ساختمان داده در C++

۱- فصل اول:

نوع داده انتزاعی(مجرد): ADT : وقتی در برنامه ایی به نوعی داده نیاز باشد که در آن زبان وجود ندارد برنامه نـویس باید نوع مورد نظرش را ایجاد کند. نوع داده ایی را که برنامه نویس ایجاد می کند نوع داده انتزاعی می گویند.
ADT یک مدل ریاضی است که عملیاتی بر روی آن مدل تعریف مـی شـود. هرنـوع داده متـشکل از چنـد مقـدار و مجموعه ای از عملیات بر روی آنها است.
مانند نوع داده int که در زبان C عملیاتهایی مانند > < / = - + * و غیره برای آنها تعریف شده است.
در این درس انواع مختلفی از ADT ها را بوجود می آوریم مانند آرایه ها و ماتریس ها و درخت ها و غیره

۱-۱- نحوه تجزیه و تحلیل و سنجش کارایی یک برنامه:

تعریف : میزان حافظه یا پیچیدگی فضای یک برنامه مقدار حافظه مورد نیاز برای اجرای کامـل یـک برنامـه اسـت.
مقدار زمان یا پیچیدگی زمانی یک برنامه مقدار زمانی از کامپیوتر است که برای اجرای کامل برنامه لازم است.

۱-۱-۱- *میزان حافظه (پیچیدگی حافظه)*: فضای مورد نیاز برای یک برنامه شامل موارد زیر است:

- بخش ثابت که مستقل از بعضی خصیصه های ورودی و خروجی مانند تعداد وو اندازه است. این بخش شامل فضای دستورالعمل (کد برنامه) ، فضای لازم برای متغیرهای ساده و ثابت ها می باشد
- بخش متغیر که شامل فضای مورد نیاز متغیرهای ساختاری برنامه که اندازه آن بستگی به نمونه مسئله ایی که حل می شود، دارد و فضای لازم برای متغیرهای مرجع که اندازه آنها نیز به خصیصه های نمونه بستگی دارد و فضای لازم برای پشته بازگشتی می باشد.

۱-۱-۲-*پیچیدگی زمانی*:

زمان برنامه (T(P)) مجموع زمان کامپایل و زمان اجرای برنامه است. زمان کامپایل برای یک برنامه فقط یک بار رخ می دهد در ضمن به خصیصه های نمونه بستگی ندارد. بنابراین زمان اجرا یک برنامه T_p مورد بررسی قرار می گیرد.
برای بدست آوردن زمان اجرای برنامه راه های مختلفی وجود دارد مثلا تعداد جمع هـا و ضـرب هـا و تقـسیمات و تفریقات یک برنامه را بدست آوریم و زمان را به صورت زیر محاسبه کنیم:

$$T_p= C_1ADD(n) + C_2Sub(n) + C_3Mul(n) + C_4Div(n)$$

که n نشان دهنده مشخصه های نمونه است. بدست آوردن چنین فرمول دقیقی یک کار غیر ممکن است زیرا زمان مورد نیاز برای جمع و ضرب و تفریق و تقسیم به عوامل متعددی بستگی دارد. در ضمن بین کامپیوتر های مختلف نیز متعدد است.
روش دیگری که برای بدست آوردن پیچیدگی زمانی یک برنامه استفاده می شود این است که تعداد مراحل برنامـه را بشماریم. پیچیدگی زمانی یک برنامه را بر اساس تعداد مراحل آن بدست می آوریم.
برای بدست آوردن تعداد مراحل یک برنامه با توجه به نوع دستورات عمل می کنیم. مثلا دستورات انتـساب را یـک مرحله محسوب می کنیم. دستورات تکرار را با توجه به تعداد تکرارشان محاسبه می کنیم. دستورات تعریفی را جزء مراحل محسوب نمی کنیم. شرط **IF** را یک مرحله محسوب می کنیم.

تعداد مراحل فراخوانی یک تابع مساوی مجموع تعداد مراحل فراخوانی هر تابع است.

یک از روشهایی که برای محاسبه تعداد مراحل یک برنامه استفاده می شود استفاده از متغیری به نـام count اسـت که در ازای هر مرحله Count++ می شود.

مثال:

```
Float Sum(Float  *A , Const  int  n)
{
  Float  S=0;    Int  i;
   For (I=0 ; I<n ; I++)
      S+ = A[ I ];
   Return (s);
}
```

```
Float Sum(Float  *A , Const  int  n)
{
  Float  S=0;    Int  i;
     Count ++;
  For (I=0 ; I<n ; I++) {
       Count ++;
        S+ = A[ I ];
      Count++;
    }
Coumt++;  // For Last time of for
Return (S);
Count++;
}
```

که پیچیدگی زمانی آن **2n+3** است.

مثال)جمع دو ماتریس

که پیچیدگی زمانی آن

2mn+2n+1 می شود.

```
For ( I=0 ; I<m ; I++)
  For  ( j=0 ; j<n ; j++)
    C[I][j]=A[I][j]+B[I][j];
```

```
For ( I=0 ; I<m ; I++){
   Count++;
  For  ( J=0 ; J<n; J++) {
     Count++;
     C[I][j]=A[I][j]+B[I][j];
      Count++;
   }
  Count++; // For Last time of for
}
Count++; // For last time of for
```

البته این روش نیز چندان دقیق نیست. مثلا ما دستور **A:=5*2+4 –B** رایک مرحله می گیریم و A:=8 را نیز یـک مرحله می گیریم در صورتیکه از نظر زمانی با هم متفاوتند.

تعداد مراحل *لزوما* پیچیدگی جمله را منعکس نمی کند.

ولی در یک برنامه بزرگ می توان تعداد مراحل را با کمی اغماض پیچیدگی برنامه دانست و می تـوان برنامـه هـا را براساس پیچیدگی زمانی آنها با هم مقایسه کرد.

روش دیگر در کتاب بحث شده است که به روش **S/e** نام برده می شود که شباهت زیادی به روش **Count** دارد.

تحقیق : پیچیدگی الگوریتم فیبوناچی **4n+1** است چرا؟

هدف از تعیین تعداد مراحل این است که بتوانیم پیچیدگی زمانی دوبرنامه که عمـل مـشابهی را انجـام مـی دهنـد مقایسه کنیم و همچنین بتوانیم میزان افزایش زمان اجرا را وقتی مشخصات نمونه تغییر می کنند پیشگویی کنیم.

مثلا اگر پیچیدگی زمانی برنامه ایی **2n** باشد و پیچیدگی زمانی برنامهٔ دیگری n^2است کدام یک سریعتر است؟

البته هر دوبرنامه یک الگوریتم است.

در مورد پیچیدگی زمانی یک برنامه تعاریف ریاضی(نشانه گذاری مجانبی) وجود دارد که در اینجا به اختصار بحث می شود تا بتوانیم عبارات با معنی(ولی غیردقیق) در مورد پیچیدگی زمانی یک برنامه بدست آوریم.

تعریف Big Oh : **f(n) = O(g(n))** است اگر و فقط اگر مقادیر ثابتی مانند C و n_0 باشد که رابطه زیر برقرار باشد:

$$f(n) \le C\, g(n) \quad \text{برای تمام } n \ge n_0$$

مثال) فرض کنید $f(n)=3n+2$ باشد که دراین صورت اگر $n_0=2$ باشد و $C=4$ باشد و $g(n)=n$ رابطه زیر برقرار است

$$\mathbf{3n+2 \le 4\, n}$$

پس $3n+2=O(n)$

البته می توان g(n) های دیگری نیز پیدا کرد که این خاصیت را داشته باشند.

$$3n+2 \le n^2$$

$C=1$ و $n_0=4$

ولی معمولا تابعی را به عنوان g(n) در نظر می گیرند که از بقیه کوچکتر باشد.

$$n \le n^2$$

$f(n)=O(g(n))$ فقط نشان می دهد که g(n) حد بالایی مقدار f(n) است برای تمام $n \ge n_0$

<u>قضیه</u>: اگر $f(n)=a_m n^m+\ldots+a_1 n+a_0$ در این صورت $f(n)=O(n^m)$

تمرین: نشان دهید که $n! = O(n^n)$

تعریف امگا: $f(n)=\Omega(g(n))$ اگر و فقط اگر به ازای مقادیر ثابت C و n_0 رابطه $f(n) \ge C.g(n)$ برای $n \ge n_0$ همیشه برقرار باشد.

مثال) مثلا $3n+2 \ge 3.n$ است در ازای $n \ge 1$ $3n+2= \Omega(n)$

$10n^2+4n+7 \ge n^2$ در ازای $n \ge 1$ یعنی $10n^2+4n+7= \Omega(n^2)$ البته می توان نوشت که $10n^2+4n+7= \Omega(n)$

اما معمولا تابعی را در نظر می گیرند که از بقیه بزرگتر باشد.

عبارت $f(n)=\Omega(g(n))$ فقط نشان می دهد که g(n) یک حد پایین برای تابع f(n) است.

<u>قضیه</u>: اگر $f(n)=a_m n^m+\ldots+a_1 n+a_0$ در این صورت $f(n)= \Omega(n^m)$

تعریف تتا: $f(n)=\theta(g(n))$ است اگر و فقط اگر به ازای مقادیر c_1 و c_2 و n_0 رابطه زیر درست باشد

$$c_1.g(n) \le f(n) \le c_2.g(n) \quad \text{در ازای } n \ge n_0$$

<u>قضیه</u>: اگر $f(n)=a_m n^m+\ldots+a_1 n+a_0$ در این صورت $f(n)= \theta(n^m)$

نشانه گذاری تتا از امگا و Big oh دقیق تر است زیرا هم بعنوان کران بالا و هم بعنوان کران پایین f(n) می باشد.

سئوالی که مطرح می شود این است که این نشانه گذاری ها اگر یکی بخواهد دقیقا شمارش مراحل را انجام دهد، چه استفاده ایی دارد؟ پاسخ این است که پیچیدگی مجانبی(θ, Ω,O) بدون تعیین شمارش مراحل براحتی تعیین می گردد.

مثال) جمع دو ماتریس:

```
Void   ADD(int  *a)
{
 int  I , j;
For   (I=0 ;  I <Rows ; I++)      //➔   θ(Rows)
  For  (j=0 ; j<Cols ; j++)       // ➔ θ(Rows.Cols)
```

www.Arsanjan.blogfa.com

حال حداکثر پیچیدگی زمانی را به عنوان
پیچیدگی در نظر بگیریم یعنی
$\theta(Rows.Cols)$

بوسیله این پیچیدگی ها می توان دو برنامه را که عمل یکسانی را انجام می دهند را با هم مقایسه کرد.
مثلا برنامه ایی که $O(n^2)$ است و برنامه ایی که $O(2^n)$ را با هم مقایسه می کنیم. اگر $n=10$ باشد $n^2=100$ و $2^n=1024$
می شود.

فصل دوم: آرایه ها

از دیدگاه برنامه نویسان آرایه مجموعه ایی از خانه های پشت سر هم در حافظه است. ولی همیشه این طور تصور نمی شود. اگر آرایه را به عنوان یک ADT در نظر بگیریم مجموعه ای از زوج های <index , Value> است طوری که یک تناظر یک به یک بین اندیس و محتوای سلول دارد.
آرایه را می توان یک ADT با عمل های Stroe , Retrieve تعریف کرد به صورت زیر:

```
#include <stdio.h>
#define  Max  1024
class Array{
  float  Ar[Max];
  int end;
  public:
```

تمرین: کلاس آرایه روبرو را به صورت زیر کامل تر کنید:

الف) تابعی بنویسید که اندازه آرایه را برگرداند.

ب) تابعی برای چاپ محتوای آرایه بنویسید.

سربارگذاری عملگر:

عملگر = = برای بررسی تساوی دو داده ساده مانند int یا Float و غیره استفاده می شود. اما نمی توان این عملگر را برای نوع داده ساخته شده(ADT) استفاده کرد.

به عنوان مثال اگر کلاسی برای شی مستطیل تعریف کرده باشیم نمی توانیم دو شی از این کلاس را با عملگر = = مقایسه کرد.

می توانیم عملگر های را برای کلاس هایی که تعریف می کنیم سربارگذاری کنیم. یعنی مثلا عملگر تساوی نیز برای دو شی مستطیل بکار رود و درصورتی که دو مستطیل با هم برابر باشند یک برگرداند و درغیر این صورت صفر برگرداند.

این عمل را می توان برای عملگرهای دیگر مانند + یا << و غیره انجام داد.

```
#include <stdio.h>
#define  Max  1024
class Rectangle{
  int x1,x2,y1,y2;
  public:
    set(int a1,int b1,int a2,int b2)
     {
      x1=a1;  x2=a2;
      y1=b1;  y2=b2;
     }
   operator==(const Rectangle &s)
   {
     if (this==&s) return 1;
     if ((x1==s.x1) && (y1==s.y1) && (x2==s.x2) &&(y2==s.y2))
       return 1;
     else
      return 0;
   }
};
void main()
{
  Rectangle m,n;
  m.set(1,2,3,4);
  n.set(1,2,3,4);
```

```
 if (m==n)
   printf("Yes\n");
   else
     printf("No\n");
}
```

تمرین الف: برایADT آرایه یک عملگر = = برای تساوی دو آرایه تعریف کنید.

تمرین ب : برای ADT آرایه یک عملگر + جهت جمع محتوای دو آرایه با طول یکسان تعریف کنید.

نمایش چند جمله ایی ها بوسیله آرایه: $a_mX^m+a_{m-1}X^{m-1}+...+a_0$

سه روش مختلف را برای نمایش چند جمله ایی ها بحث می کنیم.

الف) یک روش به این صورت است که یک آرایه به اندازه MaxDegree تعریف کنیم و MaxDegree بزرگ ترین درجه چندجمله ایی است.

```
Class Polynomial{
   Int  degree;
   Float   coef[MaxDegree+1];
```


a.degree=n;

با این نحوه نمایش الگوریتمهای بسیار ساده برای بسیاری از اعمال چندجمله ای(جمع و تفریق و ...) بدست می آوریم. اما این روش از نظر حافظه بسیار ضعیف عمل می کند زیرا برای همه اشیا این کلاس یک آرایه به طول Maxdegree استفاده می شود. مثلا برای چند جمله ایی درجه ۲

ب)روش دوم استفاده از حافظه پویا برای ایجاد یک آرایه با طول مشخصی(درجه چند جمله ایی) است.

Cofe=new float[degree+1];

این روش نیز معایبی دارد مثلا برای نمایش یک چند جمله ایی خلوت(Sparce) فضای اضافی تخصیص داده می شود مانند$X^{1000}+1$

ج) روش سوم: در این روش یک آرایه به طول MaxDegree برای همه چند جمله ای ها تعریف می شود.

```
Private:
  Static   Ar[MaxTerm , 2];
   Static  int  free;
   Int    start,  finish;
```

مثلا برای نمایش دادن دو چند جمله ایی $A(x)=2X^{1000}+1$ و $B(x)=X^4+3X^2+4X+8$ به صورت زیر عمل می کنیم.

2	1	1	3	4	8				
1000	0	4	2	1	0				

نکته ایی که در این روش وجود دارد این است که یک آرایه Ar برای همه اشیا این کلاس استفاده می شوند و همگی یک متغیرFree دارند که انتهای آرایه را مشخص می کند و هرکدام از چند جمله ایی ها دارای یک Start و یک finish هستند.

عیب این روش این است که الگوریتمهای اعمال چند جمله ایی ها مانند جمع و ... ساده نیست. درضمن اگـر یـک شی از بین برود باید بقیه اشیا شیفت به چپ داشته باشند.

پروژه: روش سوم نمایش چند جمله ای ها را پیاده سازی کنید. در پیاده سازی باید بتـوان حاصـل جمـع دو چنـد جمله ای را محاسبه کرد. ص 84

ADT ماتریس اسپارس:

ماتریس اسپارس ماتریسی است که بیشتر عناصر آن صفر است و تعداد اندکی از عناصر آن غیر صفر است. به عنوان مثال یک ماتریس ۱۰۰۰×۱۰۰۰ را در نظر بگیرید که فقط ۱۰۰ عنصر غیر صفر داشته باشد.

معمولا برای نمایش ماتریس آرایه دو بعدی استفاده می شود اما برای ماتریس اسپارس استفاده از این روش کارآمد نیست زیرا فضای بسیار زیادی از حافظه هدر می شود.

در بیشتر مسائل اطراف ما ماتریس اسپارس دیده می شود به عنوان مثال یک تصویر سیاه و سفید که بیشتر تصویر سیاه باشد را می توان یک ماتریس اسپارس در نظر گرفت.(مانند تصاویر رادیولوژی)

یکی از روشهایی که برای ذخیره ماتریس اسپارس استفاده می شود استفاده از یک آرایه سه ستونی است که که هر سطر آن معرف یک <Row , Col , Value> است.

0	3	5	0	0	17	0	11	4	0
6	0	0	14	0	0	0	0	0	0
0	4	0	0	2	15	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	19	0	0	0	13	0
1	9	0	0	0	0	11	0	0	0
0	0	0	0	0	0	16	0	20	0
6	7	0	12	0	0	0	0	0	15
0	0	0	0	0	0	1	0	0	0
0	9	0	0	12	0	0	20	0	18

R	C	V
0	1	3
0	2	5
0	5	17
0	7	11
0	8	4
1	0	6
1	3	14
2	1	4
2	4	2
2	5	15
4	4	19
4	8	13
5	0	1
5	1	9
5	6	11
6	6	16

ماتریس

R	C	V
0	1	6
0	5	1
1	0	3
1	2	4
1	5	9
2	0	5
3	1	14
4	2	2
4	4	19
5	0	17
5	2	15
6	5	11
6	6	16
7	0	11
8	0	4
8	4	13

ترانهاده ماتریس

در نمایش سطری نمایش ماتریس اسپارس ماتریس
بر اساس سطرها مرتب است.

ترانهاده یک ماتریس اسپارس:

ترانهاده یک ماتریس یعنی جای سطرها و ستونهای آن را عوض کنیم.

برای نحوه نمایش ماتریس اسپارسی که بحث شد(سطری) باید وقتی جای سطرها و ستونها را تغییر دادید برحـسب سطر ها (که حال همان ستونها هستند)مرتب کنید.

اگر ماتریس اسپارس را به صورت آرایه دو بعدی ذخیره کنیم برای جابجا کردن سطرها و ستونها زمانی معادل O(Rows.Cols)نیاز است که بسیار زیاد است.

اما اگر ماتریس اسپارس به صورت سه گانه ها (سطری) ذخیره شود با یک الگوریتم مناسب می توان ترانهاده آن را محاسبه کرد که دارای پیچیدگی زمانی O(Terms.Columns) خواهد بود. Terms نشان دهنده تعداد عناصر غیر صفر در ماتریس اسپارس است.

```
For  (all  elements in Column  j)
   Place   element (I , j , Value) in position (j, I ,Value)
```

این حلقه نشان می دهد که تمام عضوهای ستون 0 را پیدا کرده و آنها را سطر 0 ذخیره می کند و سپس تمام عضوهای ستون 1 را پیدا کرده و آنها را در سطر 1 ذخیره می کند و ...

البته در صفحه ۹۳ کتاب روش بهتری برای اینکار نوشته شده است که دارای پیچیدگی زمانی O(cols+Rows) است.

ضرب ماتریس اسپارس:

حاصل ضرب دو ماتریس اسپارس لزوما اسپارس نیست.

1	0	0
1	0	0
1	0	0

1	1	1
0	0	0
0	0	0

1	1	1
1	1	1
1	1	1

حاصل ضرب دو ماتریس از ضرب سطرها در ستونها بدست می آید. اگر دوماتریس اسپارس A,B را به صورت سه گانه ها (سطری)ذخیره کنیم برای محاسبه A*B ابتدا ترانهاده ماتریس B را به روشی که بحث شد بدست آوریم حال می توانیم سطرها را در ستونها ضرب کنیم. دلیل این کار این است که اگر برای پیدا کردن ستون J در ماتریس B باید کل سه گانه ها پویش (جستجو) شود اما وقتی ترانهاده B بدست می آید عناصر ستون J کنار هم قرار می گیرند.

در صفحه ۹۶ کتاب برنامه ضرب دو ماتریس اسپارس نوشته شده است؟

تحقیق: پیچیدگی زمانی این برنامه را پیچیدگی زمانی برنامه ضرب معمولی دو ماتریس مقایسه کنید؟

تمرین: برنامه ایی بنویسید که دو ماتریس اسپارس هم درجه را دریافت کند و با هم جمع بزند و نمایش دهد؟

نمایش آرایه های چند بعدی:

آرایه های چند بعدی معمولا با ذخیره عناصر در یک آرایه یک بعدی پیاده سازی می شوند.

به عنوان مثال ماتریس را می توان یک آرایه یک بعدی در نظر گرفت.

0,0	0,1	0,2	0,3	1,0	1,1	1,2	1,3	2,0	2,1	2,2	2,3

آرایه بالا می تواند نشان دهنده یک ماتریس ۴×۳ باشد.

A[1][2]= A + 2 + 1*5
A[1][3] = A+ 3 + 1*5
A[2][4] = A+ 4 + 2*5
A[6][2] = A+ 2 + 6*5

A[X][Y] = A + Y +X*N
N ➔ Number of Columns

A[0][0]= A+0+0*5
A[0][1]= A+1+0*5
A[0][2]= A+2+0*5
A[0][4]= A+4+0*5
A[1][0]= A+0+1*5
A[1][1]= A+1+1*5

ADT رشته ها:

رشته در واقع آرایه ای از نوع char است و می توان برای ADT آن عملیاتهایی مانند ایجاد یک رشته تهی جدید، خواندن و نوشتن رشته، کپی کردن رشته، مقایسه رشته ها، درج یک رشته داخل رشته دیگر،پیداکردن یک الگو در یک رشته و غیره .

در زبان C رشته ها به صورت آرایه از کاراکترها که به کاراکتر 0\ ختم می شوند نمایش داده می شوند.

تمرین: برنامه ایی بنویسید که دو رشته را دریافت کند و با هم مقایسه کنددرصورت بزرگتر بودن رشته اولی 1 و در صورت کوچک تر بودن رشته اولی 1– درصورت مساوی بودن صفر برگرداند.

تطابق الگو:Pattern Matching

دو رشته را دریافت کرده ایم و می خواهیم رشته دومی را در اولی جستجو کنیم.

یکی از این روشها(راحترین و غیر موثرترین) این است که هر کاراکتر رشته را تا زمان پیدا شدن الگو و یا رسیدن به انتهای رشته تست متوالی کنیم. S1='aabbaababbabc' , S2='aba'

این روش دارای زمان محاسباتی O(n.m) است که n,m طول رشته S1 و S2 است.

اما روش دیگری برای پیدا کردن الگو وجود دارد که به الگوریتم کنوث،موریس و پرات معروف است.ص ۱۰۸

در واقع در این الگوریتم هر گاه انتهای رشتهٔ Pat با محل مقایسه String مطابق شد به اندازه طول رشته Pat مقایسه انجام می شود.

```
Int  nfind (char *string ,  Char *pat)
{
 int   I,j,start=0;
 int  lasts=strlen(string)-1;  // انتهای رشته اولی را مشخص می کند
 int  lastp=strlen(pat)-1 ;    // انتهای رشته دومی را مشخص میکند
 int   endmatch=lastp;
 for  (I=0 ;  endmatch<=lasts  ;  endmatch++ , start++) {
    if   (string[endmatch]=pat[lastp])
      for (j=0 , I=start ; j<lastp && string[I]=pat[j];  I++, j++)
         ;
if (j=lastp)
   return(start);
}
return   -1;
}
```

فصل سوم: پشته ها و صف ها

پشته (stack) نوعی ADT است که به نام LIFO معروف است. معروفترین پشته که کاربرد زیادی نیز دارد پشته سیستم است که برای فراخوانی توابع در زمان اجرا استفاده می شود.(در دروس برنامه سازی بحث شده است.)

Template در زبان C++:

قبلا وقتی یک کلاس در زبان C تعریف می شد همه اجزا ان دارای یک نوع خاصی بودند برای اینکه این نوع ها تغییر کنند لازم است که کد کلاس تغییر کند. به عنوان مثال وقتی یک آرایه int را برای نمایش چند جمله ای ها تعریف می کنید،ضرایب معادله فقط می توانند int باشند حال اگر بخواهید ضرایب اعشاری نیز داشته باشید باید کلاس مربوط به چند جمله ای ها را تغییر دهید و یا یک کلاس دیگر بنویسید.

اما با استفاده از خاصیت template می توانید این مشکل را حل کنید.

مثال:

```
Template  <class  Mytype>
Class  Array
{
  mytype  F;
  mytype  A[20];

public:
```

www.Arsanjan.blogfa.com

آرایه A برای شی M1 از نوع float و برای M2 از نوع
char و برای M3 از نوع int خواهد بود.

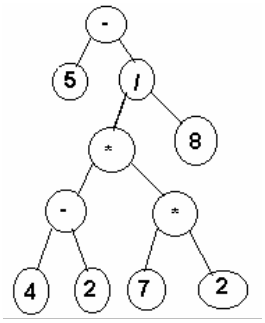
برای پیاده سازی یک ADT پشته می توان از یک آرایه ثابت استفاده کرد. و عملیاتهایی مانند empty، Full ، Pop , Push و Top را تعریف کرد.

یکی از موارد استفاده پشته ارزیابی یک عبارت است. برای این که بتوان عبارات را ارزیابی کرد ابتدا آنها را به روش PreFix تبدیل می کنیم. مثلا a+b را به صورت ab+ می نویسیم.

مثال) 2+3*4+6 را به صورت 2++346*2 می نویسیم برای اینکه بتوان چنین عبارتی را به Prefix تبدیل کنیم باید درخت ارزیابی آن را رسم کنیم.

مثال: 8 / (2-4) * 2 * 7- 5

- 5 / * - 4 2 * 7 2 8



بوسیله دو پشته می توان برنامه ایی نوشت که حاصل عبارت بالا را محاسبه کند.

پیاده سازی ADT برای یک **پشته**:

```
#include<process.h>
#include<iostream.h>
#define MAX 127
template<class eltype>class stack
{
 private:
  int ptr;
  eltype item[MAX];
 public:
  stack(void)
   {
    ptr=0;
   }
```

```
  int empty(void)
   {
    return(ptr==0);
   }
  int full(void)
   {
    return(ptr==MAX);
   }
  void push(const eltype element)
  {
   if(full())
    {
     cout<<"\n stack is full !!! run again program \n";
     exit(0);
    }
   else
   item[ptr++]=element;
  }
 eltype pop(void)
  {
   if(empty())
    return(0);
   else
     return(item[--ptr]);
  }
 eltype  top(void)
  {
   eltype element;
   element=pop();
   push(element);
   return(element);
  }
 };
```

برنامه ایی بنویسید که با استفاده از پشته طراحی شده حاصل عبارت Prefix را محاسبه کند.

```
#include"stack.hpp"
#include<ctype.h>
#include<math.h>
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
#define MAX 127
void main()
{
  float a,b,c;
  stack<float>    s1;
  stack<float>    s2;
  char s[MAX]; int i=0;
  gets(s);
  while (s[i]!='\0'){
   if ((s[i]>=48) && (s[i]<=57))
    s1.push(s[i]-48);
   else
     s1.push(s[i]);
     i++;
 } //*****************
```

```
while (!s1.empty())
  {
    while ( (!s1.empty() ) &&(
     ((s1.top()!='+') && (s1.top()!='*')
     && (s1.top()!='-') && (s1.top()!='%')
     && (s1.top()!='/')))))
    {
      s2.push(s1.pop());
    }
    c=s1.pop();
    if (c=='+')
      {
      a=s2.pop(); b=s2.pop();
      s1.push(a+b);
      }else
      if (c=='*')
      {
      a=s2.pop(); b=s2.pop();
      s1.push(a*b);
      }else
      if (c=='-')
      {
      a=s2.pop(); b=s2.pop();
      s1.push(a-b);
      }else
```

www.Arsanjan.blogfa.com

در این برنامه فرض شده است اعداد تک رقمی هستندو فقط % / * + عملگر هستند.

مثال: (2– 4) + 2 * 5 % (2– 9) + 8 ← **+ + 8 * % - 9 2 5 2 – 4 2**

اگر این عبارت به برنامه بالا داده شود حاصل 14 خواهد شد.

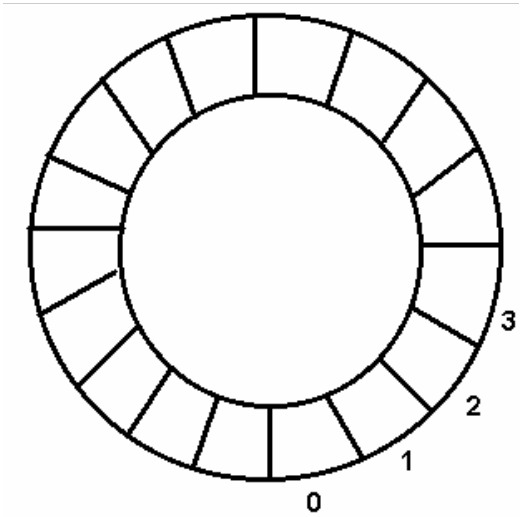
ADT صف : (QUEUE)

صف نوعی ADT است که به عنوان FIFO معروف است. اولین عضوی که وارد می شود اولین عنصری است که از لیست خارج می شود. برای پیاده سازی آن می توان از یک آرایه ثابت استفاده کرد.

در زیر یک کلاس جهت پیاده سازی Queue نوشته شده است. متغیر tail همیشه سرصف را نمایش می دهد.

```
#include<process.h>
#include<iostream.h>
#define MAXQEUE 200
template<class Mytype>class  Qeue{
    int tail;
    Mytype store[MAXQEUE];
  public:
//***********************************
   void Qeue()
     {
      tail=0;
     }
//************************************
   void Addqeue(Mytype elmnt)
    {
     if(Isfull())
      {
          cout<<"\n Qeue is full !!! \n";
          return;
      }
     else
      store[tail++]=elmnt;
    }
//***********************************
   Mytype Exitqeue()
     {
      int Cntqeue;
```

```
    Mytype Tmp;
    if(!Isempty())
      {
       Tmp=store[0];
       for(Cntqeue=0;Cntqeue<tail;++Cntqeue)
        store[Cntqeue]=store[Cntqeue+1];
       --tail;
        return(Tmp);
      }
      else
       cout<<"\n  Qeue is empty !!! \n";
   }
//*********************************
    int Isempty()
     {
       return(tail==0);
     }
//******************************
    int Isfull()
     {
       return(tail==MAXQEUE);
     }
}  // end class
```



یکی از معایب این روش این است که با خارج شدن یک عضو از صف بقیه داده ها شیفت پیدا می کنند و روش موثری نیست. برای حل این مشکل می توان یک آرایهٔ حلقوی را به عنوان صف طراحی کرد.

تمرین :کلاسی بنویسید که یک صف حلقوی را با عملیاتهای ذکر شده در کلاس بالا پیاده سازی کند.

مساله مسیر پرپیچ و خم (Mazing):

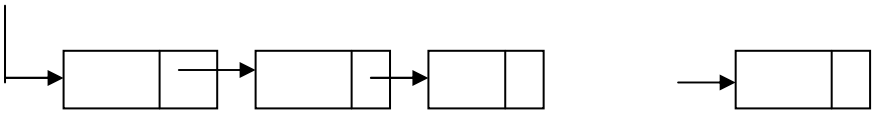
ارزشیابی عبارات:

اولین کاری که باید برای ارزیابی مقدار یک عبارت انجام دهیم انتخاب یک ترتیب برای اعمال است.

روش استانداردی که برای ارزیابی عبارات به کار می رود روش infix (میانوندی) است. زیرا در این روش علموند ها میان عملگر قرار می گیرند. روش دیگری که برای ارزیابی عبارات استفاده می شود روش Postfix (پسوندی) ا ست که در این روش ابتدا دو عملوند و سپس عملگر ظاهر می شود مثال: A+B*C ← ABC*+

نمادگذاری پسوندی خصوصیاتی دارد که باعث سادگی ارزیابی عبارت ها می شود۱- احتیاجی به پرانتز ندارد ۲- به اولویت عملگرها کاری ندارد.

لیست ها:



لیست تک پیوندی :

لیست های پیوندی گره های متوالی هستند که به صورت فلش هایی(اشاره گرها) به هم متصل هستند.

نکاتی که باید در مورد گره ها می توان استباط کرد این است که الف) گره ها واقعا پست سرهم در حافظه نیستند
ب) موقعیت گره ها در هر بار اجرا می تواند تغییر کند.
برای ایجاد لیست های پیوندی از حافظه پویا استفاده می شود.
malloc برای تخصیص حافظه و free برای از بین بردن حافظه تخصیص داده شده استفاده می شود.
در تکه برنامه روبرو یک ساختار برای هر گره تعریف شده است و سپس اشاره گر ابتدای لیست (Start) تعریف شده است.

```
struct  Node{
    int code;
    char name[20];
    float avg;
    struct Node *Next;
    };
struct Node *Start;
```

یک لیست پیوندی می تواند یک ADT تلقی شود که دارای عملیاتهایی مانند ADDToList و RemoveNode و SearchList و NumberNode و DeleteList و غیره باشد.

اضافه کردن به ابتدای لیست:

```
void AddToList()
{
    if (Start==NULL)
       {
        Start=(struct Node *)malloc(sizeof(Node));
        cout<<"Enter Code , Name , Avg";
        cin>>Start->code>>Start->name>>Start->avg;
        Start->Next=NULL;
       }
     else
      {
        struct Node *PTR;
        PTR=(struct Node *)malloc(sizeof(Node));
        cout<<"Enter Code , Name , Avg";
        cin>>PTR->code>>PTR->name>>PTR->avg;
        PTR->Next=Start;
        Start=PTR;
      }
}
```

برای اضافه کردن به انتهای لیست باید ابتدا از اول لیست تا انتهای لیست راپیمایش کرد و سپس گره جدید را به انتها متصل کرد این کاررا می توانید به عنوان تمرین انجام دهید.

حذف کردن یک گره بخصوص از لیست:

```
struct Node *  SearchList(int code)
{
  struct Node *LINK;
  LINK=Start;
  while ((LINK!=NULL) && (LINK->code!=code))
     LINK=LINK->Next;
  if (LINK==NULL)
    return(NULL);
  else
    return(LINK);
}//****************************
void RemoveNode(int code)
{
  struct Node *PTR,*LastPTR;
  PTR=SearchList(code);
  if  (PTR==NULL) return;
  if (PTR==Start)
     {//حالتی که ممکن است گره اول حذف شود
      Start=PTR->Next;
```

ابتدا گره ایی که قرار است حذف شود را Search می کنیم البته باید آدرس گره قبلی را نیز داشته باشیم تا بتوانیم پیوند را برقرار کنیم.
و باید حالت خاصی که گره اول حذف شود را نیز بررسی کنیم.

```
      free(PTR);
      return;
    }
  LastPTR=Start;
  while (LastPTR->Next!=PTR)
     LastPTR->Next;
  LastPTR->Next=PTR->Next;
  free(PTR);
 }
```

مثال) برنامه ای بازگشتی بنویسید که لیست تک پیوندی را معکوس کند؟

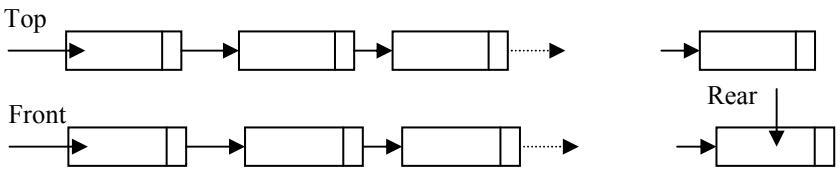
در ابتدا به این تابع پارامترهای Start,Start تحویل داده خواهد شد.

```
struct Node * Inverse(struct Node *Last,struct Node *First)
{
 struct Node *temp;
 if(First->Next==NULL)
    {
      First->Next=Last;
      Last->Next=NULL;
      return(First);
    }
    else
      {
          temp=Inverse(First,First->Next);
           if (First->Next ==NULL)
                {
                   First->Next=Last;
                   Last->Next=NULL;
                 }
          return(temp);
       } // end else
}
```

تکلیف: برنامه ایی بنویسید که ابتدای دولیست پیوندی را دریافت کند و با یکدیگر Merge کند.

پیاده سازی پشته و صف بوسیله لیست های تک پیوندی:

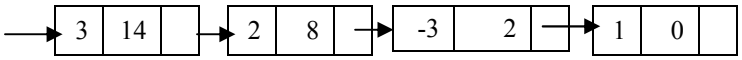
در فصل قبلی پشته و صف را با آرایه ها پیاده سازی کردیم که یکی از معایب این روش اتلاف فضا است. با توجه به اینکه تعداد گره ها در یک لیست پیوندی متغیر است میتوان به جای آرایه از یک لیست پیوندی برای نمایش صف و پشته استفاده کرد.



پروژه درسی: یک Stack با عملیاتهای قبلی ولی با حافظه پویا و لیست پیوندی را پیاده سازی کنید.

نمایش چند جمله ای ها بصورت لیست تک پیوندی:

برای نمایش چند جمله ای ها می توان از یک لیست پیوندی استفاده کرد و هر گره می تواند دارای فیلدهای ضریب(Coef) و توان(Expon) باشد. مثلا $a=3x^{14}+2x^{8}-3x^{2}+1$

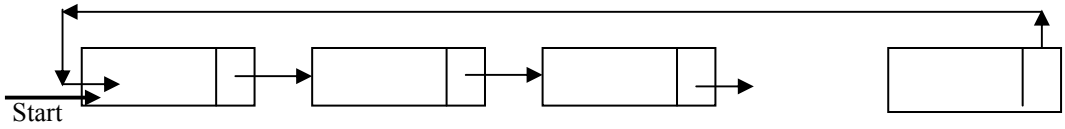


برای جمع کردن دو چند جمله ایی باید دو لیست را نشان دهنده چند جمله ایی ها هستند را با یکدیگرMerge کرد البته گره هایی که ضریب یکسان دارند با هم جمع می شوند.

معمولا در برنامه ها حذف کردن یک گره (Free) و اضافه کردن به گره ها(Malloc) بسیار کند انجام می شود به همین دلیل راهی به صورت زیر ارائه می شود: هنگامی که می خواهیم از یک لیست پیوندی گره ایی را حذف کنیم آن را Free نمی کنیم بلکه یک لیست به نام DeleteNode ایجاد کرده گره را از لیست اصلی برداشته و داخل این لیست قرار می دهیم. هنگامی که می خواهیم یک گره جدید اضافه کنیم در صورتی که لیست DeleteNode تهی نباشد گره ایی از ان را برداشته و دوباره به لیست اصلی اضافه می کنیم و در صورتی که DeleteNode تهی باشد آنگاه Malloc می کنیم.

لیستهای تک پیوندی حلقوی:

لیست پیوندی که انتهای آن به ابتدای متصل باشد را حلقوی می گویند.



عمل درج در لیست حلقوی و در ابتدای لیست کمی مشکل است زیرا گره آخری نیز باید به روز شود به همین دلیل معمولا عمل اضافه کردن گره بین گره اول و گره دوم انجام می شود.

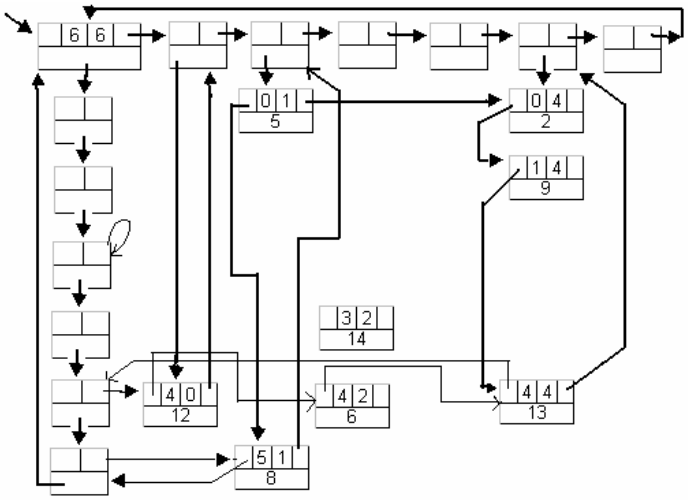
عمل حذف یک گره نیز ممکن است دردسر آفرین باشد و آن زمانی است که بخواهیم گره اول را حذف کنیم.

برای شمارش تعداد گره های یک لیست حلقوی چه راه حلی را پیشنهاد می دهید؟

نمایش ماتریس اسپارس بوسیله لیست های پیوندی:

برای نمایش یک ماتریس اسپارس می توان از لیست های پیوندی حلقوی استفاده کرد

0	5	0	0	2	0
0	0	0	0	9	0
0	0	0	0	0	0
0	0	14	0	0	0
12	0	6	0	13	0
0	8	0	0	0	0

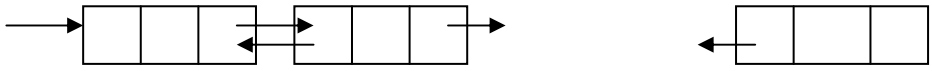


در شکل روبرو نحوهٔ اتصال گره های هر ستون و هر سطر نمایش داده شده است.

البته کامل نیست آنرا کامل کنید؟

هر سطر یک لیست پیوندی حلقوی است و هر ستون نیز یک لیست پیوندی حلقوی است.

لیستهای پیوندی دو گانه:



```
Struct  Node {
    int    Code;
    char    name[10];
    Struct   Node * Last;
    Struct    Node  * Next;
};
```

قبلا در مورد حذف یک گره از لیست پیوندی بحث کرده ایم که در آنجا باید آدرس گره قبلی را داشته باشیم تا بتوانیم آن گره را آزاد کنیم که پیدا کردن گره

قبلی مستلزم پیمایش لیست بود اما برای لیست های پیوندی دو گانه لازم به این کار نیست.

برنامه های نوشته شده در زیر جهت اضافه کردن یک گره به ابتدای لیست و حذف کردن یک گرهٔ بخصوص از لیست پیوندی دو گانه است:(با فرض اینکه گرهٔ Start یک گرهٔ سراسری است)

```
Void  Insert(void )
 {
   if (Start==NULL)
      {
        Start=(struct Node *) malloc(sizeof(Node));
         Cin>>  Start -> Code >> Start ->Name;
        Start->Next=Start->Last=NULL;
      }
    else
      {
         struct Node * Ptr;
        Ptr=(struct Node *) malloc(sizeof(Node));
         Cin >> Ptr->Code >> Ptr->Name;
         Ptr->Next=Start;
         Start->Last=Ptr;
         Ptr->Last=NULL;
         Start=Ptr;
       }
}
```

```
Void  delete(struct Node * Ptr)
{
    if  (Ptr==Start)
      {//گره ایی که حذف می شود گره اول باشد
         Ptr->Next->Last=NULL;
        Start=Ptr->Next;
        Free(Ptr);
      }
     else
    {
       Ptr->Last->Next=Ptr->Next;
       Ptr->Next-Last=Ptr->Last;
       Free(Ptr);
    }
}
```

تمرین) برنامه ایی بنویسید که یک لیست پیوندی دو گانه را معکوس کند؟

لیست های ناهمگن:

لیست هایی که تا حال بررسی کرده ایم گره های آنها از نوع Struct بودند و همه گره ها از نظر نوع با هم یکسان بودند.حال اگر لیست پیوندی ایجاد کنیم که گره های آن با هم متفاوت باشند لیست پیوندی ناهمگن بوجود می آید برای این کار می توان بجای استفاده از Struct از union ها استفاده کرد.

تهیه و تنظیم به وسیله : مرتضی خادمیان

برای دانلود کتاب های بیشتر به آدرس www.Arsanjan.blogfa.com مراجعه کنید.