# ساختمان داده‌های محض

قاسم مهدور (mahdevar@ibb.ut.ac.ir)

# فهرست مطالب

- انواع داده محض ( Abstract Data Types (ADT)
- پیاده سازی آن با استفاده از آرایه و اشاره‌گرها
- فهرست (list)



## انواع داده محض (Abstract Data Types (ADT))

- تعریف:

**یک مجموعه از اشیا به همراه اعمال تعریف شده روی آنها یک نوع داده محض را ایجاد می‌کنند.**

- برای نمونه:

1. اعداد (numbers) با اعمالی مانند جمع، ضرب، و ...
2. فهرست(list) با اعمالی مانند یافتن یک عضو خاص، افزودن عضوی جدید، و ...
3. صف(queue)، با اعمالی مانند افزودن به انتهای صف یا حذف عضو ابتدایی صف
4. و ...



## انگیزه: چرا انواع داده محض را مطالعه کنیم؟

- اگر قرار باشد برنامه‌ای برای یک کتابخانه بنویسید، باید فهرستی از کتب را ذخیره و در آنها جستجو نمود.
- برنامه آموزش دانشگاه با فهرستی از اسامی دانشجویان و استادان سر و کار دارد.

**فهرستی از ماهیت‌ها در بسیاری از برنامه‌ها وجود دارد.**

**اگر یک بار فهرست به صورت کلی (generic) پیاده سازی شود، می‌توان به کرات از آن استفاده نمود.**

## انگیزه: چرا انواع داده محض را مطالعه کنیم؟

- پنجره معروف ...Save As در بسیاری از برنامه‌ها وجود دارد. آیا می‌توانیم به سادگی از این پنجره در برنامه‌ای که می‌نویسیم استفاده کنیم؟
- آیا برای نمایش یک عکس با قالب jpg باید از صفر شروع کرد و دوباره چرخ را اختراع نمود؟ یا می‌توان آن برنامه نوشته شده را به آسانی برنامه خود افزود.

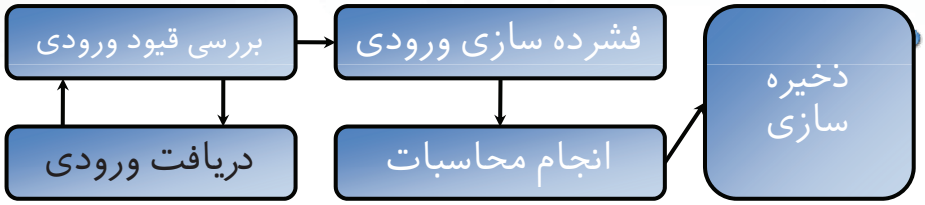
**اگر قطعات برنامه‌ای که می‌نویسیم قابل حمل باشند می‌توان بعدها از آنها استفاده نمود یا حتی آنها را به دیگران فروخت.**

**بهتر است پیچیدگی این قطعات درون آنها باقی بماند و کاربر نهایی را گیج ننماید.**



## چرا انواع داده محض را مطالعه می‌کنیم؟

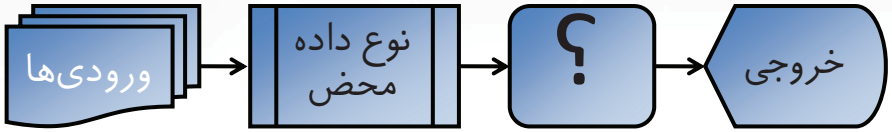
- پیمانه‌ای نمودن برنامه نویسی؛ که مزایای زیر را دارد:
  - به راحتی می‌توان برنامه را اشکال زدایی و نگه‌داری نمود، و
  - به راحتی می‌توان قسمتی از برنامه را تغیر داد.
- امکان استفاده مجدد از نوع داده‌های ساخته شده.






## چرا انواع داده محض را مطالعه می‌کنیم؟

- بسته بندی و پنهان سازی اطلاعات؛ که مزایای زیر را دارد:
  - پنهان نمودن جزییات پیاده سازی،
  - دسترسی فقط از طریق توابع مجاز (دوست!) امکان دارد، و
  - قابلیت حمل و نقل آسان.






## فهرست (List)

- تعریف فهرست
- اعمال قابل انجام روی فهرست
- پیاده سازی با استفاده از آرایه و فهرست پیوندی

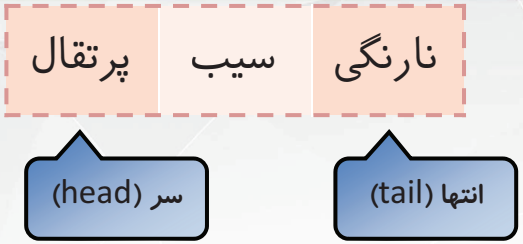
## تعریف نوع داده فهرست (List)

**مجموعه مرتب از اقلام داده**

- مثال‌ها:
  1. پرتقال، سیب، نارنگی
  2. $X_0, X_1, ..., X_n$
- اندازه فهرست خالی 0 است.
- موقعیت عضو i ام، یعنی Xi همان i است.
- به اولین عضو فهرست سر(head) و به آخرین آنها ته(tail) می‌گوییم.



## اندازه فهرست و موقعیت در آن




- اندازه فهرست، تعداد اعضای داخل آن است.
- اندازه فهرست خالی 0 است.
- موقعیت عضو i ام، یعنی $X_i$ همان i است.
- به اولین عضو فهرست سر(head) و به آخرین آنها ته (tail) می‌گوییم.
- نمونه: فهرستی با سه عضو.



## اشیا داخل فهرست

- یک فهرست از
  - اعداد صحیح، اعشاری، دودویی، ...
  - رشته‌های حرفی، یا
  - انواع داده‌های دیگر(؟)

  را می‌توان ساخت.



## اعمالی که می‌توان روی فهرست انجام داد

- ایجاد (create)،
- چاپ (print)،
- اضافه نمودن یک عضو (insert)،
- حذف یک عضو (delete)،
- جستجو برای یافتن یک عضو (find)،
- یافتن عضو بعدی (next)،
- یافتن عضو قبلی (previous)، و
- یافتن عضو k ام (retrieve).

بخش اول

# آرایه (=پیاده سازی خطی فهرست)

- نحوه ذخیره سازی آرایه در حافظه
- پیاده سازی اعمال متفاوت روی آرایه



## مثال

- یک فهرست از اعداد صحیح

1, 20, 12, 5, 99, 11

- نتیجه اجرای برخی از اعمال روی این فهرست

find (20) → 1

retrieve(4) → 99

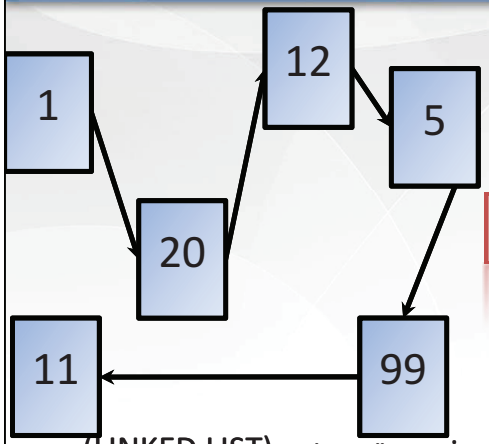
insert (7, 2) → 1, 20, 7, 12, 5, 99, 11

delete(99) → 1, 20, 7, 12, 5, 11

می‌توان تابع delete را به گونه‌ای نوشت که اندیس عضو مطلوب را دریافت کند نه محتوی آن را.



## روش‌های پیاده سازی فهرست




فهرست پیوندی(LINKED LIST)

( که می‌تواند تک پیوندی یا دوگانه باشد)

| 1 | 20 | 12 | 5 | 11 | 99 |
|---|---|---|---|---|---|

آرایه (ARRAY)



## آرایه

| 1 | 20 | 12 | 5 | 11 | 99 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|

شش عنصر آرایه پر شده است. این آرایه حداکثر می‌تواند 11 عنصر داشته باشد.

- اندازه ثابت است. نمی‌توان بعد از ساخت اندازه آن را تغییر داد.
- فضای زیادی را از دست می‌دهیم. به صورت میانگین نیمی از آرایه خالی است.

## نحوه ذخیره سازی آرایه در حافظه

| مقدار | موقعیت حافظه |
|---|---|
| 1 | 2342 |
| 20 | 2344 |
| 12 | 2344 |
| 5 | 2346 |
| 99 | 2348 |
| 11 | 2350 |
| ? | 2352 |
| ? | 2354 |
| ? | 2356 |
| ? | 2358 |
| ? | 2360 |

- در اینجا بایت 2342 ام حافظه موقعیت شروع فضای تخصیص داده شده به آرایه است.
- اندازه آرایه 11 است.
- معایب این نحوه ذخیره سازی
  - طول آن از قبل محدود شده است و
  - (فعلاً) فضای زیادی را از دست داده‌ایم.



## پیاده سازی ساختار آرایه در C

```
struct Array
{
    int n;
    int *Data;
};
void main()
{
    Array A;
}
```

توابع insert، delete و ... را به گونه‌ای خواهیم نوشت که n همواره اندازه آرایه را داشته باشد.

در اینجا قصد ایجاد یک فهرست از اعداد int را داریم.



## ایجاد آرایه

حتماً عددی به عنوان بیشینه اندازه ممکنه باید وجود داشته باشد. این یک محدودیت بزرگ برای آرایه است.

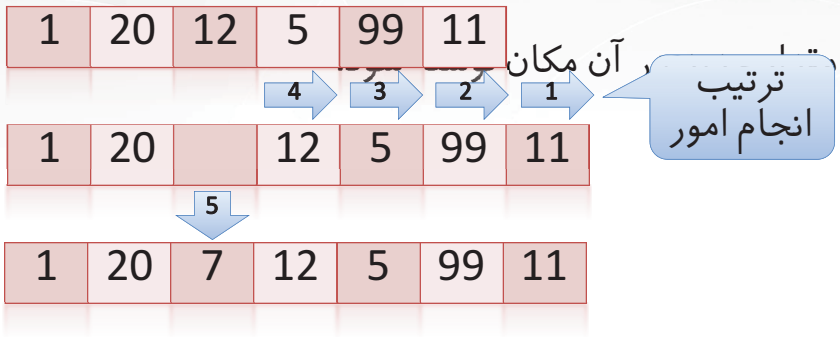
```
#include "stdafx.h"
#include "stdlib.h"
struct Array
{
    int n;
    int *Data;
};
int Create(Array *A, int L)
{
    A->Data =(int *) malloc (L * sizeof(int));
    A->n=0;
    return 1;
}
int main(int argc, char* argv[])
{
    Array A;
    Create (&A, 10);
    return 0;
}
```



## چگونگی درج عنصری جدید در آرایه

- برای اضافه نمودن گره‌ای جدید باید

1. عناصر از آن موقعیت تا انتها یک واحد به سمت انتها منتقل شوند،
2. [illegible] آن مکان




ترتیب انجام امور

## الگوریتم درج عنصری جدید در آرایه

```
int Insert(Array *A, int i, int d)
  {
     if(A->n==MAX)return 0;
     for(int t = A->n; t > i; t--)
           A->Data[t] = A->Data[t - 1];
     A->Data[i] =
     A->n++;
     return 1;
  }
```

**واضح است که پردازنده برای انجام هر یک از محاسبات مورد نیاز الگوریتم زمانی را نیاز دارد. در واقع همین اعمال تعیین می‌کنند که الگوریتم چه‌قدر طول بکشد.**



آیا الگوریتم نوشته شده سریعترین الگوریتم ممکن است؟ می‌توانید پاسخ بلی یا خیر خود را اثبات کنید؟

اگر در فهرست اسامی 10000 دانشجو وجود داشته باشد و بخواهیم دانشجو 10001 را اضافه کنیم، چه میزان وقت مورد نیاز است؟ اصلاً حافظه 1GB من کافی است؟

با فراگیری تحلیل الگوریتم‌ها می‌توان به پرسش‌های از این دست پاسخ داد.



## الگوریتم درج عنصری جدید در آرایه: تحلیل

- اگر بخواهیم به انتهای آرایه عنصری جدید اضافه کنیم، حلقه for الگوریتم گفته شده اصلاً اجرا نمی‌شود و الگوریتم سریعاً به پایان می‌رسد. این بهترین حالت ممکن است.
- برعکس، اگر قرار باشد عنصری به ابتدای فهرست افزوده شود حلقه for باید به اندازه فهرست، یعنی n بار تکرار شود. این بدترین اتفاق ممکن است، زیرا بیشترین زمان ممکن را می‌طلبد.
- اما به طور میانگین عضو جدید در میانه فهرست درج می‌گردد. در این حالت حلقه for باید n/2 بار تکرار شود. این حالت میانگین زمان اجرای الگوریتم را نیاز دارد و معدل حالات است.



## نحوه حذف یک عنصر از آرایه

- برای حذف یک عنصر باید

1. عناصر از آن موقعیت تا انتها یک واحد به سمت ابتدا منتقل شوند.

| 1 | 20 | 7 | 12 | 5 | 99 | 11 |
|---|---|---|---|---|---|---|

- برای نمونه در اینجا می‌خواهیم عدد 7 را از فهرست حذف کنیم.

1 2 3 4

ترتیب انجام امور

| 1 | 20 | 12 | 5 | 99 | 11 |
|---|---|---|---|---|---|

## الگوریتم حذف یک عنصر از آرایه

```
int Delete(Array *A, int i)
    {
     if(A->n==0) return 0;
     for(int t = i; t < A->n; t++)
           A->Data[t] = A->Data [t + 1];
     A->n--;
     return 1;
    }
```



## الگوریتم حذف یک عنصر از آرایه: تحلیل

- اگر قرار باشد نخستین عنصر فهرست را حذف کنیم حلقه for باید به اندازه فهرست منهای یک، یعنی n – 1 بار تکرار شود. این بدترین اتفاق ممکن است، زیرا بیشترین زمان ممکن را می‌طلبد.
- برعکس، اگر بخواهیم از انتهای آرایه عنصری را حذف کنیم، حلقه for الگوریتم گفته شده اجرا نمی‌شود الگوریتم سریعاً به پایان می‌رسد. این بهترین حالت ممکن است.
- اما به طور میانگین عضوی که حذف می‌شود در میانه فهرست قرار دارد. در این حالت حلقه for باید n/2 بار تکرار شود. این حالت میانگین زمان اجرای الگوریتم را نیاز دارد و معدل حالات است.



اصطلاحات سریعاً، بهترین، بیشترین زمان ممکن، بدترین حالت، و میانگین زمان اجرا تعریف دقیقی (formal) دارند که در مبحث تحلیل الگوریتم‌ها به آن خواهیم پرداخت.



## یافتن عضو k ام در یک آرایه

```
int Retrieve (Array *A, int k)
  {
    return A->Data[k];
  }
```

## یافتن عضوی خاص در یک آرایه

```
int *find (Array *N, int k)
 {
    int i=0;
    while(i < Array->n && A->Data[i] != k)
         i++;
    return I;
 }
```



به نظر شما کدام یک از اعمال زیر زمان بیشتری نیاز دارد: درج عنصری جدید حذف یک عنصر یا یافتن عنصری خاص؟

برای پاسخ گویی به این سوال باید با مفهوم پیچیدگی و تحلیل الگوریتم‌ها آشنا بود.



## آرایه – مزایا و معایب

- مزایا
  - سادگی پیاده سازی و
  - سرعت بالا اگر که به انتهای فهرست اضافه شود.
- معایب
  - از دست دادن حافظه(؟) و
  - اگر اضافه و حذف نمودن در داخل فهرست باشد، سرعت پایین خواهد نمود.



## آرایه – زمان مورد نیاز اعمال مختلف

- ایجاد (create) ← O(n)
- چاپ (print)، ← O(n)
- اضافه نمودن یک عضو (insert)، ← O(n) *؟
- حذف یک عضو (delete)، ← O(n) *؟
- جستجو برای یافتن یک عضو (find)، ← O(n) **
- یافتن عضو بعدی (next)، ← O(1)
- یافتن عضو قبلی (previous)، و ← O(1)
- یافتن عضو k ام (retrieve). ← O(1)

- * البته به طور میانگین n/2 جابجای مورد نیاز است.
- ** البته به طور میانگین n/2 مقایسه مورد نیاز است.