

Dynamic Allocation

The free Function

```
void free(void *ptr);
```

- **free(p)** deallocates memory pointed by **p**.
- Freeing a non-allocated area is an **error**.
- No partial deallocation.
- Always free allocated memory.



Returning Allocated Memory

Example: duplicate a string to a new location (with dynamic allocation), return the new string:

```
// strdup - duplicate a string
char* strdup(const char* str)
{
    char* dup =
        (char*)malloc((strlen(str)+1) * sizeof(char));
    if (dup != NULL) {
        strcpy(dup, str);
    }
    return dup;
}
```

Malloc - Reminder

```
void* malloc(unsigned int nBytes);
```

- Used to dynamically allocate **nBytes** in memory
- Returns a pointer to the allocated area on success, **NULL** on failure
- **Always** check whether memory was successfully allocated

Warming Up

```
int main()
{
    int i=0, n=0, *p=NULL;

    printf("How many numbers will you enter?\n");
    scanf("%d", &n);
    casting p = (int*)malloc(n * sizeof(int));
    if (p == NULL)
    {
        printf("Memory allocation failed!\n");
        return 1;
    }
    ...
    /* Free the allocated space */
    free(p);

    return 0;
}
```

Exercise

- Implement the function

```
char* my_strcat(const char *s1, const char *s2);
```

Output: a pointer to a dynamically allocated concatenation of s1 and s2.

Example:
my_strcat("hello_", "world!");
Output: "hello_world!"

You can use the functions in string.h

- Test your function

What's Wrong Here?

```
char* my_strcat(const char *s1, const char
*s2)
{
    char result[500]; /* assume this is
enough */

    strcpy(result, s1);
    strcpy(result + strlen(s1), s2);

    return result;
}
```

Pitfalls

- Accessing un-initialized pointer

```
int* p;
*p = 3;
```

- Using de-allocated memory

```
int* p = (int*)malloc(SIZE * sizeof(int));
... /* do something with p*/
free(p);
*p = 0;
```

Dynamic Array

- Imagine that you own a rapidly growing business that currently has 50 customers
- Every customer has a struct containing her details:

```
typedef struct {
    char name[MAX_LEN];
    ...
} Customer;
```

Dynamic Array

The pool is kept in an array of customers:
Customer customers_arr[SIZE];

But What is SIZE?

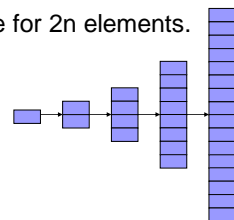
```
#define SIZE 50 ? Not enough
#define SIZE 50000 ? What a waste
```

Dynamic Array

- **Solution:** current size ~ 50, grow on demand.
- **Take 1:** On insertion - if the array is full re-allocate space for n+1 elements
- **Problem:** copying n elements per insertion is expensive.

Dynamic Array

- **Take 2:**
if the array is full:
Re-allocate space for 2n elements.



Dynamic Array - Code

```
typedef struct {
    Customer * arr;
    int len;
    int capacity;
} Darray;

Darray init_array(int capacity)
{
    Darray darr;

    darr.arr = (Customer *)malloc(capacity * sizeof(Customer));
    if (darr.arr == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    darr.capacity = capacity;
    darr.len = 0;
    return darr;
}
```

Dynamic Array - Code

```
void insert(Darray * darr, Customer * customer)
{
    if (darr->len == darr->capacity)
    {
        darr->arr = (Customer *)realloc(darr->arr, darr->capacity*2*sizeof(Customer));
        if (darr->arr == NULL)
        {
            printf("Memory allocation failed\n");
            exit(1);
        }
        darr->capacity *= 2;
    }
    darr->arr[darr->len] = *customer;
    darr->len++;
}
```

Dynamic Array - Code

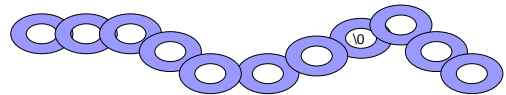
```
int main()
{
    Darray dynamic_array = init_array(ARR_INIT_SIZE);
    int i;
    Customer customer;

    // insert customers
    for (i = 0; i < 10; ++i)
    {
        printf("insert customer %d:\n", i+1);
        readString(customer.name, STRING_MAX_SIZE);
        insert(&dynamic_array, customer);
    }
    // print array
    for (i = 0; i < 10; ++i)
        printf("Customer %d: %s\n", i+1, dynamic_array.arr[i].name);

    free(dynamic_array.arr);
    return 0;
}
```

Dynamic String

- Conventionally, we represent a string as a char array that marks the end of the string with the special char '\0'.



Dynamic String - Definition

We will now consider a different representation:

```
typedef struct
{
    char *data;
    int capacity, length;
} string;
```

data points to a dynamically allocated char array.

capacity is length of the char array

len is length of the actual string



Dynamic String

Pro:

- The string's length can dynamically change
- Safe code – we can easily check that the array is long enough when expanding the string
- No need to pass the array length separately

Cons:

Functions defined in string.h won't work

Dynamic String - Allocation

Initialize:

```
string *allocateString()
{
    string *result = (string*) malloc(sizeof(string));
    if (result != NULL) ← Ensure allocation
    {
        result->data = NULL;
        result->capacity = 0;
        result->length = 0;
    }
    return result;
}
```

Dynamic String – Ensure Capacity

Re-allocate on demand

```
void ensureCapacity(string *str, int minCapacity)
{
    if (str->capacity >= minCapacity)
        return;

    str->data = (char*) realloc(str->data, minCapacity);
    if (str->data == NULL) ← Ensure re-allocation
    {
        printf("Fatal error: memeory allocation failed!\n");
        exit(1);
    }
    str->capacity = minCapacity;
}
```

Dynamic String – Free String

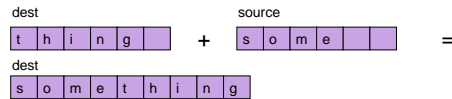
"Deep" free:

```
void freeString(string *str)
{
    if (str != NULL)
    {
        if (str->data != NULL)
            free(str->data);
        free(str);
    }
}
```

Dynamic String - Concatenate

```
Concatenate dest + source into dest

string* append(string *dest, const string *source)
{
    ensureCapacity(dest, dest->length + source->length);
    memcpy(dest->data + dest->length, source->data, source->length);
    dest->length += source->length; ← memcpy instead of strcpy
    return dest;
}
```



Dynamic String – Convert

Are both representations equivalent?

Can we convert any C string (null-terminated) to the new string struct?

- Yes, easily (see next slide).

What about converting a string struct to a C string?

Dynamic String – Convert

```
string *convertCString(const char *cstr)
{
    string *result = NULL;
    int cstrlen = 0;

    result = allocateString();
    if (result != NULL) {
        cstrlen = strlen(cstr);
        ensureCapacity(result, cstrlen);
        memcpy(result->data, cstr, cstrlen);
        result->length = cstrlen;
    }
    return result;
}
```

שאלה ממבחן - תש"ע, סמסטר א', שאלה 2

בשאלה זו תתבקשו לכתוב פונקציות ותוכנת מחשב לניהול משחק הלוטו.

סעיף א':

"מערך משורשר" - מערך המכיל ערכים של מספר תתי מערכים באותו אורך באופן רציף בזה אחר זה. לדוגמא, "מערך משורשר" המכיל 3 תתי מערכים, כל אחד בגודל 6:



שאלה ממבחן - תש"ע, סמסטר א', שאלה 2

כתבו פונקציה המקבלת:

- מערך של מספרים שלמים שהוא "מערך משורשר"
 - גודל כל תתי-מערך n.
 - 2 מספרים שלמים, i ו-j.
- הפונקציה תחזיר את arr[[j][i]].

```
int getVal(int arr[], int n, int i, int j)
{
    return arr[i * n + j];
}
```

שאלה ממבחן - תש"ע, סמסטר א', שאלה 2

סעיף ב':

מגדלים 6 מספרים בין 1 ל-49. טופס מנצח מכיל את כל 6 המספרים. ממשו פונקציה הבודקת מי הם הזוכים, המקבלת: winNums - מערך בגודל 6 המכיל את המספרים הזוכים. Tickets - מערך משורשר עם תתי מערכים באורך 6 ובהם טפסים של משתתפים. numTickets - מספר הטפסים במערך. res - מערך בגודל מספר המשתתפים. בסיום הריצה, התא ה-i ב-res יכול 1 אם הטופס ה-i זכה בפרס ו-0 אחרת. ערך מוחזר: מספר הטפסים הזוכים.

שאלה ממבחן - תש"ע, סמסטר א', שאלה 2

כדי לקבל את מלוא הנקודות בסעיף זה יש לעבור רק פעם אחת על המערך winNums ורק פעם אחת על המערך tickets.



שאלה ממבחן - תש"ע, סמסטר א', שאלה 2

```
#define TICKET_SIZE 6
#define NUM_RANGE 49
```

```
int checkWin(int winNums[], int tickets[], int numTickets, int res[])
```

```
{
    int i, j, val, numWinners = 0;
    char numbers[NUM_RANGE]={};
    // init auxiliary array
    for (i = 0; i < TICKET_SIZE; i++)
        numbers[winNums[i]] = 1;
```

Auxiliary array

שאלה ממבחן - תש"ע, סמסטר א', שאלה 2

```
// go over all tickets
for (i = 0; i < numTickets; i++) {
    res[i] = 1;
    // go over 6 numbers in a ticket
    for (j = 0; j < TICKET_SIZE; j++) {
        val = getVal(tickets, TICKET_SIZE, i, j);
        if (numbers[val] == 0) {
            // found a non-winning number - ticket lost
            res[i] = 0;
            break;
        }
    }
    // if we are here - all 6 numbers won
    numWinners += res[i];
}
return numWinners;
}
```

שאלה ממבחן - תש"ע, סמסטר א', שאלה 2

סעיף ג' (15 נקודות):

הפרס הראשון בלוטו הוא 10,000,000 ש"ח.
אם כמה משתתפים זכו הם יתחלקו בפרס באופן שווה (ייתכן שאין זוכה).

כתבו תוכנית הקולטת מהמשתמש את הנתונים הבאים לפי הסדר המתואר:

1. ששת המספרים הזוכים
 2. מספר המשתתפים
 3. טפסי הניחוש בדה אחר זה
- פלט התוכנית הוא טבלה שבכל שורה בה מופיע מספר סידורי של משתתף וסכום הזכייה שלו.

שאלה ממבחן - תש"ע, סמסטר א', שאלה 2

```
int main()
{
    int i, numParticipants, numWinners = 0;
    int winningNumbers[TICKET_SIZE];
    int *tickets, *winners;
    double jackpot = 10000000;

    printf("Enter winning numbers: ");
    for (i = 0; i < TICKET_SIZE; i++)
        scanf("%d", &winningNumbers[i]);

    printf("Enter number of participants: ");
    scanf("%d", &numParticipants);
    // dynamic allocation
    tickets = (int*) malloc(numParticipants * TICKET_SIZE * sizeof(int));
    winners = (int*) malloc(numParticipants * sizeof(int));
```

שאלה ממבחן - תש"ע, סמסטר א', שאלה 2

```
if (tickets == NULL || winners == NULL)
{
    printf("Unable to allocate memory!");
    return 1;
}
printf("Enter lottery tickets: ");
...
numWinners = checkWin(winningNumbers, tickets, numParticipants, winners);
...
// free allocated memory
free(tickets);
free(winners);

return 0;
}
```

Solution to Class exercise

```
char* my_strcat(const char *s1, const char *s2)
{
    int len;
    char *result = NULL;

    len = strlen(s1) + strlen(s2) + 1;

    result = (char*)malloc(len * sizeof(char));
    if (result == NULL) {
        printf("Memory allocation failed!\n");
        return NULL;
    }

    strcpy(result, s1);
    strcat(result, s2);

    return result;
}
```