

קורס תכנות

שיעור שנים-עשר:
ניהול זיכרון

הקצאת זיכרון דינאמית

- עד עכשיו עשינו "הקצאה סטטית"
- הגדרנו את משתני התוכנית כבר כשכתבנו אותה
- הקומפיילר הקצה עבורם מקום בזיכרון
- בפרט, בהגדרת מערך קבענו את גודלו ולא יכולנו לשנות אותו בזמן הריצה

```
int main()  
{  
    int a[10];  
    ...  
};
```

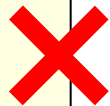
• למשל:

- הנחנו חסם עליון על כמות המשתנים הנדרשת

טעות נפוצה

- ניסיון להגדיר מערך בגודל משתנה

```
int main()
{
    ...
    printf("enter number of elements\n");
    scanf("%d", &size);
    int a[size];
    ...
};
```

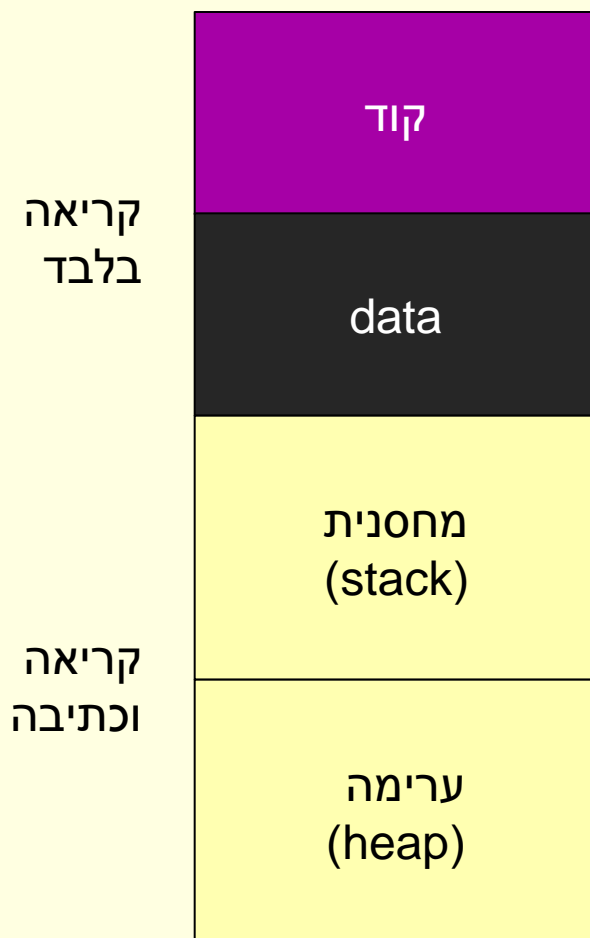


- הקומפיילר צריך לדעת כמה זיכרון להקצות לכל משתנה

הקצאת זיכרון דינאמית

- לא תמיד כל המידע ידוע בזמן כתיבת התכנית
 - לדוגמא, תלוי בקלט מהמשתמש
- ניתן להקצות זיכרון במהלך ריצת התכנית
 - הזיכרון יוקצה באזור ה"ערימה" (heap)
 - לא נעלם כאשר הפונקציה מסתיימת
 - הקצאה ושחרור באחריות המתכנת

מודל הזיכרון של התכנית



• הקוד

- פקודות התכנית

• Data

- מחרוזות שהוגדרו ע"י המתכנת

• מחסנית (Stack)

- משמש לאיחסון משתנים לוקאליים
- Last In First Out (LIFO)
- בשליטת התכנית

• ערימה (Heap)

- בשליטת המתכנת (היום!)

הקצאה דינאמית בשפת C

- C מספקת לנו מנגנון בסיסי להקצאה ושחרור של זיכרון

| פונקציה | מטרה |
|---------|---------------------------------------|
| malloc | הקצאת זיכרון בגודל מבוקש |
| calloc | הקצאת מערך של אברים ואיתחולו ל-0 |
| free | שחרור זיכרון שהוקצה קודם לכן |
| realloc | שינוי גודלו של זיכרון שהוקצה קודם לכן |

- מוגדרות ב `stdlib.h`

הפונקציה malloc

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

- מקבלת את גודל הזיכרון שנרצה להקצות (בבתים)
- מחזירה את הכתובת לזיכרון שהוקצה
 - NULL אם נכשלה
- תמיד יש לבדוק אם הקצאת הזיכרון הצליחה או נכשלה

malloc דוגמא

- הקצאה דינאמית של זיכרון בגודל 10 int

```
int main()
{
    int *a = (int*) malloc(10 * sizeof(int));
    ...
    return 0;
}
```

- האופרטור sizeof מחזיר את הגודל (בבתים) של הטיפוס המבוקש
- הפונקציה מחזירה כתובת (void*) באחריותנו להמיר את המצביע לטיפוס המתאים

שימוש ב malloc

```
type *var-name = (type*) malloc(amount * sizeof(type));
```

מצביע לתחילת
הזיכרון המוקצה

המרת הערך המוחזר
לטיפוס המתאים

גודל הטיפוס
בבתים

- הערך המוחזר הוא כתובת כלשהי
- ללא ציון הטיפוס
- כדי להשתמש במצביע יש לבצע המרה (casting)

הפונקציה free – שיחרור זיכרון

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

- באחריות המתכנת לשחרר זיכרון שהוקצה דינאמית
- ptr - מצביע לזיכרון שהוקצה דינאמית
- ptr - ערך מוחזר מקריאה ל-calloc, malloc, או realloc אחרת שגיאה
- אין שיחרור חלקי של זיכרון

הקצאת מערך בגודל משתנה

```
int main()
```

```
{
```

```
    int *a, size, i;
```

```
    printf("Enter array size\n");
```

```
    scanf("%d", &size);
```

קרא גודל מערך מהמשתמש

הקצאת זיכרון בגודל המבוקש

```
    a = (int*) malloc(size * sizeof(int));
```

```
    if (a == NULL)
```

```
        return 1;
```

בדיקה האם ההקצאה הצליחה

```
    for (i = 0; i < size; i++)
```

```
        scanf("%d", &a[i]);
```

קלט מהמשתמש

```
    for (i = 0; i < size; i++)
```

```
        printf("%d", a[i]);
```

הדפסת תוכן המערך

```
    free(a);
```

שיחרור הזיכרון

```
    return 0;
```

```
}
```

הקצאת מערך בגודל משתנה

```
int main()
{
    int *a, size, i;

    printf("Enter array size\n");
    scanf("%d", &size);

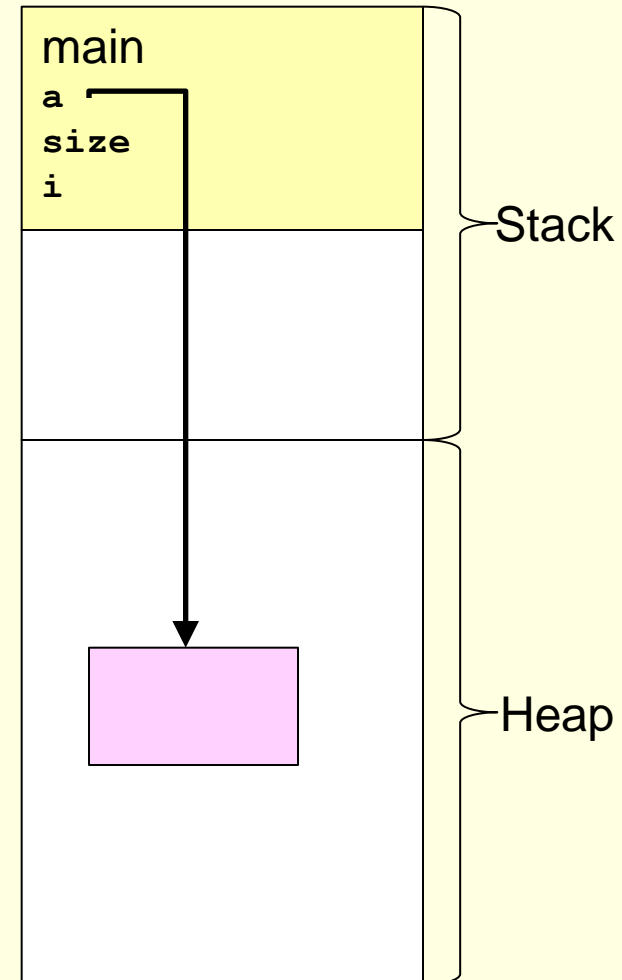
    a = (int*) malloc(size * sizeof(int));
    if (a == NULL)
        return 1;

    for (i = 0; i < size; i++)
        scanf("%d", &a[i]);

    for (i = 0; i < size; i++)
        printf("%d", a[i]);

    free(a);

    return 0;
}
```



פונקציות נוספות

```
void* calloc( unsigned int n, unsigned int size_el )
```

- מקצה מערך של n איברים, כל איבר בגודל size_el בתים, כל בית מאותחל לאפס (ולכן פחות יעילה).

```
void* realloc( void *ptr, unsigned int size )
```

- מקבלת מצביע לזיכרון שהוקצה דינאמית ומספר בתים size
 - הפונקציה מקצה זיכרון בהתאם לגודל הנדרש
 - מעתיקה את תכולת הזיכרון הישן לחדש
 - משחררת את הזיכרון הישן
- בשתי הפונקציות קריאה מוצלחת תחזיר את כתובת תחילת הזיכרון המוקצה, אחרת יוחזר NULL

דוגמא - calloc

```
int main()
{
    int *a, size, i;

    printf("Enter array size\n");
    scanf("%d", &size);

    a = (int*) calloc(size, sizeof(int));
    if (a == NULL)
        return 1;

    for (i = 0; i < size; i++)
        scanf("%d", &a[i]);

    for (i = 0; i < size; i++)
        printf("%d", a[i]);

    free(a);

    return 0;
}
```

הקצאת size אלמנטים, כל אלמנט בגודל sizeof(int) הפונקציה תאתחל את הזיכרון ל-0

דוגמא - realloc

```
int *ptr = NULL;  
int size = 0, new_size = 0;
```

```
scanf("%d", &size);
```

```
ptr = (int*) malloc( size * sizeof(int) );
```



הקצאת זיכרון

```
/* incomplete, must check if allocation succeeded */
```

```
...
```

```
scanf("%d", &new_size);
```

```
ptr = (int*) realloc( ptr, new_size*sizeof(int) );
```



הקצאת חדשה

```
/* incomplete, must check if allocation succeeded */
```

```
...
```

```
if (ptr != NULL)
```

```
    free(ptr);
```



שחרור זיכרון

זיכרון שהוקצה דינאמית

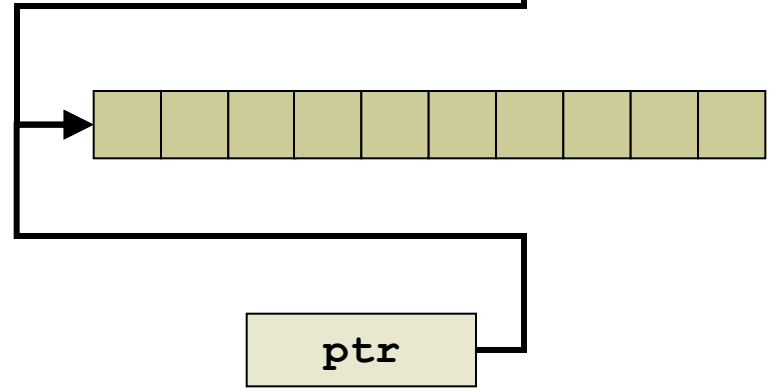
```
int* func()  
{  
    int *memPtr = NULL;  
    memPtr = (int*) malloc(10 * sizeof(int));  
    ...  
    return memPtr;  
}
```

הזיכרון שייך לתוכנית ולא לפונקציה

מותר להחזיר כתובת לזיכרון שהוקצה דינאמית

memPtr

```
int main()  
{  
    int * ptr = func();  
  
    if (ptr != NULL)  
    {  
        // do something with ptr  
        free(ptr);  
        ptr = NULL;  
    }  
}
```



שיחרור זיכרון

- אפשרות א': הפונקציה שהקצתה
 - האפשרות המועדפת – כותב הפונקציה אחראי על הזיכרון הדינאמי
- אפשרות ב': החזרת הכתובת
 - האחריות עוברת למי שקרא לפונקציה המקצה
 - חובה לתעד זאת, אחרת "דליפת זיכרון"
- עלינו להימנע ממצב שבו יש הקצאת זיכרון אבל לא מתקיים א' או ב'

קריאת שורת קלט בגודל לא ידוע

```
char* readLine()
{
    int index = 0, c, capacity = INITIAL_SIZE;
    char *buffer = (char*) malloc(capacity * sizeof(char));

    if (buffer == NULL)
        return NULL;

    for (c = getchar(); c != '\n'; c = getchar())
    {
        if (index == capacity - 1)
        {
            buffer = (char*) realloc(buffer, capacity * INCREMENT * sizeof(char));
            if (buffer == NULL)
                return NULL;
            capacity = capacity * INCREMENT * sizeof(char);
        }
        buffer[index++] = c;
    }
    buffer[index] = '\0';
    return buffer;
}
```

קריאת שורת קלט בגודל לא ידוע

```
char* readLine()
{
    int index = 0, c, capacity = INITIAL_SIZE;
    char *buffer = (char*) malloc(capacity * sizeof(char));

    if (buffer == NULL)
        return NULL;

    for (c = getchar(); c != '\n'; c = getchar())
    {
        if (index == capacity - 1)
        {
            buffer = (char*) realloc(buffer, capacity * INCREMENT * sizeof(char));
            if (buffer == NULL)
                return NULL;
            capacity = capacity * INCREMENT;
        }
        buffer[index++] = c;
    }
    buffer[index] = '\0';
    return buffer;
}
```

מערך בגודל התחלתי כלשהו

קריאת שורת קלט בגודל לא ידוע

```
char* readLine()
{
    int index = 0, c, capacity = INITIAL_SIZE;
    char *buffer = (char*) malloc(capacity * sizeof(char));

    if (buffer == NULL)
        return NULL;

    for (c = getchar(); c != '\n'; c = getchar())
    {
        if (index == capacity - 1)
        {
            buffer = (char*) realloc(buffer, capacity * INCREMENT * sizeof(char));
            if (buffer == NULL)
                return NULL;
            capacity = capacity * INCREMENT;
        }
        buffer[index++] = c;
    }
    buffer[index] = '\0';
    return buffer;
}
```

יש לוודא שהקצאת הזיכרון לא נכשלה

קריאת שורת קלט בגודל לא ידוע

```
char* readLine()
{
    int index = 0, c, capacity = INITIAL_SIZE;
    char *buffer = (char*) malloc(capacity * sizeof(char));

    if (buffer == NULL)
        return NULL;

    for (c = getchar(); c != '\n'; c = getchar())
    {
        if (index == capacity - 1)
        {
            buffer = (char*) realloc(buffer, capacity * INCREMENT * sizeof(char));
            if (buffer == NULL)
                return NULL;
            capacity = capacity * INCREMENT;
        }
        buffer[index++] = c;
    }
    buffer[index] = '\0';
    return buffer;
}
```

קריאת קלט מהמשתמש עד לסוף השורה

קריאת שורת קלט בגודל לא ידוע

```
char* readLine()
{
    int index = 0, c, capacity = INITIAL_SIZE;
    char *buffer = (char*) malloc(capacity * sizeof(char));

    if (buffer == NULL)
        return NULL;

    for (c = getchar(); c != '\n'; c = getchar())
    {
        if (index == capacity - 1) _____ האם נגמר המקום?
        {
            buffer = (char*) realloc(buffer, capacity * INCREMENT * sizeof(char));
            if (buffer == NULL)
                return NULL;
            capacity = capacity * INCREMENT;
        }
        buffer[index++] = c;
    }
    buffer[index] = '\0';
    return buffer;
}
```

קריאת שורת קלט בגודל לא ידוע

```
char* readLine()
{
    int index = 0, c, capacity = INITIAL_SIZE;
    char *buffer = (char*) malloc(capacity * sizeof(char));

    if (buffer == NULL)
        return NULL;

    for (c = getchar(); c != '\n'; c = getchar())
    {
        if (index == capacity - 1)
        {
            buffer = (char*) realloc(buffer, capacity * INCREMENT * sizeof(char));
            if (buffer == NULL)
                return NULL;
            capacity = capacity * INCREMENT;
        }
        buffer[index++] = c;
    }
    buffer[index] = '\0';
    return buffer;
}
```

נקצה מערך גדול יותר

קריאת שורת קלט בגודל לא ידוע

```
char* readLine()
{
    int index = 0, c, capacity = INITIAL_SIZE;
    char *buffer = (char*) malloc(capacity * sizeof(char));

    if (buffer == NULL)
        return NULL;

    for (c = getchar(); c != '\n'; c = getchar())
    {
        if (index == capacity - 1)
        {
            buffer = (char*) realloc(buffer, capacity * INCREMENT * sizeof(char));
            if (buffer == NULL)
                return NULL;
            capacity = capacity * INCREMENT;
        }
        buffer[index++] = c;
    }
    buffer[index] = '\0';
    return buffer;
}
```

לא לשכוח לבדוק

קריאת שורת קלט בגודל לא ידוע

```
char* readLine()
{
    int index = 0, c, capacity = INITIAL_SIZE;
    char *buffer = (char*) malloc(capacity * sizeof(char));

    if (buffer == NULL)
        return NULL;

    for (c = getchar(); c != '\n'; c = getchar())
    {
        if (index == capacity - 1)
        {
            buffer = (char*) realloc(buffer, capacity * INCREMENT * sizeof(char));
            if (buffer == NULL)
                return NULL;
            capacity = capacity * INCREMENT;
        }
        buffer[index++] = c;
    }
    buffer[index] = '\0';
    return buffer;
}
```

לא לשכוח לעדכן את capacity

קריאת שורת קלט בגודל לא ידוע

```
char* readLine()
{
    int index = 0, c, capacity = INITIAL_SIZE;
    char *buffer = (char*) malloc(capacity * sizeof(char));

    if (buffer == NULL)
        return NULL;

    for (c = getchar(); c != '\n'; c = getchar())
    {
        if (index == capacity - 1)
        {
            buffer = (char*) realloc(buffer, capacity * INCREMENT * sizeof(char));
            if (buffer == NULL)
                return NULL;
            capacity = capacity * INCREMENT;
        }
        buffer[index++] = c;
    }
    buffer[index] = '\0';
    return buffer;
}
```

נשמור את התו שקראנו במקום המתאים

קריאת שורת קלט בגודל לא ידוע

```
char* readLine()
{
    int index = 0, c, capacity = INITIAL_SIZE;
    char *buffer = (char*) malloc(capacity * sizeof(char));

    if (buffer == NULL)
        return NULL;

    for (c = getchar(); c != '\n'; c = getchar())
    {
        if (index == capacity - 1)
        {
            buffer = (char*) realloc(buffer, capacity * INCREMENT * sizeof(char));
            if (buffer == NULL)
                return NULL;
            capacity = capacity * INCREMENT;
        }
        buffer[index++] = c;
    }
    buffer[index] = '\0';
    return buffer;
}
```

בסוף נוסף '\0'

קריאת שורת קלט בגודל לא ידוע

```
char* readLine()
{
    int index = 0, c, capacity = INITIAL_SIZE;
    char *buffer = (char*) malloc(capacity * sizeof(char));

    if (buffer == NULL)
        return NULL;

    for (c = getchar(); c != '\n'; c = getchar())
    {
        if (index == capacity - 1)
        {
            buffer = (char*) realloc(buffer, capacity * INCREMENT * sizeof(char));
            if (buffer == NULL)
                return NULL;
            capacity = capacity * INCREMENT;
        }
        buffer[index++] = c;
    }
    buffer[index] = '\0';
    return buffer;
}
```

מחזירים זיכרון שהוקצה דינאמית –
באחריות הלקוח לשחרר (צריך לתעד)

שימוש ב readLine

קריאת שורת קלט מהמשתמש

```
int main()
{
    char *line = readLine();

    if (line == NULL)
    {
        printf("Fatal error: memory allocation failed!\n");
        return 1;
    }
    printf("%s\n", line);

    free(line);

    return 0;
}
```

שימוש ב readLine

```
int main()
{
    char *line = readLine();

    if (line == NULL)
    {
        printf("Fatal error: memory allocation failed!\n");
        return 1;
    }
    printf("%s\n", line);

    free(line);

    return 0;
}
```

אם הייתה בעיה נדפס הודעת שגיאה
ונסיים

שימוש ב readLine

```
int main()
{
    char *line = readLine();

    if (line == NULL)
    {
        printf("Fatal error: memory allocation failed!\n");
        return 1;
    }
    printf("%s\n", line);

    free(line);

    return 0;
}
```

שימו לב לערך המוחזר

שימוש ב readLine

```
int main()
{
    char *line = readLine();

    if (line == NULL)
    {
        printf("Fatal error: memory allocation failed!\n");
        return 1;
    }
    printf("%s\n", line);

    free(line);

    return 0;
}
```

נשתמש בקלט

שימוש ב readLine

```
int main()
{
    char *line = readLine();

    if (line == NULL)
    {
        printf("Fatal error: memory allocation failed!\n");
        return 1;
    }
    printf("%s\n", line);

    free(line);

    return 0;
}
```

כשאיננו צריכים יותר את הקלט, נשחרר את הזיכרון

הקצאת זיכרון - בעיות

- נדיר שהקצאת הזיכרון נכשלת
- אפשרויות לטיפול:
 - החזרת הערך NULL מהפונקציה - כאשר תפקיד הפונקציה הוא הקצאת הזיכרון
 - שימוש בפונקציה `exit` לסיום התכנית

הפונקציה exit

```
void exit(int status);
```

- מוגדרת ב `stdlib.h`
- מסיימת את התכנית באופן מיידי
- נשתמש רק כאשר מתרחשת בעיה שאין לנו שום דרך לטפל בה
- בדרך כלל נדפיס הודעת שגיאה למשתמש ונסיים את תכנית

דוגמה לשימוש ב-exit

- נשנה את גודלו של מערך של int בעזרת realloc
- אם השינוי נכשל, נסיים את התכנית

```
int* resize(int *array, int newSize)
{
    if (array == NULL)
        return array;

    array = (int*) realloc(array, newSize * sizeof(int));
    if (array == NULL)
    {
        printf("Fatal error: memory allocation failed!\n");
        exit(1);
    }
    return array;
}
```

הקצאה דינאמית של מבנים

• בדיוק כמו בטיפוסים בסיסיים

```
typedef struct {
    char *title, *author;
} Book;

Book* newBook(const char *title, const char *author)
{
    Book *result = (Book*) malloc(sizeof(Book));
    /* incomplete, must check if allocation succeeded */

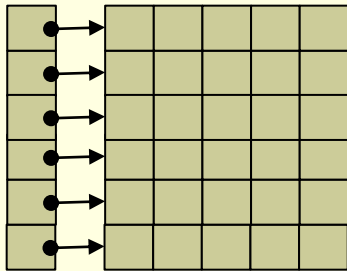
    result->title = (char*) malloc(strlen(title) + 1);
    /* incomplete, must check if allocation succeeded */
    strcpy(result->title, title);

    result->author = (char*) malloc(strlen(author) + 1);
    /* incomplete, must check if allocation succeeded */
    strcpy(result->author, author);

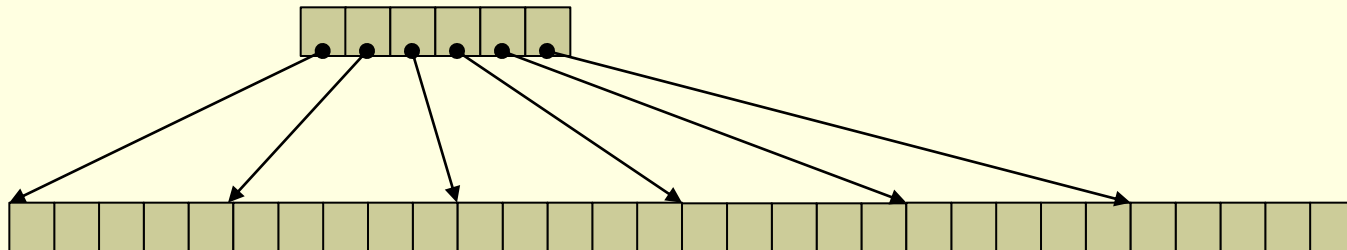
    return result;
}
```

הקצאה דינאמית של מערכים דו-ממדיים

- שתי גישות אפשריות:
1. מערך של מערכים



- 2. מצביעים לתוך מערך "גדול"



מערך של מערכים

```
int main() {
    int rows, cols, i;
    int** table;

    scanf("%d%d", &rows, &cols);

    table = (int**) malloc(rows * sizeof(int*));
    /* incomplete, must check if failed */
    for (i = 0; i < rows; i++) {
        table[i] = (int*) malloc(cols * sizeof(int));
        /* incomplete, must check if failed */
    }

    init_table(table, rows, cols);
    print_table(table, rows, cols);

    for (i = 0; i < rows; i++)
        free(table[i]);
    free(table);
    return EXIT_SUCCESS;
}
```

איתחול והדפסה

```
void init_table(int **table, int rows, int cols)
{
    int i, j;

    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            table[i][j] = (i + 1) * (j + 1);
}

void print_table(int **table, int rows, int cols)
{
    int i, j;

    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++)
            printf("%4d", table[i][j]);
        printf("\n");
    }
}
```


מצביעים לתוך מערך גדול

```
int main() {
    int rows, cols, i;
    int* data;
    int** table;

    scanf("%d%d", &rows, &cols);

    data = (int*) malloc(rows * cols * sizeof(int));
    /* incomplete, must check if failed */
    table = (int**) malloc(rows * sizeof(int*));
    /* incomplete, must check if failed */
    for (i = 0; i < rows; i++)
        table[i] = data + (cols * i);

    init_table(table, rows, cols);
    print_table(table, rows, cols);

    free(table);
    free(data);

    return EXIT_SUCCESS;
}
```

משתנים בשורת הפקודה

Command Line Arguments

- בעת הרצת התוכנית ניתן להעביר לה משתנים בשורת הפקודה
- משתנים אלו יועברו כארגומנטים ל-main
- כל המשתנים הם מחרוזות

מספר המשתנים

מערך של מחרוזות (המשתנים)

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

מי מעביר את הערכים?

- סביבת הריצה (מערכת ההפעלה) אחראית להעביר את המשתנים ל-main
- המשתנים יקיימו את התנאים הבאים:
 - `argc` יהיה חיובי
 - `argv[argc] == NULL`
 - `argv[0]` היא מחרוזת המכילה את שם התכנית

• התכנית מדפיסה את ערכי המשתנים

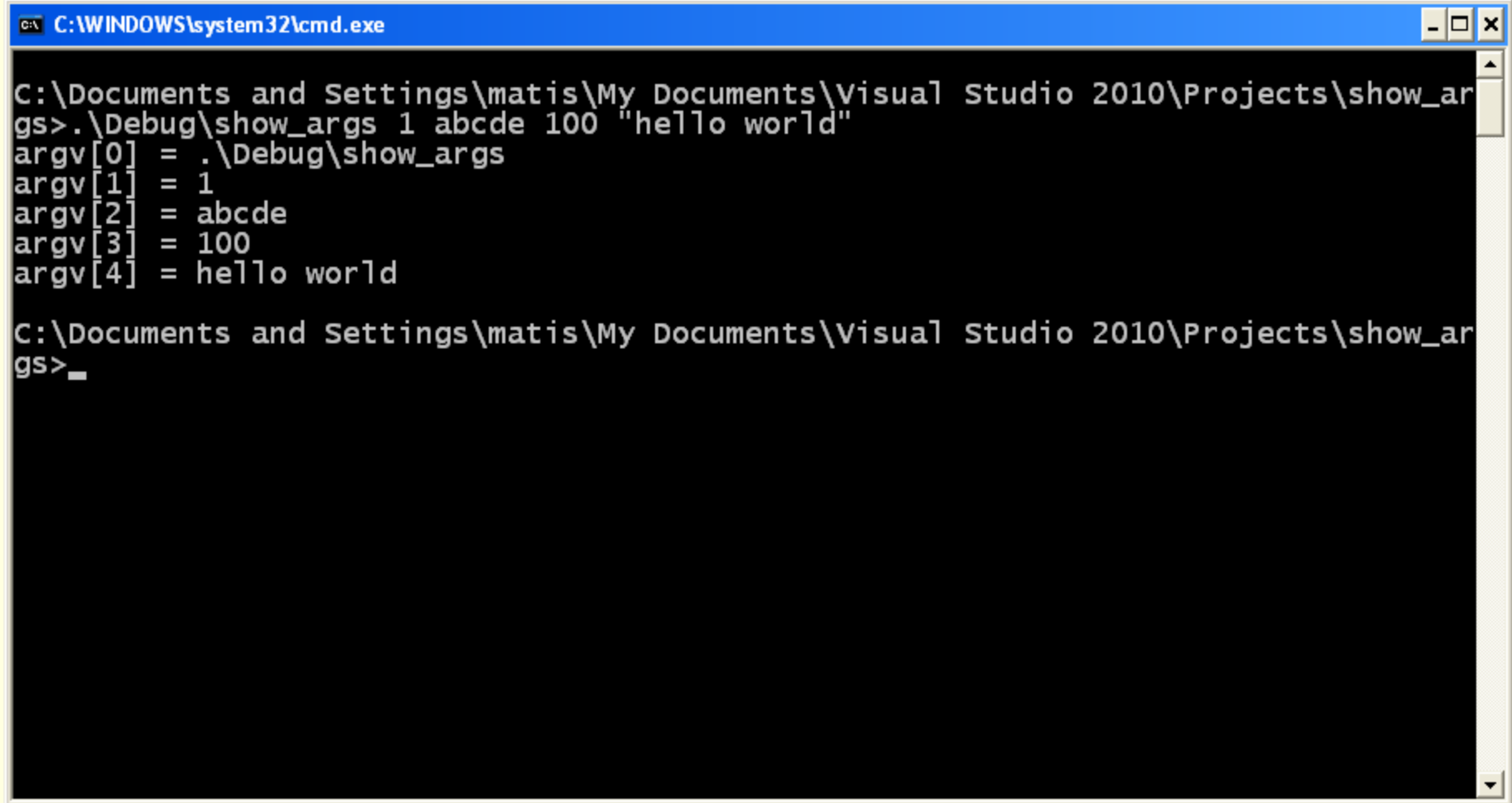
```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int i;

    for (i = 0; i < argc; i++)
        printf("argv[%d] = %s\n", i, argv[i]);

    return EXIT_SUCCESS;
}
```

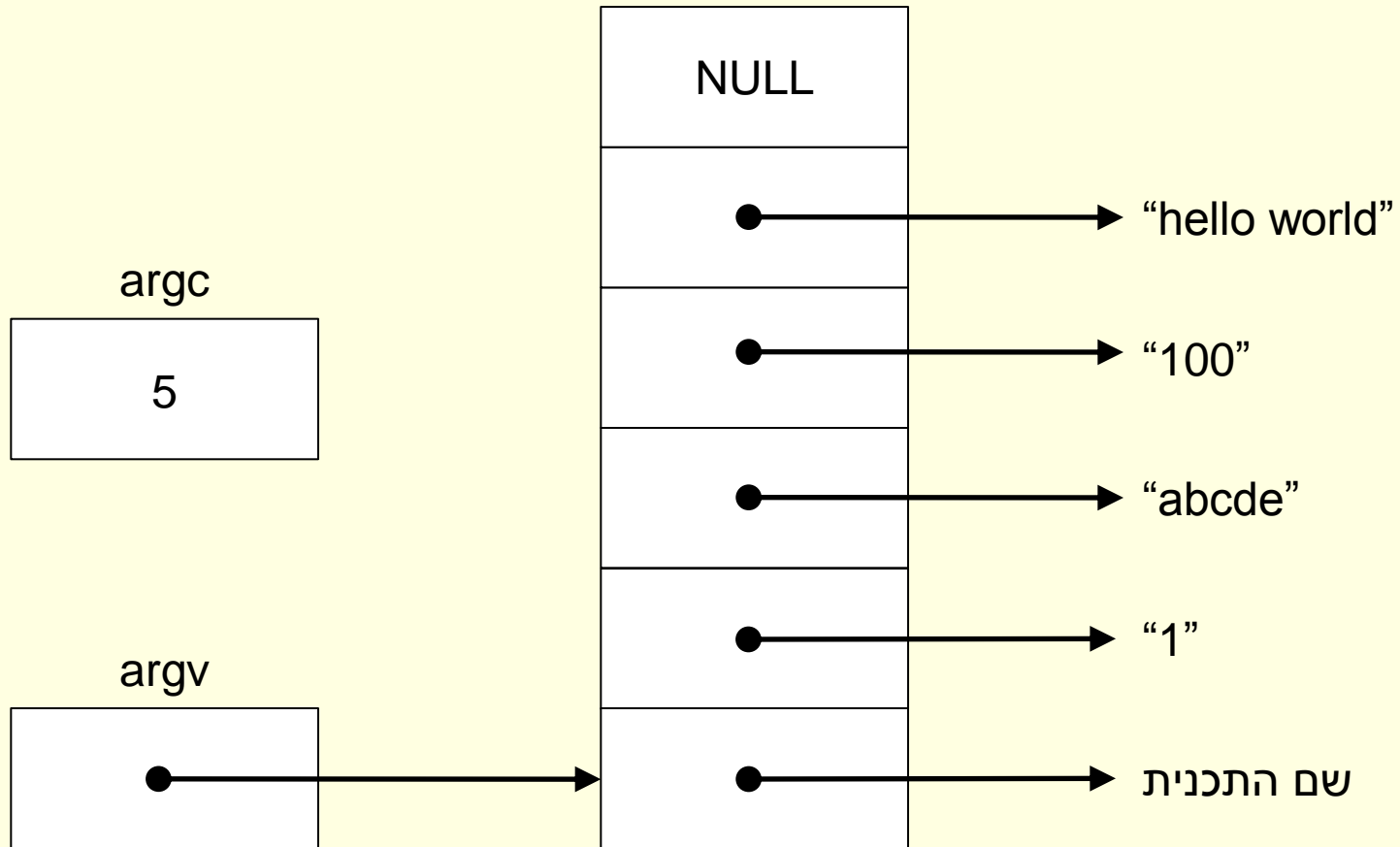
דוגמת הרצה



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\matis\My Documents\Visual Studio 2010\Projects\show_ar
gs>.\Debug\show_args 1 abcde 100 "hello world"
argv[0] = .\Debug\show_args
argv[1] = 1
argv[2] = abcde
argv[3] = 100
argv[4] = hello world

C:\Documents and Settings\matis\My Documents\Visual Studio 2010\Projects\show_ar
gs>_
```

מערך של ארגומנטים



מחשבון פשוט

```
int main(int argc, char *argv[])
{
    double leftOperand, rightOperand, result;
    char operator;

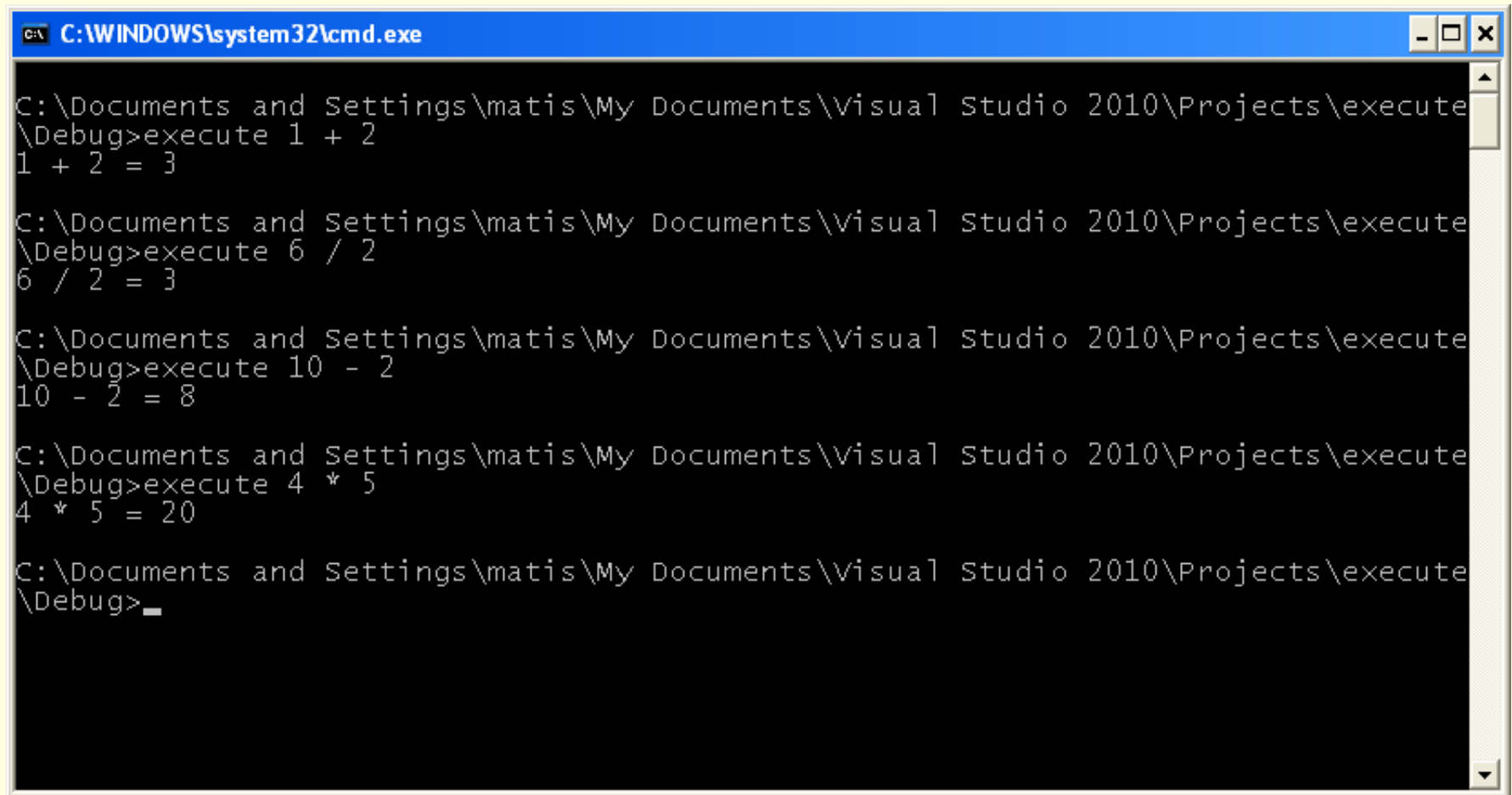
    /* incomplete, make sure all arguments are present */

    leftOperand = atof(argv[1]);
    rightOperand = atof(argv[3]);
    operator = *argv[2];

    switch (operator) {
    case '+':
        result = leftOperand + rightOperand;
        break;
    case '-':
        result = leftOperand - rightOperand;
        break;
    case '*':
        result = leftOperand * rightOperand;
        break;
    case '/':
        result = leftOperand / rightOperand;
        break;
    }

    printf("%g %c %g = %g\n", leftOperand, operator, rightOperand, result);
    return EXIT_SUCCESS;
}
```


דוגמת הרצה



```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\matis\My Documents\Visual Studio 2010\Projects\execute
\Debug>execute 1 + 2
1 + 2 = 3

C:\Documents and Settings\matis\My Documents\Visual Studio 2010\Projects\execute
\Debug>execute 6 / 2
6 / 2 = 3

C:\Documents and Settings\matis\My Documents\Visual Studio 2010\Projects\execute
\Debug>execute 10 - 2
10 - 2 = 8

C:\Documents and Settings\matis\My Documents\Visual Studio 2010\Projects\execute
\Debug>execute 4 * 5
4 * 5 = 20

C:\Documents and Settings\matis\My Documents\Visual Studio 2010\Projects\execute
\Debug>_
```