# اصول ساختمان داده ها در c++

ساختمان داده ها شاخه ای از مهندسی نرم افزار است که به مطالعه انواع ساختارهای داده و نیز الگوریتم های مربوط به آنها می پردازد.

گلزاری پور

آموزشکده فنی و حرفه ای شهید بهشتی کرج

## ساختمان داده

ساختمان داده ها شاخه ای از مهندسی نرم افزار است که به مطالعه انواع ساختارهای داده و نیز الگوریتم های مربوط به آنها می پردازد.

### ساختار داده[1]

مجموعه ای است ساختمند و نامدار از داده های مرتبط به هم را ساختارداده گویند.

### پیمایش

دستیابی به تمام عناصر یک ساختار داده به طوری که هر عنصر از آن تنها یک بار مورد دستیابی قرار گیرند عمل پیمایش نامیده می شوند.

### الگوریتم[2]

الگوریتم مجموعه ای متناهی از دستورالعمل هاست که روش حل مسئله ای را بیان می کند و داری شرایط و مشخصات زیر است:

1. ورودی : یک الگوریتم می تواند داری چندین ورودی باشد . همچنین می تواند هیچ ورودی نداشته باشد.
2. خروجی : الگوریتم باید دارای حداقل یک نتیجه خروجی باشد.
3. وضوح : هر دستورالعمل درالگوریتم باید بدون ابهام باشد.
4. خاتمه پذیری : الگوریتم باید پس از طی مراحل محدودی خاتمه یابند.
5. انجام پذیری : هر دستورالعمل در الگوریتم باید قابل انجام باشد.

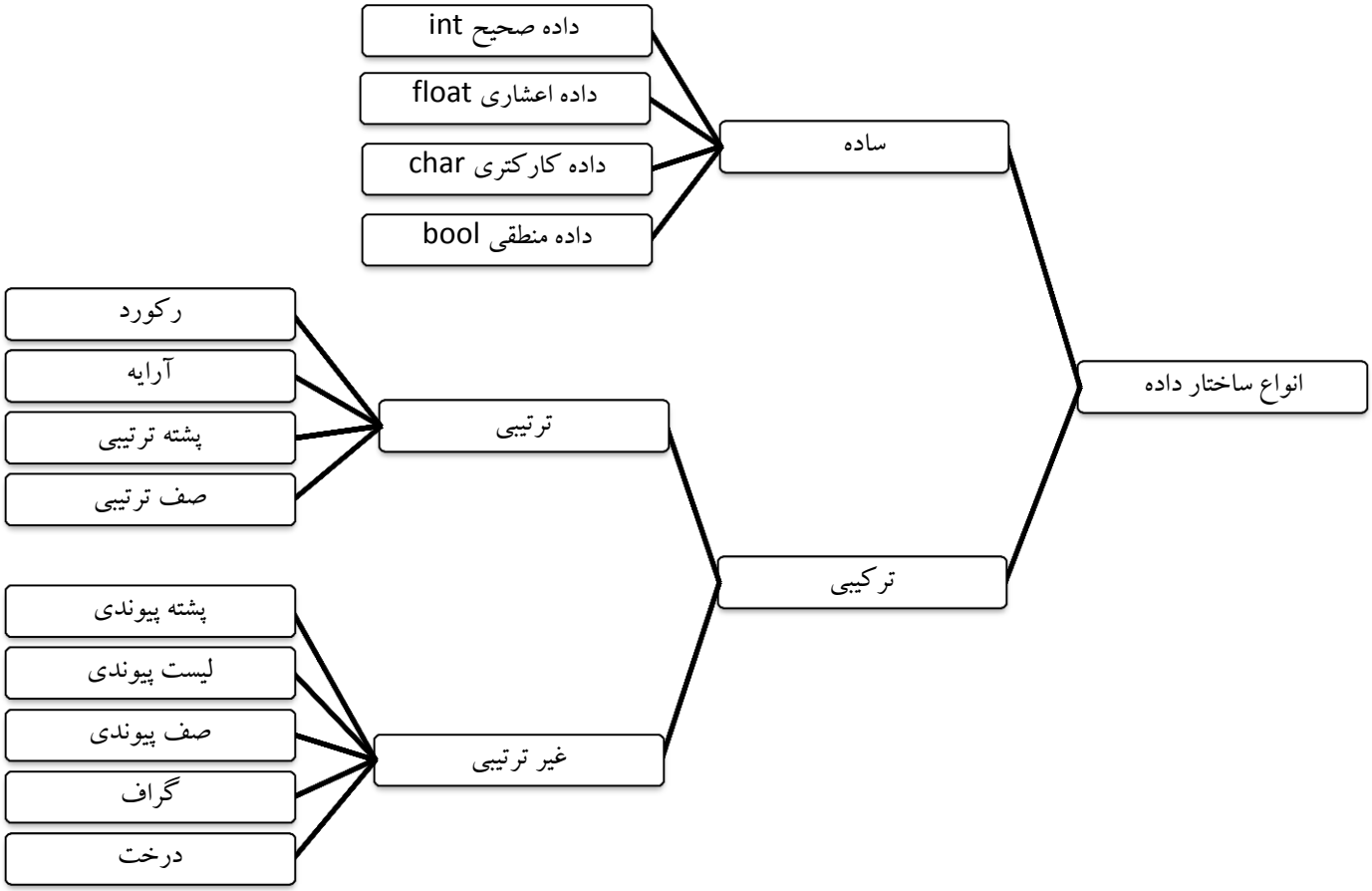
## مراحل کلی تولید نرم افزار

به طور کلی فرآیند تولید یک محصول نرم افزاری شامل مراحل شش گانه زیر است:

1. نیازمندیها
2. تجزیه و تحلیل
3. طراحی
4. پیاده سازی
5. تست و بازبینی
6. نگهداری

[1] Data structure
[2] Algorithm

به این مراحل اصطلاحا چرخه زندگی سیستم [1] گفته می شود که یک فرآیند تکاملی است .




ساختارهای داده ای ساده شامل داده های صحیح ، کاراکتری ، اعشاری و منطقی هستند . به عنوان مثال در زبان c++ به ترتیب انواع داده های bool ، float ، char، int وجود دارد ساختارهای داده ترکیبی همانطور که از نامشان پیداست از ترکیب و سازماندهی ساختارهای ساده پدید می آید.

معیارهای انتخاب ساختار داده: معیارهای مهم در انتخاب ساختار داده را می توان به ترتیب اهمیت به صورت زیر بیان کرد

1. امکان پذیری : نوشتن الگوریتم های لازم بر روی ساختار داده با توجه به اهداف مسئله امکان پذیر باشد.
2. سرعت اجرای الگوریتم : ساختاری انتخاب شود که الگوریتم های اعمال شده بر روی آن دارای سرعت اجرای مناسبی باشد.

[1] System life cycle

3. حافظه مصرفی : حافظه اشغال شده توسط داده های ساختار بهینه باشد.
4. سازگاری با ماهیت مسئله : ساختاری انتخاب شوند که با منطق برنامه و ماهیت داده ها ی مسئله هم گونی داشته باشند .

مثال : برنامه ای را در نظر بگیرید که هدف آن ذخیره اطلاعات تعداد نامشخصی از دانشجویان و انجام عملیات اضافه ، حذف و جستجو باشند برای این برنامه ساختارهای داده مختلفی به شرح زیر قابل تعریف است :

- ✓ ساختار داده ساده که شرط اول ( امکان پذیری ) را دارا نیست .
- ✓ بکارگیری آرایه ای از رکورد های دانشجو .
- ✓ استفاده از لیست پیوندی (این ساختار تمام معیارهای لازم به عنوان یک ساختار داده مناسب را دارد .)

## اعمال اصلی در ساختار داده

ساختار داده ابزاری است جهت نمایش و مدیریت داده ها ، یک جنبه مهم جهت تحقق مدیریت داده ها انجام عملیات بر روی داده های ساختار است.

اعمال اصلی در ساختارهای داده ای به شرح زیر است:

1. ایجاد[1] ساختار
2. درج[2] عنصر جدید
3. حذف[3] عنصر
4. به هنگام سازی[4] عنصر
5. جستجوی[5] عنصر
6. پیمایش[6] ساختار
7. حذف ساختار

[1] Create
[2] Insert
[3] Delete
[4] Update
[5] Search
[6] Traversal

## معیارهای سنجش کارایی الگوریتم

کارایی الگوریتم به دو عامل اصلی زیر بستگی دارد:

1. پیچیدگی حافظه[1] : مقدار حافظه مورد نیاز جهت اجرای کامل الگوریتم
2. پیچیدگی زمانی[2] : مدت زمان لازم جهت اجرای کامل الگوریتم

### توجه

انتخاب ساختار داده ارتباط نزدیکی با پیچیدگی حافظه و پیچیدگی زمانی الگوریتم دارد.

## زمان واقعی

برای بیان پیچیدگی زمانی الگوریتم دو روش وجود دارد:

یکی اینکه محاسبه پیچیدگی الگوریتم به صورت دقیق از لحظ زمان اجرای کامل برنامه مورد نظر بر روی کامپیوتر انجام شود این روش علاوه بر مشکل بودن به نوع ماشینی که الگوریتم برروی آن ایجاد شود بستگی دارد . مثال : پردازنده پنتیوم four، tree، two به حافظه ربط دارد به عوامل سخت افزاری نرم افزاری دیگری مانند حافظه نهان پردازنده ، میزان حافظه اصلی ، سرعت کامپایلر و موارد دیگر بستگی دارد.

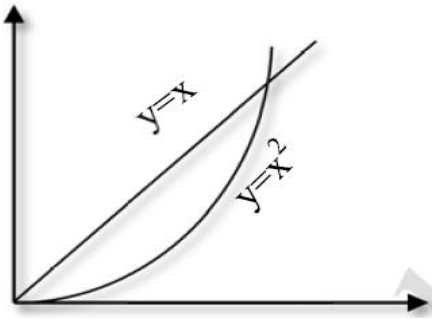
در نتیجه معیار صحیحی برای بیان زمان اجرای الگوریتم وجود ندارد . مگر اینکه خود را محدود به عوامل سخت افزاری و نرم افزاری خاصی نمائیم . در اینگونه موارد معمولا از زمان سنج برای محاسبه زمان اجرا استفاده می شود . در اغلب زبانهای برنامه نویسی تابعی جهت دریافت زمان سیستم وجود دارد. تابع زمانی روش دیگر این است که پیچیدگی الگوریتم را به صورت تابعی از ورودی بیان کنیم در این روش بدون اینکه نیاز به محاسبه دقیق زمان اجرای الگوریتم بر روی یک پلتفرم [3] سخت افزاری و نرم افزاری خاص باشد معیاری تئوریک بر پایه مفاهیم ریاضی برای مقایسه کارایی الگوریتم ها ارائه می شود که به مستقل از نوع کامپیوتری است که برای اجرای الگوریتم استفاده میشود.

[1] Space complexity
[2] Time complexity
[3] Platform

## تابع زمانی

تابعی است که زمان اجرای یک الگوریتم را بر حسب ورودی های آن بیان می کند.

تابع زمانی یک تابع ریاضی است که می تواند یک متغیره یا چند متغیره باشد تعداد متغیرهای تابع زمانی برابر با تعداد ورودی های الگوریتم است . این تابع رفتار زمان الگوریتم را بر حسب ورودی های آن نشان میدهد ، به عبارت دیگر نشان می دهد رشد زمان اجرای الگوریتم چه رابطه ای با ورودی های آن دارد.




## تفاوت برنامه و الگوریتم

برنامه همیشه باید قابل اجرا باشد و الگوریتم در یک نقطه ای باید پایان داشته باشد.

## مرحله

هر دستور از برنامه که انجام آن زمان ثابتی طول می کشد به عنوان یک مرحله از برنامه در نظرگرفته می شود.

## رتبه [1] زمانی

رتبه زمانی ، نمادی برای بیان ماهیت تابع زمانی الگوریتم است.

## مثال

الگوریتم زیر مجموع عناصر یک آرایه به طول n را محاسبه می کند می خواهیم با تعیین تعداد مراحل برنامه[2] تابع زمانی آن را معین می کنیم.

```
float sum ( float a[ ] , int n )
{
float s = 0 ;
for ( int i = 0 ; i < n ; i++ )
s = s + a[ i ] ;
return s;
}
```

## جواب

$0 + 1+( n + 1) + n +1$

[1] Order
[2] Program steps

روشن است که رتبه زمانی دقت تابع زمانی را ندارد . رتبه زمانی یک معیار تخمینی برای بیان رفتار تابع زمانی الگوریتم است و چگونگی رشد زمان الگوریتم را بر حسب ورودی های آن نشان می دهد با داشتن تابع زمانی یک الگوریتم به راحتی می توان رتبه زمانی آن را تعیین کرد ولی با داشتن رتبه زمانی نمی توان تابع زمانی رابه دست آورد.

**مثال**

تابع زمانی $T(n) = 2n+3$

رتبه زمانی $F(n) = n$

**پیچیدگی یک الگوریتم از لحاظ فضا و زمان به دو قسمت بستگی دارد**

1. **قسمت ثابت** : به قسمتهایی از الگوریتم مربوط می شود که مستقل از اندازه ورودی الگوریتم است.
2. **قسمت متغیر** :به قسمتهایی از الگوریتم مربوط می شود که اندازه آن بستگی به اندازه ورودی الگوریتم دارد.

## میزان حافظه (پیچیدگی فضای لازم)

فضای مورد نیاز یک برنامه شامل موارد زیر است :

- نیازمندیهای فضای ثابت: این مطلب به فضای مورد نیازی که به تعداد و اندازه ورودی و خروجی بستگی ندارد، اشاره دارد. نیازمندی های ثابت شامل فضای دستورالعمل (فضای لازم برای ذخیره کد برنامه)، فضای لازم برای متغیر ساده، متغییرهای ساختاری با اندازه ثابت و ثابت ها می باشند.
- نیازمندیهای فضای متغیر: این مورد شامل فضای مورد نیاز متغیرهای ساخت یافته ای است که اندازه آن بستگی به نمونه I از مساله ای که حل می شود، دارد.

# میزان حافظه (پیچیدگی فضای لازم)

$$S(P) = c + S_p(I)$$

نیازمندیهای فضای متغیر

نیازمندیهای فضای کل

نیازمندیهای فضای ثابت

# پیچیدگی زمانی

زمان T(n) برنامه عبارتست ازمجموع زمان کامپایل و زمان اجرای برنامه.

زمان کامپایل مشابه اجزای فضای ثابت است زیرا به خصیصه های نمونه بستگی ندارد.

در ارزیابی یک الگوریتم دو فاکتور مهمی که باید مورد توجه قرار گیرد یکی حافظه مصرفی و دیگری زمان مصرفی الگوریتم است. یعنی الگوریتمی بهتر است که فضا و زمان کمتری را بخواهد. البته غالباً در الگوریتم های این کتاب فاکتور زمان مهمتر از فاکتور فضا می باشد. از آنجا که کامپایل برنامه فقط یکبار صورت می گیرد لذا در مورد زمان، فقط زمان اجرای برنامه را در نظر گرفته و از زمان کامپایل صرف نظر می کنیم.

به طور کلی زمان اجرای یک الگوریتم با افزایش اندازه ورودی (n) زیاد می شود و زمان اجرا با تعداد دفعاتی که عملیات اصلی انجام می شود تناسب دارد. بنابراین بازدهی الگوریتم را با تعیین تعداد دفعاتی که یک عمل اصلی انجام می ود به عنوان تابعی از ورودی تحلیل می کنیم.

# پیچیدگی زمانی

برای محاسبه تابع T(n) برای یک الگوریتم موارد زیر را باید در محاسبات در نظر بگیریم:

- زمان مربوط به اعمال جایگزینی که مقدار ثابت می باشند.
- زمان مربوط به انجام اعمال محاسبات که مقدار ثابتی دارند.
- زمان مربوط به تکرار تعدادی دستور یا دستورالعمل(حلقه ها)
- زمان مربوط به توابع بازگشتی

# پیچیدگی زمانی

مثال: تعداد کل مراحل برنامه زیر را محاسبه کنید(این تابع جمع عناصر یک آرایه در زبان C را محاسبه می کند).

```
float   sum(float   list[], int n) ---> 0
{ ---------------------------------> 0
  float   s=0; ----------------------> 1
  int   i; --------------------------> 0
  for(i=0; i<n; i++) ----------------> n+1
    s=s+list[]; -------------------> n
  return s; -------------------------> 1
} ---------------------------------> 0
                                      2n+3
```

# پیچیدگی زمانی

مثال: تعداد کل مراحل قطعه برنامه زیر را محاسبه کنید.

```
for  i:=1 to n do ──────→ n+1
     x:=x+1; ──────────→ n
```

$$2n+1$$

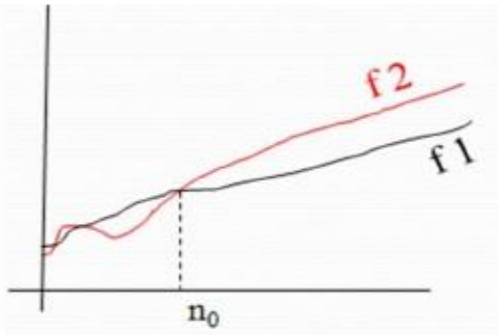
مثال: تعداد کل مراحل قطعه برنامه زیر را محاسبه کنید.

```
for  i:=1 to n do ──────→ n+1
for  j:=1 to n do ──────→ n(n+1)
     x:=x+1; ──────────→ n*n
```

$$2n^2+2n+1$$

# نماد (O) Big-oh

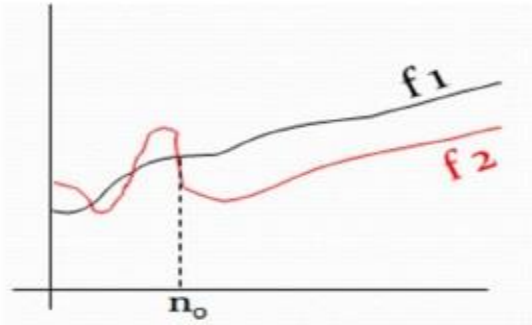
گوئیم $T(n) \in O(g(n))$ اگر و فقط اگر ثابت C و ثابت صحیح $n_0$ وجود داشته باشند که برای همه مقادیر $n>=n_0$ داشته باشیم $T(n)<=Cg(n)$.




قضیه: اگر $f(n)=a_m n^m+...+a_1 n+a_0$ باشد، آنگاه $f(n)=O(n^m)$ خواهد بود.

# نماد اُمگا (Ω)

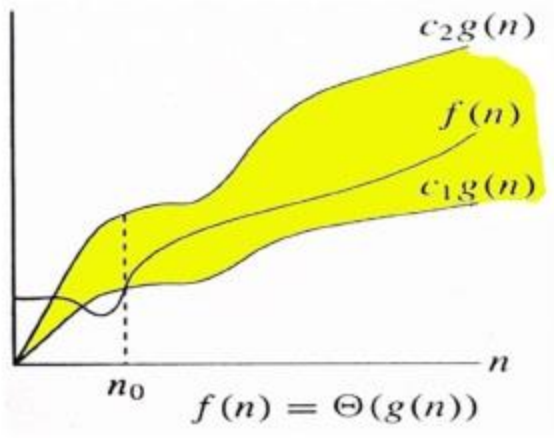
گوئیم $T(n) \in \Omega(g(n))$ اگر و فقط اگر ثابت C و ثابت صحیح $n_0$ وجود داشته باشند که برای همه مقادیر $n>=n_0$ داشته باشیم $T(n)>=Cg(n)$ .




قضیه: اگر $f(n)=a_m n^m+...+a_1 n+a_0$ باشد، آنگاه $f(n)=\Omega(n^m)$ خواهد بود.

# نماد تتا ($\theta$)

گوییم $T(n) \in \theta(g(n))$ اگر و فقط اگر ثابت های $C_1$ و $C_2$ و ثابت صحیح $n_0$ وجود داشته باشند به طوری که برای همه مقادیر $n>=n_0$ داشته باشیم $C_1g(n)=<T(n)<=C_2g(n)$ .




قضیه: اگر $f(n)=a_mn^m+...+a_1n+a_0$ باشد، آنگاه $f(n)= \theta(n^m)$ خواهد بود.

# الگوریتم های بازگشتی

الگوریتم های بازگشتی الگوریتم هایی هستند که داخل خود چند بار فراخوانی می شوند و بعد از تعدادی فراخوانی به مقداری ثابت می رسند، سپس شروع به محاسبه مقدار تابع می نمایند.

مزایا:

- ✓ سادگی پیاده سازی
- ✓ سادگی درک الگوریتم

معایب:

- ✖ اتلاف حافظه
- ✖ سرعت اجرایی کمتر