



به نام خدا

دانشکده مهندسی برق و کامپیوتر دانشکده فنی دانشگاه تهران

مبانی کامپیوتر و برنامه نویسی



استاد : دکتر مرادی

عنوان:

آرایه های پویا و ساختمان ها

نیمسال دوم

99-98

آرایه ی پویا^۱:

آرایه هایی که تا به حال دیده اید و از آن ها استفاده کرده اید، آرایه های ایستا^۲ بوده اند. اگر یادتان باشد در تعریف این آرایه ها حتماً باید طول آن ها را با یک عدد ثابت مشخص می کردید. امروز می خواهیم با نوع دیگری از آرایه ها به نام آرایه پویا آشنا شویم. طول این آرایه ها در هنگام کامپایل نامشخص بوده و در هنگام اجرا تعیین می گردد.

دستور تخصیص حافظه^۳ (malloc):

شما می توانید توسط تابع malloc که از توابع کتابخانه ای stdlib.h می باشد، از سیستم عامل درخواست کنید که مقدار مشخصی حافظه در heap گرفته و آن را در اختیار شما قرار دهد. نحوه ی استفاده از این تابع به صورت زیر است:

```
<type>* pointer = (<type>*) malloc(number * sizeof(<type>));
```

<type> : نوع داده ای که می خواهید آرایه ای پویا از آن داشته باشید.

number : طول آرایه ای که می خواهید.

حال به نکات زیر **توجه کنید**:

1. آرگومان تابع malloc مقدار حافظه درخواستی بر حسب بایت می باشد.
2. sizeof از عملگر های زبان C است که سایز هر type ای که به آن بدهید را بر حسب بایت برمی گرداند. چون سایز یک type (مثلاً int) در سیستم های مختلف ممکن است متفاوت باشد، بهتر است از عملگر sizeof استفاده کنید.
3. مقدار برگشتی تابع malloc در صورتی که تخصیص حافظه موفقیت آمیز باشد، اشاره گر به سر آرایه ی پویا خواهد بود و در غیر این صورت NULL است. لذا بعد از فراخوانی این تابع **حتماً** باید بررسی کنید که اگر مقدار بازگشتی NULL بود، ضمن دادن پیغام خطا به کاربر از برنامه خارج شوید.
4. همچنین مقدار برگشتی این تابع از جنس void* (یعنی اشاره گری که می تواند از هر نوعی باشد و لزوماً قرار نیست نوع خاصی، مثلاً integer، داشته باشد) بوده و برای همین آن را به type مورد نظر cast کرده ایم.
5. هر حافظه ای که توسط تابع malloc می گیرید را در پایان **باید** توسط تابع free(pointer) آزاد کنید.

¹ dynamic

² static

³ memory allocation

6. نحوه‌ی استفاده از آرایه‌های ایستا و پویا هیچ تفاوتی با هم ندارد. پس همان گونه که قبلاً با آرایه‌های ایستا کار می‌کردید، می‌توانید با آرایه‌های پویا نیز کار کنید.

برای درک نکات بالا به برنامه زیر توجه کنید:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int arr_size, i;
    int* dynamic_arr;
    printf("Enter the size of array:\n");
    scanf("%d", &arr_size);
    /* Requesting an integer array with capacity of arr_size elements.
     * On success dynamic_arr will be a pointer to the beginning of the array.
     * On failure dynamic_arr will be null. */
    dynamic_arr = (int*) malloc (arr_size * sizeof(int));
    if (dynamic_arr == NULL) {
        printf("Oops! Memory allocation failed.\n");
        exit(EXIT_FAILURE);
    }
    /* From now on you can work with the dynamic array just like static arrays!*/
    printf("Enter %d numbers:\n", arr_size);
    for (i = 0; i < arr_size; i++)
        scanf ("%d", &dynamic_arr[i]);
    free(dynamic_arr); /* Do not forget to free the allocated memory! */
    return 0;
}
```

1. انجام دهید

یکی از توابع کاربردی برای تخصیص حافظه‌های پویا تابع `realloc` است. از این تابع می‌توانید برای تغییر مقدار حافظه‌ای که قبلاً از سیستم گرفته بودید، استفاده کنید. تعریف این تابع به صورت زیر است:

```
<type>* pointer = (<type>*) realloc(pointer, number * sizeof(<type>));
```

این تابع به عنوان ورودی اشاره‌گر فعلی و اندازه‌ی جدید مورد نظر را گرفته و اشاره‌گر جدید را بر می‌گرداند. برای آشنایی با رفتار این تابع، قطعه کدهای زیر را اجرا کنید. تفاوت نتیجه‌ی این دو قطعه کد در چیست؟

قطعه کد اول:

```
#include <stdio.h>
#include <stdlib.h>
#define INITIAL_ARRAY_SIZE 4
#define FINAL_ARRAY_SIZE 5
#define ZERO 0
#define ONE 1
```

```

#define FIVE 5

void print_array(int* array_of_int, int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", array_of_int[i]);
    printf("\n");
}

void initislizing_the_array(int* array_of_int, int size) {
    for (int i = ZERO; i < size; i++)
        array_of_int[i] = ZERO;
}

int main(int argv, char** argc) {
    int* array_of_int = (int*)malloc(INITIAL_ARRAY_SIZE *
sizeof(int));
    initislizing_the_array(array_of_int, INITIAL_ARRAY_SIZE);
    print_array(array_of_int, INITIAL_ARRAY_SIZE);
    array_of_int = (int*)realloc(array_of_int, FINAL_ARRAY_SIZE *
sizeof(int));
    initislizing_the_array(array_of_int, FINAL_ARRAY_SIZE);
    array_of_int[FINAL_ARRAY_SIZE - ONE] = FIVE;
    print_array(array_of_int, FINAL_ARRAY_SIZE);
    return 0;
}

```

قطعه کد دوم:

```

#include <stdio.h>
#include <stdlib.h>
#define INITIAL_ARRAY_SIZE 4
#define FINAL_ARRAY_SIZE 5
#define ZERO 0
#define ONE 1
#define FIVE 5

void print_array(int* array_of_int, int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", array_of_int[i]);
    printf("\n");
}

void initislizing_the_array(int* array_of_int, int size) {
    for (int i = ZERO; i < size; i++)
        array_of_int[i] = ZERO;
}

```

```

int main(int argv, char** argc) {
    int* array_of_int = (int*)malloc(INITIAL_ARRAY_SIZE *
sizeof(int));
    initislizing_the_array(array_of_int, INITIAL_ARRAY_SIZE);
    print_array(array_of_int, INITIAL_ARRAY_SIZE);
    array_of_int = (int*)realloc(NULL, FINAL_ARRAY_SIZE *
sizeof(int));
    array_of_int[FINAL_ARRAY_SIZE - ONE] = FIVE;
    print_array(array_of_int, FINAL_ARRAY_SIZE);
    return 0;
}

```

قسمت 1: نتیجه را به دستیاران آموزشی نشان دهید.

2. انجام دهید (Free)



1) هنگام کار با تخصیص حافظه‌ی پویا به دو نکته باید توجه نمود :

1. لزوم آزاد کردن حافظه‌های گرفته شده

2. عدم استفاده از حافظه‌ی آزاد شده

2) قطعه کد زیر را اجرا کنید.

```

int main() {
    int* p = (int*)malloc(10 * sizeof(int));
    int i;
    printf("P = 0x%p\n", p);
    for (i = 0; i < 10; i++)
    {
        p[i] = i;
    }
    free(p);
    printf("P = 0x%p\n", p);
    printf("P[0] = %d", *p);
    return 0;
}

```

پس از آزادسازی حافظه‌ی گرفته شده دقیقاً چه اتفاقی می‌افتد؟ خروجی‌ها را توجیه کنید.

قسمت 2: نتیجه را با دستیاران آموزشی در میان بگذارید.

فکر کنید : چرا اگر در کد بالا اقدام به چاپ `p[15]` کنیم برنامه خطا نمی‌دهد؟

3. انجام دهید (Memory Leak) ←

حافظه‌های گرفته شده از سیستم هنگام اجرای برنامه حتماً باید در انتهای برنامه آزاد شوند. در این قسمت مشاهده می‌کنیم که در صورتی حافظه‌ی گرفته شده آزاد نشود ممکن است چه مشکلاتی برای سیستم ایجاد شود.

1) کد زیر را در حالت debug و با قرار دادن breakpoint در محل ذکر شده اجرا کنید.

```
void main() {  
    int *p = NULL;  
    int i = 500000;  
    while (1) {  
        p = realloc(p, i * sizeof(int)); /*put breakpoint here*/  
        i+=500000;  
    }  
    free(p);  
    return 0;  
}
```

2) با استفاده از کلیدهای ctrl+shift+esc پنجره‌ی task manager را باز نمایید.

3) سربرگ Processes را انتخاب کنید.

4) برنامه‌ی اجرایی your_project_name.exe را پیدا کنید و مقدار memory مورد استفاده‌ی آن را مشاهده کنید.

5) با فشردن دکمه‌ی f5 به تعداد 10 بار حلقه‌ی while را اجرا کنید.

6) مقدار memory گرفته شده توسط برنامه را مجدداً مشاهده کنید.

7) چه نتیجه‌ای می‌گیرید؟

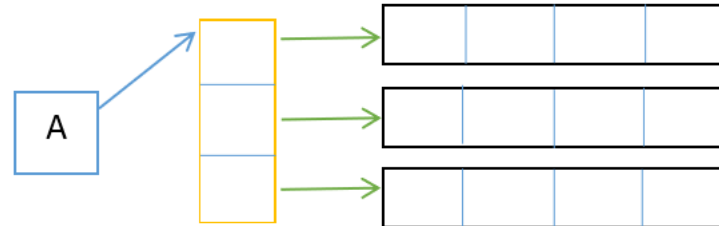
قسمت 3: نتیجه را با دستیاران آموزشی در میان بگذارید.

فکر کنید: آیا آدرس p که به ابتدای حافظه‌ی مورد نظر ما اشاره می‌کند همیشه یکسان است؟

آرایه پویای دو بعدی (!):

قبلاً با آرایه‌های چند بعدی هم کار کرده بودیم. حال می‌خواهیم همان آرایه‌ها را نیز به شکل پویا درست کنیم. یک آرایه‌ی دوبعدی، عملاً آرایه‌ای از آرایه‌های یک بعدی است. مثلاً `int a[3][4]` یک آرایه‌ی 3 تایی از آرایه‌هایی به طول 4 می‌باشد. شکل زیر را نگاه کنید:

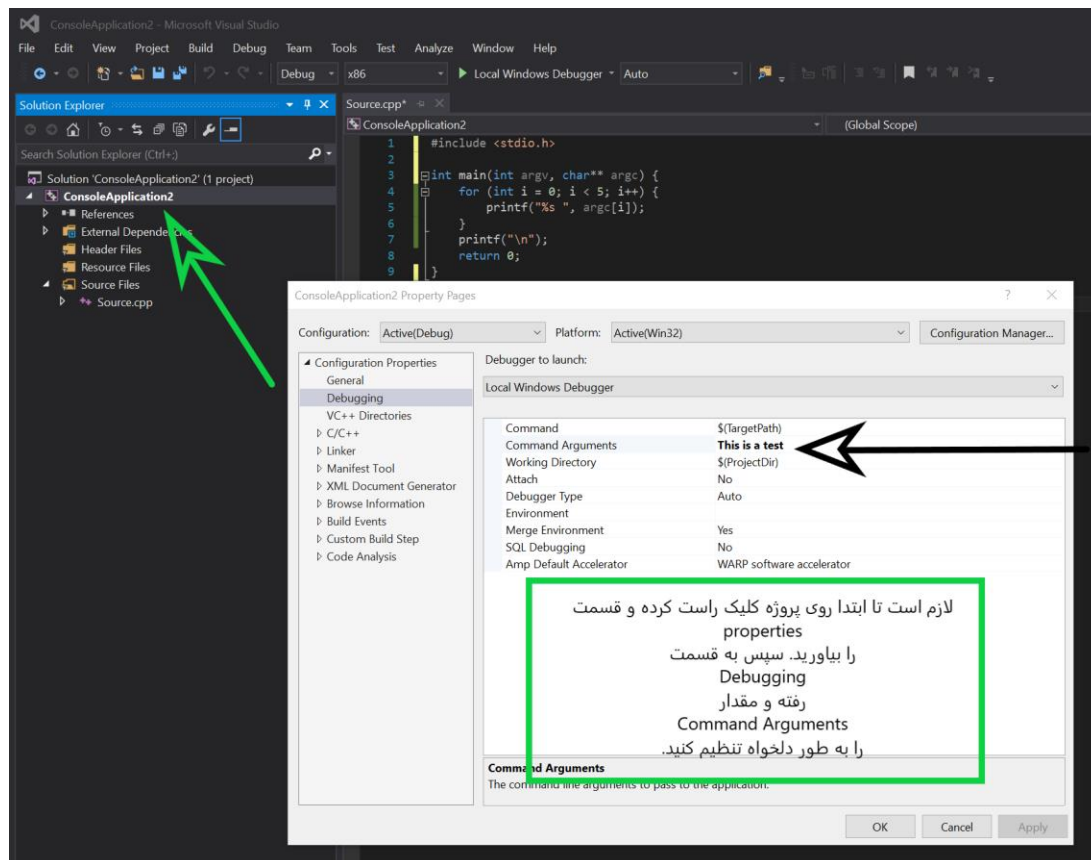
	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]



پس اگر ما ابتدا یک آرایه از جنس `int**` به طول 3 بسازیم که `A` سر آن باشد، و سپس هر کدام از خانه های آن آرایه را برابر با سر یک آرایه ی 4 تایی قرار دهیم، یک آرایه ی دوبعدی ساخته ایم.

4. انجام دهید! ←

در این قسمت قرار است تا با شیوه ی دیگری از دادن ورودی به برنامه آشنا شوید. تا کنون، با استفاده از تابع `scanf` و در زمان اجرای برنامه، ورودی هایی را از کاربر دریافت می کردید. اکنون شیوه ی دیگری را به شما معرفی می کنیم که بوسیله ی آن می توانید ورودی هایی را پیش از اجرای برنامه دریافت کنید. در نتیجه هنگامی که برنامه اجرا می شود، متغیرهایی که ورودی های مذکور را نگهداری می کنند، مقدار گرفته اند. نام این متغیرها در زبان C به ترتیب `argv` و `argc` است. در ادامه با کاربرد آنها آشنا می شوید. اکنون مراحل زیر را مطابق شکل ها انجام دهید:



پس از انجام دستور بالا، قطعه کد زیر را اجرا کرده و مقادیر موجود در `argc` را تفسیر کنید. در اینجا مقدار `argv` نشان‌دهنده‌ی چیست؟

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

```
int main(int argv, char** argc) {
    for (int i = 0; i < 5; i++) {
        printf("%s ", argc[i]);
    }
    printf("\\n");
    return 0;
}
```

قسمت 4: نتیجه را به دستیاران آموزشی نشان دهید.

5. انجام دهید! ←

قسمت‌های مشخص شده در کد زیر کامل کرده، سپس کد زیر را اجرا کنید.

```

#include <stdio.h>
#include <stdlib.h>
int main() {
    int row, col, i;
    int** A;
    printf("Enter row and column:\n");
    scanf("%d %d", &row, &col);
    A=(...) malloc(...*sizeof(...)); /* 1. Complete this instruction */
    if (A == NULL) exit(EXIT_FAILURE);
    for(i = 0; i < row ; i++) {
        A[i]=(...) malloc(...*sizeof(...)); /* 2. Complete this instruction */
        if (A[i] == NULL)
            exit(EXIT_FAILURE);
    }
    /* Now you have a 2D integer array */

    Your Program Goes Here.

    /* Don't forget to free the allocated memory when you don't need it any more */
    for (i = 0; i < row; i++)
        free(A[i]);
    free(A);
    return 0;
}

```

سوال: نحوه ی آزاد سازی حافظه در سوال قبل را توضیح دهید. آیا لازم است که در یک حلقه هر سطر را جداگانه آزاد کنیم؟ یا دستور `free(A)` به تنهایی اکتفا می کند؟

قسمت 5: نتیجه را به دستیاران آموزشی نشان دهید.

توجه: همه ی نکات ذکر شده به راحتی قابل تعمیم به یک آرایه ی n بعدی است

6. فکر کنید. 

کد زیر را اجرا کنید و درباره ی خطوط اشاره شده فکر کنید.

```

#include <stdio.h>
#include <stdlib.h>
void main() {
    int i;
    char* s;

```



```

int *p=(int*)malloc(10*sizeof(int));
for(i=0;i<10;i++)
    p[i]=i+48;
s=(char*)p;
for(i=0; i<40; i++) /* Pay attention to bound of for */
    printf("%c", s[i]); /* What's happening here? what is value of s[1]? why? */
printf("\n");
p++;
printf("p[1] is %d\n",*p);
free(p); /* What's happening here? */
}

```

قسمت 6: نتیجه را با دستیاران آموزشی در میان بگذارید.

7. فکر کنید. ←

در درس با ساختار ساختمان ها (structures) در زبان C آشنا شدید. عکس پایین نمونه ای برای یاد آوری شما میباشد:

Records (structs)

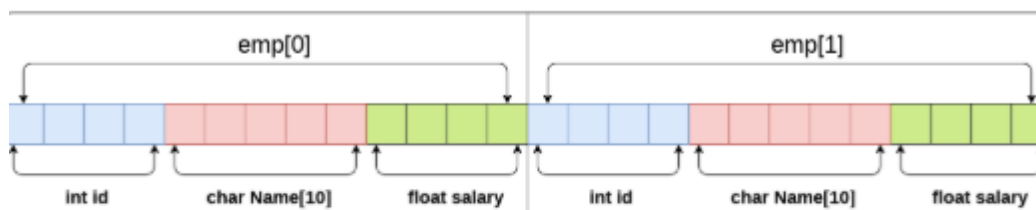
- struct: collection of a fixed number of components (members), accessed by name
 - Members may be of different types
- Syntax:

```

struct structName
{
    dataType1 identifier1;
    dataType2 identifier2;
    .
    .
    .
    dataTypeN identifierN;
};

```

همچنین میتوان آرایه ای از struct ها داشت :



```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

$\text{sizeof}(\text{emp}) = 4 + 5 + 4 = 13 \text{ bytes}$

هدف شما در این قسمت صرفاً تحلیل قطعه کدی است که در کنار این فایل به نام `part7_code` قرار دارد.

قسمت 7: نتیجه را با دستیاران آموزشی در میان بگذارید.

بخش های اختیاری در یک فایل جداگانه بر روی سایت قرار گرفته است. توصیه می شود که حتماً آن ها را انجام دهید.

موفق باشید