18. RemoveAt()

```
void RemoveAt (
            StringList * list,
            int index )
```

Kitörli a lista megadott elemét.

Paraméterek

list	Lista
index	Index

4.7.2.19. RemoveAtInt()

```
void RemoveAtInt (
            List * list,
            int index )
```

Kitörli a lista megadott elemét.

Paraméterek

list	Lista
index	Index

4.7.2.20. RemoveFirst()

```
void RemoveFirst (
            List * list )
```

Kitörli a lista első elemét.

Paraméterek

list	Lista

4.7.2.21. RemoveFirstString()

```
void RemoveFirstString (
            StringList * list )
```

Kitörli a lista első elemét.

Paraméterek

list	Lista

4.7.2.22. Set()

```
void Set (
            List * list,
            int index,
            int value )
```

Beállítja a tömb bizonyos indexénél található értéket.

Paraméterek

list	Lista
index	Index
value	Beállítandó érték

4.7.2.23. SetString()

```
void SetString (
            StringList * list,
            int index,
            char * value )
```

Beállítja a tömb bizonyos indexénél található értéket.

Paraméterek

list	Lista
index	Index
value	Beállítandó érték

4.7.2.24. SortBeolvas()

```
char * SortBeolvas (
            FILE * fp )
```

A függvény beolvas egy teljes sort (enterig vagy fájl vége jelig) a megadott fájlból, és visszaadja egy dinamikusan foglalt sztringben. A sztring nullával van lezárva, az entert nem tartalmazza. A hívó felelőssége a free()-t meghívni a kapott pointerre.

Paraméterek

fp	Fájl pointer

Visszatérési érték

char∗ Beolvasott string

4.7.2.25. SortBeolvasStd()

```
char * SortBeolvasStd ( )
```

A függvény beolvas egy teljes sort (enterig vagy fájl vége jelig) , és visszaadja egy dinamikusan foglalt sztringben. A sztring nullával van lezárva, az entert nem tartalmazza. A hívó felelőssége a free()-t meghívni a kapott pointerre.

Visszatérési érték

char∗ Beolvasott string

4.7.2.26. Split()

```
StringList * Split (
            char * string,
            char split )
```

Az elválasztó karakter alapján több elemre szed szét egy stringet. A hívó felelőssége meghívni a DestroyStringList()-et.

Paraméterek

string	Felbontandó string
split	Karakter ami alapján felbontja

Visszatérési érték

StringList* Felbontott string lista

4.7.2.27. SplitInt()

```
List * SplitInt (
            char * string,
            char split )
```

Egy stringben megadott integereket szétszed az elválasztó karakter alapján. A hívó felelőssée meghívni a DestroyList()-et.

Paraméterek

string	Felbontandó string
split	Karakter ami alapján felbontja

Visszatérési érték

List* Felbontott int lista

4.8. essentials.h

Ugrás a fájl dokumentációjához.

```
1 #ifndef ESSENTIALS_H
2 #define ESSENTIALS_H
3 #include <stdlib.h>
4 #include <stdbool.h>
5 #include <stdio.h>
6
11 typedef struct Node {
12     struct Node *nextNode;
13     int value;
14 } Node;
15
20 typedef struct List
21 {
22     Node *startNode; //Ha minden elromlana akkor ezt írd vissza void *-ra
23     int len;
24 } List;
25
30 typedef struct StringNode {
31     struct StringNode *nextNode;
32     char *value;
33 } StringNode;
34
39 typedef struct StringList
40 {
41     StringNode *startNode;
42     int len;
43 } StringList;
44
45 List *CreateList(int startValue);
46 StringList *CreateStringList(char *start);
47
48 void DestroyList(List *list);
49 void DestroyStringList(StringList *list);
50
51 void Add(List *list, int value);
52 void AddString(StringList *list, char *item);
53 void AddStringPointer(StringList *list, char *item);
54
```

```
55 int At(List *list, int index);
56 char *AtString(StringList *list, int index);
57
58 void Set(List *list, int index, int value);
59 void SetString(StringList *list, int index, char *value);
60
61 void RemoveFirst(List *list);
62 void RemoveFirstString(StringList *list);
63 void RemoveAt(StringList *list, int index);
64 void RemoveAtInt(List *list, int index);
65
66 int Find(List *list, int value);
67 int FindString(StringList *list, char* value);
68
69 bool Contains(List *list, int value);
70 bool ContainsString(StringList *list, void *value);
71
72 StringList *Split(char *string, char split);
73 List *SplitInt(char *string, char split);
74
75 char *Alloc(int n, FILE *fp);
76 char *SortBeolvas(FILE *fp);
77
78 int ConvertToInt(char *str);
79 char *CpyStrPtr(char *str);
80 char *Merge(StringList *list, int start, int end);
81
82 char *SortBeolvasStd();
83 #endif
```

4.9. gameSpace.c fájlreferencia

Játék állapotát kezeli és a játéktérben elhelyezkedő elemeket definiálja.

```
#include "gameSpace.h"
#include "essentials.h"
#include <string.h>
```

Függvények

- void ChangeStates (State states[ ], int stateCount, char ∗list)

  Megváltoztatja egy event string alapján az object/room állapotát.

4.9.1. Részletes leírás

Játék állapotát kezeli és a játéktérben elhelyezkedő elemeket definiálja.

4.9.2. Függvények dokumentációja

4.9.2.1. ChangeStates()

```
void ChangeStates (
            State states[],
            int stateCount,
            char * list )
```

Megváltoztatja egy event string alapján az object/room állapotát.

Paraméterek

states	Állapotok tömb
stateCount	Állapotok tömb elemszáma
list	Event lista

4.10. gameSpace.h fájlreferencia

```
#include "essentials.h"
#include <stdbool.h>
```

Adatszerkezetek

- struct Room

  Szobát definiáló struct.
- struct Object

  Objectet definiáló struct.
- struct State

  Állapotot definiáló struct.

Típusdefiníciók

- typedef struct Room Room

  Szobát definiáló struct.
- typedef struct Object Object

  Objectet definiáló struct.
- typedef struct State State

  Állapotot definiáló struct.

Függvények

- void ChangeStates (State states[ ], int stateCount, char ∗list)

  Megváltoztatja egy event string alapján az object/room állapotát.

4.10.1. Típusdefiníciók dokumentációja

4.10.1.1. Object

```
typedef struct Object Object
```

Objectet definiáló struct.

4.10.1.2. Room

```
typedef struct Room Room
```

Szobát definiáló struct.

4.10.1.3. State

```
typedef struct State State
```

Állapotot definiáló struct.

4.10.2. Függvények dokumentációja

4.10.2.1. ChangeStates()

```
void ChangeStates (
            State states[],
            int stateCount,
            char * list )
```

Megváltoztatja egy event string alapján az object/room állapotát.

Paraméterek

states	Állapotok tömb
stateCount	Állapotok tömb elemszáma
list	Event lista

4.11. gameSpace.h

Ugrás a fájl dokumentációjához.

```
1 #ifndef GAMESPACE_H
2 #define GAMESPACE_H
3 #include "essentials.h"
4 #include <stdbool.h>
5
10 typedef struct Room
11 {
12     int id;
13     int state;
14     char *name;
15     char *lookAroundText;
16     char *entryText;
17     List *connectedItems;
18     List *connectedRooms;
19 } Room;
20
```

```
25 typedef struct Object {
26     int id;
27     int state;
28     char *name;
29     char *lookAtText;
30     bool collectible;
31     StringList *modifyEvent;
32     StringList *modify;
33     StringList *modifyString;
34 } Object;
35
40 typedef struct State {
41     bool room; //A false azt jelenti, hogy object
42     int id;
43     int state;
44 } State;
45
46 void ChangeStates(State states[], int stateCount, char *list);
47 #endif
```

4.12. graphics.c fájlreferencia

Képernyőn lévő megjelenítésekért felelős.

```
#include "graphics.h"
#include "debugmalloc.h"
#include <stdio.h>
#include "econio.h"
#include <stdbool.h>
#include "gameSpace.h"
#include "essentials.h"
#include "interpreter.h"
#include "loaderSystem.h"
```

Adatszerkezetek

- struct MenuItem

  Egy menü element definiál.

Típusdefiníciók

- typedef struct MenuItem MenuItem

  Egy menü element definiál.

Függvények

- void DrawMenuItem (int x, int y, char *text, bool selected)

  Szöveg kirajzolása a képernyőre egy adott pozícióban.
- void DrawMenu (int selected)

  Összes menü elem kirajzolása, a selected elem kijelölésével.
- void DrawGameSpace ()

  Játéktér kirajzolása.
- void PlayGame (List *playerInventory)

  Játék elindítása.
- void NewGame ()

Új játék előkészítése.

- void LoadGame ()

  Előző játék betöltése.
- void HowToPlay ()

  How to play menüpont kezelése.
- void MenuMode ()

  Kezeli a program menü módját.

Változók

- const int WIDTH = 120
- const int HEIGHT = 30
- MenuItem items [4]
- Room ∗ rooms
- Object ∗ objects
- State ∗ states
- int roomCount
- int objectCount
- int stateCount
- int previousRoomId
- int currentRoomId
- int previousRoomState
- int currentRoomState

4.12.1. Részletes leírás

Képernyőn lévő megjelenítésekért felelős.

4.12.2. Típusdefiníciók dokumentációja

4.12.2.1. MenuItem

```
typedef struct MenuItem MenuItem
```

Egy menü element definiál.

4.12.3. Függvények dokumentációja

4.12.3.1. DrawGameSpace()

```
void DrawGameSpace ( )
```

Játéktér kirajzolása.

4.12.3.2. DrawMenu()

```
void DrawMenu (
            int selected )
```

Összes menü elem kirajzolása, a selected elem kijelölésével.

Paraméterek

selected	A kiválasztott elem sorszáma

4.12.3.3. DrawMenuItem()

```
void DrawMenuItem (
            int x,
            int y,
            char * text,
            bool selected )
```

Szöveg kirajzolása a képernyőre egy adott pozícióban.

Paraméterek

x	X pozíció
y	Y pozíció
text	szöveg
selected	kiválasztott elem sorszáma

4.12.3.4. HowToPlay()

```
void HowToPlay ( )
```

How to play menüpont kezelése.

4.12.3.5. LoadGame()

```
void LoadGame ( )
```

Előző játék betöltése.

4.12.3.6. MenuMode()

```
void MenuMode ( )
```

Kezeli a program menü módját.

4.12.3.7. NewGame()

```
void NewGame ( )
```

Új játék előkészítése.

4.12.3.8. PlayGame()

```
void PlayGame (
            List * playerInventory )
```

Játék elindítása.

Paraméterek

playerInventory	A játékos eszköztára

4.12.4. Változók dokumentációja

4.12.4.1. currentRoomId

```
int currentRoomId
```

4.12.4.2. currentRoomState

```
int currentRoomState
```

4.12.4.3. HEIGHT

```
const int HEIGHT = 30
```

4.12.4.4. items

```
MenuItem items[4]
```

4.12.4.5. objectCount

```
int objectCount
```

4.12.4.6. objects

```
Object* objects
```

4.12.4.7. previousRoomId

```
int previousRoomId
```

4.12.4.8. previousRoomState

```
int previousRoomState
```

4.12.4.9. roomCount

```
int roomCount
```

4.12.4.10. rooms

```
Room* rooms
```

4.12.4.11. stateCount

```
int stateCount
```

4.12.4.12. states

```
State* states
```

4.12.4.13. WIDTH

```
const int WIDTH = 120
```

4.13. graphics.h fájlreferencia

Függvények

- void MenuMode ()

  Kezeli a program menü módját.

4.13.1. Függvények dokumentációja

4.13.1.1. MenuMode()

```
void MenuMode ( )
```

Kezeli a program menü módját.

4.14. graphics.h

Ugrás a fájl dokumentációjához.

```
1 #ifndef GRAPHICS_H
2 #define GRAPHICS_H
3
4 void MenuMode();
5
6 #endif
```

4.15. interpreter.c fájlreferencia

A játékos által megadott parancs értelmezéséért felelős.

```
#include "debugmalloc.h"
#include "interpreter.h"
#include <stdlib.h>
#include "essentials.h"
#include <string.h>
#include "gameSpace.h"
#include <stdbool.h>
```

Függvények

- int WhatState (int id, State states[ ], int stateCount)

  Visszaadja, hogy egy room vagy object jelenleg milyen állapotban van.
- int HasEvent (StringList ∗l, char ∗string)

  Megmondja, hogy egy elem rendelkezik-e egy bizonyos eventel, ha igen akkor visszaadja, hogy hanyadik event, ha nem akkor -1-et ad vissza.
- char ∗ Execute (int ∗currentRoomId, char ∗command, Object objects[ ], int objectCount, Room rooms[ ], int roomCount, State states[ ], int stateCount, List ∗playerInventory)

  Értelmezi a játék által megadott parancsot. Ha szobát vált a játékos akkor megváltoztatja a jelenlegi szoba ID-ját, állapotokat átállít, játékos eszköztárat menedzsel.

4.15.1. Részletes leírás

A játékos által megadott parancs értelmezéséért felelős.

4.15.2. Függvények dokumentációja

4.15.2.1. Execute()

```
char * Execute (
            int * currentRoomId,
            char * command,
            Object objects[],
            int objectCount,
            Room rooms[],
            int roomCount,
            State states[],
            int stateCount,
            List * playerInventory )
```

Értelmezi a játék által megadott parancsot. Ha szobát vált a játékos akkor megváltoztatja a jelenlegi szoba ID-ját, állapotokat átállít, játékos eszköztárat menedzsel.

Paraméterek

currentRoomId	Jelenlegi szoba azonosítója
command	A játékos által megadott parancs
objects	Objecteket tartalmazó tömb
objectCount	Objecteket tartalmazó tömb elemszáma
rooms	Szobákat tartalmazó tömb
roomCount	Szobákat tartalmazó tömb elemszáma
states	Állapotokat tartalmazó tömb
stateCount	Állapotokat tartalmazó tömb elemszáma
playerInventory	Játékos eszköztára

Visszatérési érték

char∗ A parancs által kiváltott esemény szövege.

4.15.2.2. HasEvent()

```
int HasEvent (
            StringList * l,
            char * string )
```

Megmondja, hogy egy elem rendelkezik-e egy bizonyos eventel, ha igen akkor visszaadja, hogy hanyadik event, ha nem akkor -1-et ad vissza.

Paraméterek

l	A tesztelendő elem event listája
string	A tesztelendő event

Visszatérési érték

int Az event sorszáma. -1 ha nem létezik

4.15.2.3. WhatState()

```
int WhatState (
            int id,
            State states[],
            int stateCount )
```

Visszaadja, hogy egy room vagy object jelenleg milyen állapotban van.

Paraméterek

id	object id-ja
states	Állapotokat tartalmazó tömb
stateCount	Állapotokat tartalmazó tömb hossza

Visszatérési érték

int Az object állapota

4.16. interpreter.h fájlreferencia

```
#include <stdbool.h>
#include "gameSpace.h"
```

```
#include "essentials.h"
```

Függvények

- char * Execute (int *currentRoomId, char *command, Object objects[], int objectCount, Room rooms[], int roomCount, State states[], int stateCount, List *playerInventory)

  Értelmezi a játék által megadott parancsot. Ha szobát vált a játékos akkor megváltoztatja a jelenlegi szoba ID-ját, állapotokat átállít, játékos eszköztárat menedzsel.
- int WhatState (int id, State states[], int stateCount)

  Visszaadja, hogy egy room vagy object jelenleg milyen állapotban van.

4.16.1. Függvények dokumentációja

4.16.1.1. Execute()

```
char * Execute (
            int * currentRoomId,
            char * command,
            Object objects[],
            int objectCount,
            Room rooms[],
            int roomCount,
            State states[],
            int stateCount,
            List * playerInventory )
```

Értelmezi a játék által megadott parancsot. Ha szobát vált a játékos akkor megváltoztatja a jelenlegi szoba ID-ját, állapotokat átállít, játékos eszköztárat menedzsel.

Paraméterek

currentRoomId	Jelenlegi szoba azonosítója
command	A játékos által megadott parancs
objects	Objecteket tartalmazó tömb
objectCount	Objecteket tartalmazó tömb elemszáma
rooms	Szobákat tartalmazó tömb
roomCount	Szobákat tartalmazó tömb elemszáma
states	Állapotokat tartalmazó tömb
stateCount	Állapotokat tartalmazó tömb elemszáma
playerInventory	Játékos eszköztára

Visszatérési érték

char* A parancs által kiváltott esemény szövege.

4.16.1.2. WhatState()

```
int WhatState (
              int id,
              State states[],
              int stateCount )
```

Visszaadja, hogy egy room vagy object jelenleg milyen állapotban van.

Paraméterek

id	object id-ja
states	Állapotokat tartalmazó tömb
stateCount	Állapotokat tartalmazó tömb hossza

Visszatérési érték

int Az object állapota

4.17. interpreter.h

Ugrás a fájl dokumentációjához.

```
1 #ifndef INTERPRETER_H
2 #define INTERPRETER_H
3 #include <stdbool.h>
4 #include "gameSpace.h"
5 #include "essentials.h"
6
7 char *Execute(int *currentRoomId, char *command, Object objects[], int objectCount, Room rooms[], int
      roomCount, State states[], int stateCount, List *playerInventory);
8 int WhatState(int id, State states[], int stateCount);
9
10 #endif
```

4.18. loaderSystem.c fájlreferencia

Felelős az értelmezendő szövegek betöltéséért.

```
#include "debugmalloc.h"
#include "essentials.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "gameSpace.h"
```

Függvények

- StringList * ReadFile (char *file)

  Beolvas egy fájlt és egy listába tördeli szét sorok szerint. A hívó felelőssége meghívni a DestroyStringList() a listára.
- void DestroyRooms (Room *room, int roomCount)

  Felszabadítja az összes room objectet.
- void DestroyObjects (Object *object, int objectCount)

  felszabadítja az összes object objectet
- void LoadStry (Room **rooms, Object **objects, int *roomCount, int *objectCount, State *states[], int *elementCount)

  Betölti a rooms.stry-t és az objects.stry-t. A hívó köteles meghívni mint a DestroyObjects és a DestroyRooms függvényeket.

4.18.1. Részletes leírás

Felelős az értelmezendő szövegek betöltéséért.

4.18.2. Függvények dokumentációja

4.18.2.1. DestroyObjects()

```
void DestroyObjects (
            Object * object,
            int objectCount )
```

felszabadítja az összes object objectet

4.18.2.2. DestroyRooms()

```
void DestroyRooms (
            Room * room,
            int roomCount )
```

Felszabadítja az összes room objectet.

4.18.2.3. LoadStry()

```
void LoadStry (
            Room ** rooms,
            Object ** objects,
            int * roomCount,
            int * objectCount,
            State * states[],
            int * elementCount )
```

Betölti a rooms.stry-t és az objects.stry-t. A hívó köteles meghívni mint a DestroyObjects és a DestroyRooms függvényeket.

Paraméterek

rooms	
objects	
roomCount	
objectCount	
states	
elementCount	

4.18.2.4. ReadFile()

```
StringList * ReadFile (
             char * file )
```

Beolvas egy fájlt és egy listába törgeli szét sorok szerint. A hívó felelőssége meghívni a DestroyStringList() a listára.

Paraméterek

file	Fájl

Visszatérési érték

StringList* Beolvasott sorok

4.19. loaderSystem.h fájlreferencia

```
#include "gameSpace.h"
```

Függvények

- void DestroyRooms (Room *room, int roomCount)

  Felszabadítja az összes room objectet.
- void DestroyObjects (Object *object, int objectCount)

  felszabadítja az összes object objectet
- void LoadStry (Room **rooms, Object **objects, int *roomCount, int *objectCount, State *states[ ], int *elementCount)

  Betölti a rooms.stry-t és az objects.stry-t. A hívó köteles meghívni mint a DestroyObjects és a DestroyRooms függvényeket.

4.19.1. Függvények dokumentációja

4.19.1.1. DestroyObjects()

```
void DestroyObjects (
            Object * object,
            int objectCount )
```

felszabadítja az összes object objectet

4.19.1.2. DestroyRooms()

```
void DestroyRooms (
            Room * room,
            int roomCount )
```

Felszabadítja az összes room objectet.

4.19.1.3. LoadStry()

```
void LoadStry (
            Room ** rooms,
            Object ** objects,
            int * roomCount,
            int * objectCount,
            State * states[],
            int * elementCount )
```

Betölti a rooms.stry-t és az objects.stry-t. A hívó köteles meghívni mint a DestroyObjects és a DestroyRooms függvényeket.

Paraméterek

rooms	
objects	
roomCount	
objectCount	
states	
elementCount	

4.20. loaderSystem.h

Ugrás a fájl dokumentációjához.

```
1 #ifndef LOADERSYSTEM_H
2 #define LOADERSYSTEM_H
3 #include "gameSpace.h"
4
5 void DestroyRooms(Room *room, int roomCount);
6 void DestroyObjects(Object *object, int objectCount);
```

```
7 void LoadStry(Room **rooms, Object **objects, int *roomCount, int *objectCount, State *states[], int
      *elementCount);
8
9 #endif
```

4.21. main.c fájlreferencia

```
#include "debugmalloc.h"
#include <stdio.h>
#include <stdlib.h>
#include "essentials.h"
#include "loaderSystem.h"
#include "gameSpace.h"
#include "interpreter.h"
#include "graphics.h"
#include <windows.h>
```

Függvények

- int main (int argc, char const ∗argv[ ])

4.21.1. Függvények dokumentációja

4.21.1.1. main()

```
int main (
            int argc,
            char const * argv[] )
```

Visszatérési érték

0 Sikeres működés

1 Memória foglalás hiba

2 Fájlkezelés hiba

3 Túlindexelés hiba

Tárgymutató