Tartalomjegyzék

1. beker1.c

```c
#include <stdio.h>
#include <string.h>

int main()
{
    puts("Adj␣meg␣szamokat␣0␣vegjelig!");
    puts("");

    while (1)
    {
        int szam;

        printf("Szam:␣");
        scanf("%d", &szam);

        if (szam == 0) {
            break;
        }
    }

    return 0;
}
```

2. beker2.c

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    puts("Adj␣meg␣szamokat␣0␣vegjelig!");
    puts("");

    int *szamok = NULL;
    int elemszam = 0;

    while (1)
    {
        int szam;

        printf("Szam:␣");
        scanf("%d", &szam);

        if (szam == 0) {
            break;
        }
        // else
        szamok = realloc(szamok, (elemszam + 1) * sizeof(int));
        szamok[elemszam] = szam;
        ++elemszam;
    }

    for (int i = 0; i < elemszam; ++i)
    {
        printf("%d␣", szamok[i]);
    }
    puts("");

    free(szamok);

    return 0;
}
```

3. buggy.c

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

int main()
{
    int *a;
    int *b;

    a = malloc(sizeof(int));

    *a = 20;
    *b = 13;     // Sulyos hiba!

    return 0;
}
```

4. dyn_array_v1.c

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

typedef struct {
    int *elems;     // a dinamikusan lefoglalt tombre mutat
    int length;     // a betett elemek szama
    int capacity;   // a tomb fizikai kapacitasa
                    // capacity >= length
} DynArray;

void mem_error_exit()
{
    fprintf(stderr, "Error:␣cannot␣allocate␣memory\n");
    exit(1);
}

DynArray * da_create()
{
    DynArray *result = malloc(sizeof(DynArray));
    if (result == NULL) {
        mem_error_exit();
    }

    int initial_capacity = 2;

    result->elems = malloc(initial_capacity * sizeof(int));
    if (result->elems == NULL) {
        mem_error_exit();
    }
    result->length = 0;
    result->capacity = initial_capacity;

    return result;
}

void da_append(DynArray *self, int data)
{
    if (self->length == self->capacity)
    {
        int new_capacity = 2 * self->capacity;
        self->elems = realloc(self->elems, new_capacity * sizeof(int
           ));
        if (self->elems == NULL) {
            mem_error_exit();
        }
        self->capacity = new_capacity;
    }
    //
    self->elems[self->length] = data;
    self->length += 1;
}

void * da_destroy(DynArray *self)
{
    free(self->elems);
    free(self);
    //
    return NULL;
}

int main()
{
    DynArray *li = da_create();

    // li.append(1);
    da_append(li, 1);
    da_append(li, 2);
    da_append(li, 3);
    for (int i = 4; i <= 20; ++i) {
        da_append(li, i);
    }

    for (int i = 0; i < li->length; ++i) {
        printf("%d␣", li->elems[i]);
    }
    puts("");

    li = da_destroy(li);

    printf("%p\n", li);

    return 0;
}
```

5. dyn_array_v2.c

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

#define INITIAL_CAPACITY 2
#define MULTIPLIER 1.5

typedef struct {
    int *elems;     // a dinamikusan lefoglalt tombre mutat
    int length;     // a betett elemek szama
    int capacity;   // a tomb fizikai kapacitasa
                    // capacity >= length
} DynArray;

void mem_error_exit()
{
    fprintf(stderr, "Error:␣cannot␣allocate␣memory\n");
    exit(1);
}

DynArray * da_create()
{
    DynArray *result = malloc(sizeof(DynArray));
    if (result == NULL) {
        mem_error_exit();
    }

    result->elems = malloc(INITIAL_CAPACITY * sizeof(int));
    if (result->elems == NULL) {
        mem_error_exit();
    }
    result->length = 0;
    result->capacity = INITIAL_CAPACITY;

    return result;
}

void da_append(DynArray *self, int data)
{
    if (self->length == self->capacity)
    {
        int new_capacity = (int)(MULTIPLIER * self->capacity);
        self->elems = realloc(self->elems, new_capacity * sizeof(int
           ));
        puts("#␣ujraallokalas␣tortent");
        if (self->elems == NULL) {
            mem_error_exit();
        }
        self->capacity = new_capacity;
    }
    //
    self->elems[self->length] = data;
    self->length += 1;
}

void * da_destroy(DynArray *self)
{
    free(self->elems);
    free(self);
    //
    return NULL;
}

int main()
{
    DynArray *li = da_create();

    // li.append(1);
    da_append(li, 1);
    da_append(li, 2);
    da_append(li, 3);
    for (int i = 4; i <= 20; ++i) {
        da_append(li, i);
    }

    for (int i = 0; i < li->length; ++i) {
        printf("%d␣", li->elems[i]);
    }
    puts("");

    li = da_destroy(li);

    printf("%p\n", li);

    return 0;
}
```

6. prog1.c

```
#include "prog1.h"

//
   ////////////////////////////////////////////////////////////////////////////////

//
//   Implementation
//

#include <stdio.h>
#include <string.h>

#define BUFSIZE 1024

/**
 * Ez egy dinamikusan lefoglalt sztringet ad vissza,
 * amit a hivo oldalon majd valamikor fel kell szabaditani.
 */
string get_string(const char* prompt)
{
    char buf[BUFSIZE];

    printf("%s", prompt);
    fgets(buf, sizeof(buf), stdin);
    buf[strlen(buf) - 1] = '\0';

    return strdup(buf);
}
```

7. prog1.h

```
#ifndef PROG1_H
#define PROG1_H

/**
 * Our own type for (pointers to) strings.
 */
typedef char * string;

//
   ////////////////////////////////////////////////////////////////////////////////

//
//   Public Interface
//

string get_string(const char* prompt);

#endif    // PROG1_H
```

8. tombok1.c

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

int main()
{
    int *list = malloc(3 * sizeof(int));
    if (list == NULL) {
        exit(1);
    }
    list[0] = 1;
    list[1] = 2;
    list[2] = 3;

    int *tmp = malloc(4 * sizeof(int));
    if (tmp == NULL) {
        exit(1);
    }

    for (int i = 0; i < 3; ++i)
    {
        tmp[i] = list[i];
    }
    tmp[3] = 4;

    free(list);
    list = tmp;

    for (int i = 0; i < 4; ++i)
    {
        printf("%d\n", list[i]);
    }

    return 0;
}
```

9. tombok2.c

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

int main()
{
    int *list = malloc(3 * sizeof(int));
    if (list == NULL) {
        exit(1);
    }
    list[0] = 1;
    list[1] = 2;
    list[2] = 3;

    int *tmp = realloc(list, 4 * sizeof(int));

    list = tmp;
    list[3] = 4;

    for (int i = 0; i < 4; ++i)
    {
        printf("%d\n", list[i]);
    }

    return 0;
}
```

10. tombok3.c

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

int main()
{
    int *list = malloc(3 * sizeof(int));
    if (list == NULL) {
        exit(1);
    }
    list[0] = 1;
    list[1] = 2;
    list[2] = 3;

    list = realloc(list, 4 * sizeof(int));

    list[3] = 4;

    for (int i = 0; i < 4; ++i)
    {
        printf("%d\n", list[i]);
    }

    free(list);

    return 0;
}
```