به نام خدا

دستورکار آزمایشگاه 2-8

آرایه های پویا

آرایه‌ی پویا[1]:

آرایه هایی که تا به حال دیده اید و از آن ها استفاده کرده اید، آرایه های ایستا[2] 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[3] (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**:

۱. آرگومان تابع malloc مقدار حافظه درخواستی بر حسب بایت می باشد.
۲. 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.
۳. 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.
۴. همچنین مقدار برگشتی این تابع از جنس *void بوده و برای همین آن را به type مورد نظر cast کرده ایم.

[1] dynamic
[2] static
[3] memory allocation

۵. هر حافظه ای که توسط تابع malloc می گیرید را در پایان **باید** توسط تابع free(pointer) آزاد کنید.

۶. نحوه ی استفاده از آرایه های ایستا و پویا هیچ تفاوتی با هم ندارد. پس همان گونه که قبلاً با آرایه های ایستا کار می کردید، می توانید با آرایه های 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]);
	/* Do not forget to free the allocated memory! */
	free(dynamic_arr);
	return 0;
}
```

۱. فکر کنید.

۱) کد زیر را اجرا کنید و درباره‌ی 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? */
}
```

نتیجه را با دستیاران آموزشی در میان بگذارید.

۲. انجام دهید

یکی از توابع کاربردی برای تخصیص حافظه‌های پویا تابع realloc است . از این تابع می‌توانید برای تغییر مقدار حافظه‌ای که قبلا 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. لازم به ذکر است در صورتی که آرگومان ورودی اول برابر با NULL باشد این تابع مانند تابع malloc عمل می کنید.

۱) هدف نوشتن برنامه‌ای است که تا زمانی که در ورودی عدد 0 وارد نشده است، اعدادی را از ورودی دریافت نموده و در یک آرایه به صورت پویا ذخیره کند. سپس اعداد ذخیر شده در خروجی چاپ شوند.
۲) ابتدا یک اشاره‌گر به متغیر نوع int تعریف کنید و به آن مقدار اولیه بدهید.(مقدار اولیه باید برابر چه مقداری باشد؟)
۳) در یک حلقه‌ی for اعداد را از ورودی دریافت نموده و در صورتی که عدد وارد شده برابر 0 نبود با استفاده از تابع realloc طول آرایه‌ی اعداد را افزایش داده و عدد وارد شده در ورودی را ذخیره نمایید.
۴) با گرفتن عدد 0 در ورودی از حلقه خارج شده و با استفاده از یک حلقه اعداد را در خروجی چاپ نمایید.

نتیجه را به دستیاران آموزشی نشان دهید.

۳. انجام دهید (Free)

۱) هنگام کار با تخصیص حافظه‌ی پویا به دو نکته باید توجه نمود :
    ۱. لزوم آزاد کردن حافظه‌های گرفته شده
    ۲. عدم استفاده از حافظه‌ی آزاد شده
۲) 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 دقیقا چه اتفاقی می‌افتد؟ خروجی‌ها را توجیه کنید.

نتیجه را با دستیاران آموزشی در میان بگذارید.

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

۴. انجام دهید (Memory Leak)

حافظه‌های گرفته شده از سیستم هنگام اجرای برنامه حتما 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.

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

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

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

۲) سربرگ Processes را انتخاب کنید.

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

۴) با فشردن دکمه‌ی f5 به تعداد ۱۰ بار حلقه‌ی while را اجرا کنید.

۵) مقدار memory گرفته شده توسط برنامه را مجددا مشاهده کنید.

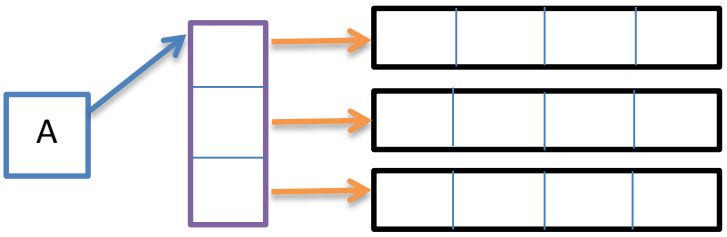
۶) چه نتیجه‌ای می‌گیرید؟

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 (اگر یادتان باشد!) با آرایه های چند بعدی هم کار کرده بودیم. حال می خواهیم همان آرایه ها را نیز به شکل پویا درست کنیم. یک آرایه ی دوبعدی، عملاً آرایه ای از آرایه های یک بعدی است. مثلاً int A[3][4] یک آرایه ی ۳ تایی از آرایه هایی به طول ۴ می باشد. شکل زیر را نگاه کنید:



پس اگر ما ابتدا یک آرایه از جنس int** به طول ۳ بسازیم که A سر آن باشد، و سپس هر کدام از خانه های آن آرایه را برابر با سر یک آرایه ی ۴ تایی قرار دهیم، یک آرایه‌ی دوبعدی ساخته‌ایم.. در حالت کلی اگر بخواهیم یک آرایه پویای دوبعدی از جنس int بسازیم به صورت زیر عمل می کنیم:

۵. انجام دهید!

قسمت های 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;
}
```

نتیجه را به دستیاران آموزشی نشان دهید.

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؟

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

ساختمان داده [1]:

ساختمان‌های داده روش‌های ذخیره اطلاعات در کامپیوتر با هدف دسترسی آسانتر و بهینه‌تر هستند. در زبان برنامه‌نویسی c می‌توانیم با توجه به نیازمان ساختمان داده تعریف کنیم به صورتی که هر ساختمان داده متشکل از چند متغیر است و پس از تعریف ساختمان داده می‌توان از آن به صورت یک متغیر استفاده کرد. برای درک بهتر این موضوع به مثال زیر توجه کنید.در مثال زیر ساختمان داده‌ای برای نگه‌داری مختصات دوبعدی ساخته‌ایم.

```
struct coordinate
{
      int x;
      int y;
};
```

بعد از تعریف این ساختار بیرون از بدنه main می‌توانیم مانند دیگر متغیرها، متغیری از نوع coordinate بسازیم. برای دسترسی به متغیرهای تعریف شده درون ساختار داده از علامت "." بعد از متغیر تعریف شده استفاده می‌کنیم. در صورتی که اشاره‌گری از نوع ساختمان داده‌ی تعریف شده تعریف کنیم برای دسترسی به متغیرهای تعریف شده‌ی درون آن از علامت "->" استفاده می‌کنیم. به مثال زیر توجه کنید.

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


struct coordinate
{
      int x;
      int y;
};


int main() {
```

[1] Data Structure

```
    coordinate a;
    coordinate *p;
    a.x = 25;
    p = &a;
    p->y = 32;
    return 0;
}
```

۶. انجام دهید!

می‌خواهیم از دانشجویانی که در کلاس مبانی کامپیوتر ثبت‌نام کرده‌اند لیستی تهیه کنیم. به همین منظور می‌خواهیم متغیری از نوع Student تعریف کنیم تا اطلاعات دانشجویان را در آن نگه‌داری کنیم.

۱) ساختمان داده‌ای از نوع Student تعریف کنید.

۲) در این ساختمان داده اطلاعات زیر باید نگه‌داری شوند. این اطلاعات را با متغیرهای مناسب تعریف کنید.

۱. نام
۲. نام خانوادگی
۳. شماره‌ی دانشجویی
۴. نمره‌ی میانترم
۵. نمره‌ی پایانترم
۶. نمره‌ی تکالیف

- نکته : برای تعریف متغیرهای نام و نام خانوادگی از آرایه‌ای از کاراکتر با طول محدود استفاده کنید. نیازی به استفاده از آرایه‌ی پویا نیست.
- نکته : نمرات می توانند اعشاری باشند.

۳) از ساختمان داده‌ی تعریف شده یک متغیر از نوع ایستا بسازید و با استفاده از ورودی اطلاعات آن را مقداردهی کنید.

۴) با استفاده از تابع malloc یک متغیر از نوع ساختمان داده‌ی تعریف شده، ایجاد کنید و با استفاده از ورودی اطلاعات آن را مقداردهی کنید.

۵) اطلاعات دو دانش آموزی که از ورودی گرفته‌اید را چاپ نمایید.

نتیجه را به دستیاران آموزشی نشان دهید.

نکته: کد ساختمان داده‌ی تعریف شده را جهت استفاده در آزمایش‌های بعدی نگه دارید.