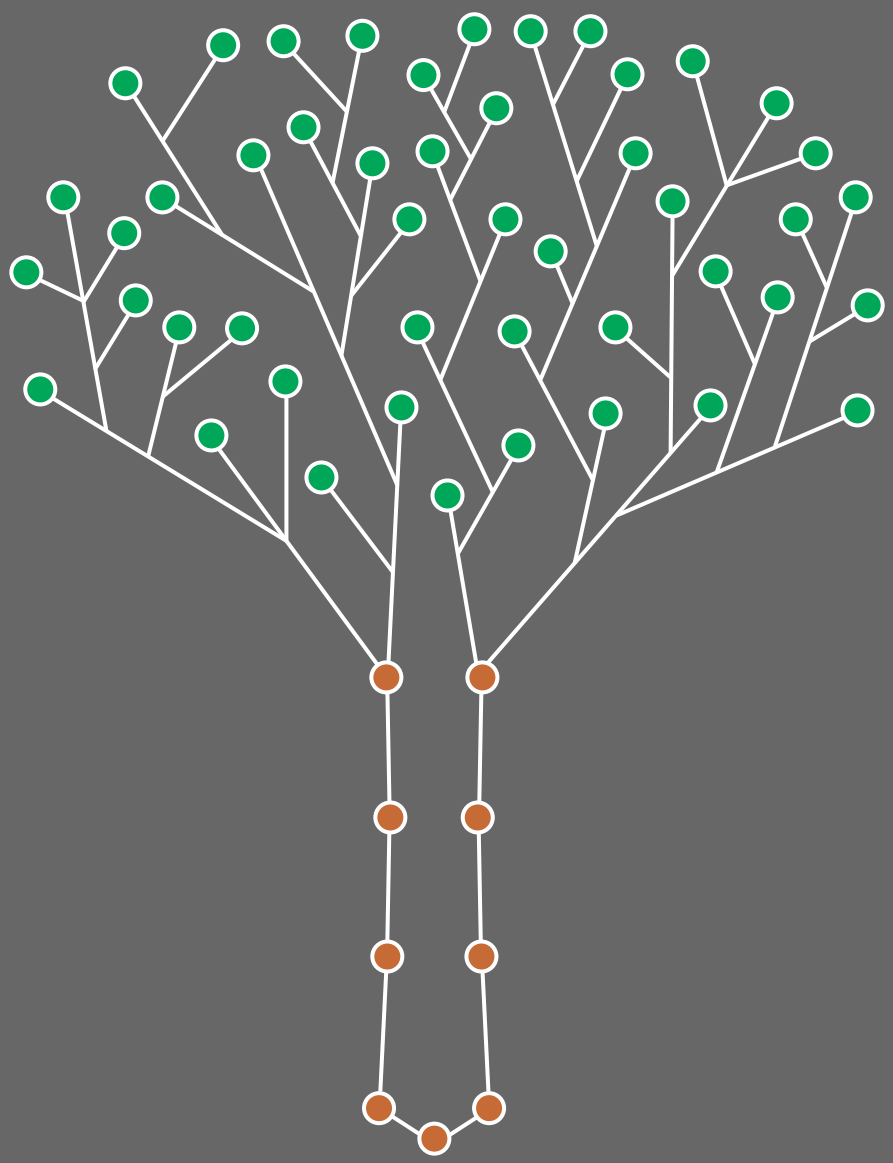
# پرسش و پاسخ داده ساختارها



تالیف

مسعود فلاح پور

به نام یزدان نیک اندیش

# پرسش و پاسخ داده ساختارها

تالیف
مسعود فلاح پور

آبان ۱۳۹۳
نگارش ۰/۲

# مجوز

تقدیم به

پدر و مادرم؛ که همواره یار و یاورم بوده‌اند

# فهرست مطالب

# فهرست الگوریتم‌ها

# پیش‌گفتار

در علوم کامپیوتر درس‌هایی وجود دارند که از آنها به عنوان درس‌هایی بنیادین یاد می‌شود. برخی از این درس‌ها عبارت‌اند از: داده‌ساختارها، تجزیه و تحلیل الگوریتم‌ها، طراحی کامپایلر و نظریه‌ی زبان‌ها و ماشین‌ها.

از میان این درس‌های بنیادین یکی از مهمترین آنها، درس داده‌ساختارها است که به عنوان پیشنیازی برای درس‌هایی همچون تجزیه و تحلیل الگوریتم‌ها و سیستم‌های عامل نیز مطرح است. اگر داده‌ساختار را به این صورت تعریف کنیم که «یک داده‌ساختار روشی برای ذخیره و سازماندهی داده‌ها است به طوریکه بازیابی و/یا تغییر داده‌ها به سادگی و با کارایی بالا انجام شود» آنگاه می‌توان به تعریفی از درس داده‌ساختارها نیز رسید. در درس داده‌ساختارها به بررسی دقیق و موشکافانه‌ی انواع داده‌ساختارها و چگونگی پیاده‌سازی آنها در یک زبان برنامه‌نویسی پرداخته می‌شود.

به دلیل اهمیت درس داده‌ساختارها کتاب‌های مختلفی در مورد آن نوشته شده است که بسیاری از آنها دارای قالب کم و بیش یکسانی هستند. قالب کلی این کتاب‌ها به این شکل است که در هر فصل از کتاب ابتدا به معرفی و بررسی یک داده‌ساختار خاص پرداخته شده و در انتهای فصل تمریناتی مرتبط با آن داده‌ساختار ارائه می‌شود. در کتاب حاضر سعی شده است از قالبی متفاوت استفاده شود.

در این کتاب فرض بر این است که خواننده با مباحث مختلف داده‌ساختارها آشنایی نسبی دارد و در نتیجه هر فصل از این کتاب دارای بخش نخست کتابهای معمول، یعنی معرفی و بررسی یک داده‌ساختار، نیست. تمرکز این کتاب بر روی مطرح کردن تعدادی سوال در مورد هر یک از انواع داده‌ساختارها و دادن پاسخ گام به گام و تشریحی به هر یک از سوالات است. به بیانی دیگر می‌توان قالب این کتاب را به صورت پرسش و پاسخ در نظر گرفت که به خواننده کمک می‌کند تا فهم عمیقتری از داده‌ساختارهای مختلف به دست آورد.

در نگارش حاضر، تنها فصل‌های اول، دوم و پنجم در کتاب گنجانده شده‌اند و سایر فصول پس از آماده‌سازی و کسب اطمینان از کیفیت علمی و ظاهری آنها در نگارش‌های بعدی به کتاب اضافه خواهند شد.

متن کتاب با استفاده از سیستم حروفچینی لاتک ، بسته‌ی زی‌پرشین و ویرایشگر bidiTexmaker آماده شده است. برای متن پارسی از قلم XB Niloofar و برای کلمات انگلیسی و شبه‌کدها از قلم Computer Modern استفاده شده است. برای طراحی جلد کتاب از نرم‌افزار Corel DRAW و برای رسم شکل‌ها از بسته‌ی PSTricks و نرم‌افزارهای Corel DRAW و LaTeXDraw استفاده شده است. برای دسترسی به متن خام کتاب می‌توانید به نشانی `https://github.com/MasoodFallahpoor/DS-Book` مراجعه کنید.

در آماده‌سازی این کتاب تلاش شده است تا چه از نظر علمی و چه از نظر ظاهری کتابی شایسته و خالی از خطا به خوانندگان تقدیم شود. اما از آنجایی که هیچ کتابی نمی‌تواند به طور کامل از خطا در امان باشد از این رو از شما خواننده‌ی گرامی خواهشمندم در صورت مشاهده هرگونه خطای املایی، نگارشی و یا علمی به نشانی `masood.fallahpoor@gmail.com` اطلاع دهید تا خطای موجود در ویرایش‌های بعدی کتاب رفع شود.

مسعود فلاح‌پور
آبان ۱۳۹۳

# درباره مولف

**مسعود فلاح‌پور** در سال ۱۳۶۷ در تهران متولد شد و تحصیلات اولیه خود را در همین شهر پشت سر گذاشت. او در سال ۱۳۸۷ مدرک کاردانی و در سال ۱۳۹۰ مدرک کارشناسی ناپیوسته خود را از دانشکده فنی شماره دو تهران (شهید شمسی‌پور) دریافت کرد. مسعود دارای مدرک کارشناسی ارشد مهندسی کامپیوتر در گرایش مهندسی نرم‌افزار از دانشکده مهندسی برق و کامپیوتر دانشگاه شهید بهشتی است.

مسعود به هر دو جنبه‌ی نظری و عملی علم کامپیوتر علاقه‌مند است. از جمله علاقه‌مندی‌های او در بخش نظری می‌توان به سیستم‌های عامل، داده‌ساختارها، طراحی الگوریتم‌ها و طراحی کامپایلر اشاره کرد. علاقه به سیستم عامل گنو/لینوکس، برنامه‌نویسی به زبان‌های جاوا و سی و همچنین برنامه‌نویسی اندروید از جمله علاقه‌مندی‌های او در بخش عملی است.

# قدردانی

قدردانی و نام بردن از تمام افرادی که در به ثمر رسیدن این کتاب نقش داشته‌اند کاری است بس دشوار. به همین جهت فقط از برخی افراد نام برده خواهد شد.

بر خود لازم می‌دانم از آقای مهدی جوانمرد که ایده اولیه نوشتن این کتاب را مطرح کردند و همچنین جمع‌آوری بخشی از سوالات هر فصل را بر عهده داشتند صمیمانه سپاسگزاری کنم.

همچنین قدردانی می‌کنم از استاد گرامی، دکتر محسن ابراهیمی مقدم، که فهم دقیق و عمیق بسیاری از مفاهیم داده‌ساختارها را مدیون ایشان هستم.

در نهایت نیز از تلاش‌های چندین و چند ساله‌ی آقای وفا کارن پَهلَو برای توسعه‌ی بسته‌ی زی‌پرشین کمال قدردانی را دارم زیرا با خلق این بسته کمک شایانی به جامعه‌ی دانشگاهی ایران کرده‌اند.

# قواعد شبه‌کد

برای بیان الگوریتم‌های بیان شده در کتاب، به جای استفاده از یک زبان برنامه‌نویسی خاص، از شبه‌کد استفاده شده است. با استفاده از شبه‌کد می‌توان الگوریتم‌ها را به شکلی ساده بیان کرد و از بیان جزئیات غیر ضروری خودداری کرد. در ادامه توضیحاتی در مورد کلیات شبه‌کد استفاده شده در کتاب بیان خواهد شد.

## توضیحات

اگر در قسمتی از شبه کد نیاز به توضیح وجود داشته باشد، مانند زبان C++ از دو علامت اسلش پشت سرهم برای شروع توضیح استفاده می‌شود. در ادامه نمونه‌ای از یک توضیح آورده شده است.

```
// This is a comment
```

## زیربرنامه‌ها

تمامی الگوریتم‌های کتاب به صورت زیربرنامه تعریف می‌شوند. یک زیربرنامه دارای دو نوع است: تابع و رویه. اگر زیربرنامه بخواهد مقداری را به عنوان خروجی بازگرداند آنگاه از تابع استفاده می‌کنیم و اگر مقداری را برنگرداند از رویه استفاده خواهیم کرد.

تعریف یک تابع با کلمه کلیدی function آغاز می‌شود. سپس نام تابع بیان می‌شود و در صورتی که تابع دارای ورودی باشد، ورودی‌های تابع در داخل پرانتز آورده می‌شوند. در ادامه‌ی تعریف تابع، بدنه تابع شروع می‌شود و در انتها مقداری به عنوان خروجی تابع توسط دستور return برگشت داده می‌شود. عبارت end function نیز خاتمه تعریف تابع را نشان می‌دهد. شکل کلی تعریف یک تابع در ادامه نشان داده شده است.

```
function FunctionName(param1, param2, ... , paramN)
    // body of function
    return result
end function
```

شکل کلی تعریف یک رویه هم مانند یک تابع است با این تفاوت‌ها که تعریف یک رویه با کلمه کلیدی procedure آغاز می‌شود، مقداری توسط رویه بازگردانده نمی‌شود و همچنین خاتمه رویه توسط عبارت end procedure مشخص می‌شود.

## متغیرها

برای تعریف متغیرها از قالب $VarName : VarType$ استفاده می‌شود. برای مثال اگر بخواهیم متغیر $i$ را از نوع عدد صحیح تعریف کنیم به صورت $i : integer$ عمل می‌کنیم.

متغیرهای مورد استفاده در شبه‌کدها در اکثر مواقع به صورت صریح تعریف نمی‌شوند و فرض بر این است که با اولین استفاده از یک متغیر، آن متغیر به صورت ضمنی تعریف نیز می‌شود. در حالت تعریف ضمنی متغیرها، با توجه به شبه‌کدی که متغیر در آن مورد استفاده قرار گرفته است، به راحتی می‌توان به نوع آن نیز پی برد.

## آرایه‌ها

اندیس تمامی آرایه‌ها از عدد یک آغاز می‌شود مگر اینکه در یک شبه‌کد صراحتاً چیز دیگری بیان شود. برای دسترسی به خانه‌ی $i$ ام آرایه یک بعدی $A$ از قالب $A[i]$ و برای دسترسی به عنصر سطر $i$ ام و ستون $j$ ام آرایه دو بعدی $B$ از قالب $B[i, j]$ استفاده می‌شود.

اگر متغیر $A$ نشان دهنده یک آرایه یک بعدی باشد آنگاه طول این آرایه در خصیصه $length$ آن قرار دارد و برای دسترسی به آن از قالب $A.length$ استفاده می‌شود. اگر $A$ یک آرایه دو بعدی باشد تعدادی سطرهای آن در خصیصه $row$ و تعداد ستون‌های آن در خصیصه $column$ قرار دارد و برای دسترسی به آنها به ترتیب از قالب $A.rows$ و $A.columns$ استفاده می‌شود.

جهت اشاره به بازه‌ای از یک آرایه از قالب $A[i\,..\,j]$ استفاده می‌شود که در آن $i$ اندیس شروع بازه و $j$ اندیس پایان بازه است.

## حلقه‌ها

برای تکرار یک تا تعدادی دستور از دو نوع حلقه استفاده خواهد شد: حلقه for و حلقه while. از ساختار حلقه for زمانی استفاده می‌شود که تعداد تکرار بدنه حلقه از قبل مشخص باشد و از حلقه while زمانی استفاده می‌شود که تعداد تکرار بدنه حلقه از قبل معلوم نباشد.

تعریف حلقه for با کلمه کلیدی for آغاز می‌شود. سپس مقدار اولیه شمارنده حلقه به متغیر شمارنده حلقه انتساب داده می‌شود و پس از کلمه کلیدی to مقدار نهایی شمارنده حلقه مشخص می‌شود. بعد از تعریف سرآیند حلقه، بدنه حلقه تعریف می‌دهد و در نهایت عبارت end for پایان حلقه را نشان می‌شود. در ادامه شکل کلی تعریف حلقه for نشان داده شده است.

```
for counter = startValue to endValue
    // body of for loop
end for
```

با هر بار اجرای این حلقه یک واحد به متغیر شمارنده حلقه افزوده می‌شود و بدنه حلقه تا زمانی اجرا می‌شود که شرط $counter \leqslant endValue$ برقرار باشد. اگر بخواهیم شمارنده حلقه به جای افزایش، کاهش یابد آنگاه به جای کلمه کلیدی to از کلمه کلیدی downto استفاده می‌شود.

تعریف حلقه while با کلمه کلیدی while آغاز می‌شود. سپس یک عبارت منطقی قرار می‌گیرد و تا زمانی که عبارت منطقی برقرار باشد بدنه حلقه اجرا می‌شود. خاتمه حلقه while نیز با عبارت end while نشان

داده می‌شود.

```
while booleanExpression
    // body of while loop
end while
```

## دستورات شرطی

برای شروع یک دستور شرطی از کلمه کلیدی if استفاده می‌شود و در ادامه یک عبارت منطقی آورده می‌شود. اگر عبارت منطقی درست باشد دستورات بخش اول و در غیر این صورت دستورات بخش دوم اجرا می‌شوند. خاتمه تعریف دستور شرطی نیز با عبارت end if نشان داده می‌شود. برای تعریف یک دستور شرطی از قالب کلی زیر استفاده می‌شود.

```
if booleanExpression
    // statements to be executed when booleanExpression is TRUE
else
    // statements to be executed when booleanExpression is FALSE
end if
```

وجود بخش else اجباری نیست و این یعنی اگر این بخش وجود نداشته باشد و شرط دستور شرطی برقرار نباشد آنگاه بدنه دستور شرطی اجرا نخواهد شد.

## دستور return

با اجرای دستور return اجرای زیربرنامه بلافاصله پایان می‌یابد. از این دستور در صورت نیاز برای بازگرداندن مقدار یا مقادیری به عنوان خروجی یک تابع نیز می‌توان استفاده کرد. در ادامه شکل‌های مختلف دستور return آورده شده است.

```
return
return result
return (result1, result2, ... , resultN)
```

در صورتی که شکل خروجی تابعی مانند $\textsc{Func}(A, B)$ مانند حالت سوم دستور return باشد آنگاه از شکل زیر برای دریافت تمامی خروجی‌های آن استفاده خواهد شد. با اجرای دستور زیر مقدار result1 در var1 قرار می‌گیرد، result2 در var2 قرار می‌گیرد و به همین ترتیب تا resultN که در varN قرار می‌گیرد.

```
var1, var2, ... , varN=Func(A, B)
```

## عملگرها

عملگرهای منطقی مورد استفاده در الگوریتم‌ها عبارت‌اند از $>$، $<$، $\leq$، $\geq$، $\neq$ و $==$ که از عملگر آخر برای بررسی تساوی دو مقدار استفاده می‌شود.

برای ترکیب عبارات منطقی از عملگرهای and، or و not استفاده می‌شود. جهت انتساب مقداری به یک متغیر از عملگر $=$ استفاده می‌شود.

برای انجام چهار عمل اصلی ریاضی از عملگرهای $+$، $-$، $\times$ و $/$ استفاده می‌شود. همچنین از عملگر mod به عنوان عملگر باقیمانده استفاده می‌شود.

## اشاره‌گرها

برای تخصیص فضا به یک اشاره‌گر از زیربرنامه NEW استفاده می‌شود. برای مثال اگر $p$ یک اشاره‌گر باشد و بخواهیم به آن فضا اختصاص دهیم باید زیربرنامه NEW را به صورت $\text{NEW}(p)$ فراخوانی کنیم. همچنین برای آزادسازی فضای اختصاص یافته به یک اشاره‌گر از زیربرنامه FREE استفاده می‌شود.

اگر فرض کنیم $p$ یک اشاره‌گر باشد که به یک ساختار[1] اشاره دارد و این ساختار دارای فیلدی به نام $next$ است آنگاه از قالب $p.next$ برای دسترسی به محتوای فیلد $next$ استفاده می‌شود.

برای بیان مقدار تهی در زمان کار با اشاره‌گرها از ثابت NULL استفاده می‌شود.

[1] منظور از ساختار، چیزی مانند struct در زبان C یا record در زبان پاسکال است.

# فصل ۳

# لیست‌های پیوندی

## ۱.۳ مقدمه

## ۲.۳ منابع مطالعاتی

## ۳.۳ نوع داده‌ی انتزاعی لیست

◂ **سوال ۱.** بدون در نظر گرفتن جزئیات مربوط به پیاده‌سازی داده‌ساختار لیست و به کمک توابع اولیه‌ی FIRST، NEXT، DELETE و RETRIEVE، الگوریتمی بازگشتی بنویسید که با دریافت لیستی همچون $L = \langle a_1, a_2, a_3, \cdots, a_n \rangle$ مقادیر آن را با ترتیب $a_1, a_n, a_2, a_{n-1}, \ldots$ چاپ کند و پس از چاپ مقدار هر عنصر آن عنصر را از لیست حذف کند. عملکرد توابع اولیه به شرح زیر است:

- $\text{FIRST}(L)$: اشاره‌گر به عنصر اول از لیست $L$ را برمی‌گرداند.
- $\text{NEXT}(L, p)$: اشاره‌گر به عنصری از لیست $L$ که بعد از عنصری است که $p$ به آن اشاره دارد را برمی‌گرداند.
- $\text{DELETE}(L, p)$: عنصری که اشاره‌گر $p$ به آن اشاره دارد را از لیست $L$ حذف می‌کند.
- $\text{RETRIEVE}(L, p)$: محتوای داده‌ای عنصری از لیست $L$ که اشاره‌گر $p$ به آن اشاره دارد را برمی‌گرداند.

◃ **پاسخ سوال ۱.**

ابتدا یک تابع کمکی به نام LAST می‌نویسیم که به کمک آن می‌توان اشاره‌گر به آخرین عنصر یک لیست را به دست آورد. شبه کد این تابع در الگوریتم (۱.۳) نشان داده شده است. از تابع LAST در زیربرنامه‌ی چاپ مقادیر لیست استفاده خواهد شد.

**الگوریتم ۱.۳** به دست آوردن اشاره‌گر به آخرین عنصر یک لیست

```
function LAST(L)
    p = FIRST(L)
    while NEXT(L, p) ≠ NULL
        p = NEXT(L, p)
    end while
    return p
end function
```

شبه کد زیربرنامه‌ی چاپ مقادیر لیست پیوندی با ترتیب خواسته شده در صورت سوال، در الگوریتم (۲.۳) آورده شده است. روند کار این زیربرنامه بسیار ساده است. اگر لیست ورودی خالی باشد آنگاه زیربرنامه کار خاصی انجام نداده و اجرای آن خاتمه می‌یابد. در صورتی که لیست ورودی دارای تنها یک عنصر باشد آنگاه مقدار این عنصر به دست آمده و چاپ می‌شود. اگر لیست ورودی دارای بیش از یک عنصر باشد در این صورت ابتدا عنصر اول چاپ می‌شود، سپس عنصر آخر و در ادامه عناصر اول و آخر لیست حذف شده و زیربرنامه به صورت بازگشتی خود را فراخوانی می‌کند تا سایر مقادیر لیست نیز چاپ شوند.

◄ **سوال ۲.** الگوریتم (۳.۳) قرار است تمام عناصری از یک لیست که دارای مقدار $x$ هستند را از لیست حذف کند. توضیح دهید چرا این الگوریتم در مواقعی به درستی کار نمی‌کند. برای رفع مشکل باید چه تغییر یا تغییراتی در الگوریتم ایجاد کرد؟

◁ **پاسخ سوال ۲.**

◄ **سوال ۳.** لیست $L$ دارای طول $n$ است. اگر قطعه کد زیر را بر روی لیست $L$ اعمال کنیم آنگاه هر یک از توابع FIRST، NEXT و END چند بار اجرا می‌شوند؟

```
p = FIRST(L)
while p ≠ END(L)
    q = p
    while q ≠ END(L)
        q = NEXT(q, L)
        r = FIRST(L)
        while r ≠ q
            r = NEXT(r, L)
        end while
    end while
    p = NEXT(p, L)
end while
```

◁ **پاسخ سوال ۳.**

**الگوریتم ۲.۳** چاپ مقادیر یک لیست با ترتیبی خاص

```
procedure PrintList(L)
    if L == NULL
        return
    end if
    if First(L) == Last(L)
        f = First(L)
        x = Retrive(L, f)
        Write(x)
    else
        f = First(L)
        l = Last(L)
        xf = Retrieve(L, f)
        xl = Retrieve(L, l)
        Write(xf)
        Write(xl)
        Delete(L, f)
        Delete(L, l)
        PrintList(L)
    end if
end procedure
```

**الگوریتم ۳.۳** حذف عناصر با مقدار $x$ از یک لیست

```
procedure DeleteValues(L, x)
    p = First(L)
    while p ≠ End(L)
        if Retrieve(p, L) == x
            Delete(p, L)
        end if
        p = Next(p, L)
    end while
end procedure
```

## ۴.۳ لیست‌های پیوندی یکطرفه

◂ **سوال ۴.** زیربرنامه‌ای با نام MakeNull را در نظر بگیرید که از آن برای حذف تمامی عناصر یک لیست پیوندی یکطرفه استفاده می‌شود. این زیربرنامه را می‌توان به صورت بازگشتی یا غیربازگشتی نوشت. نسخه‌ی غیربازگشتی این زیربرنامه در الگوریتم (۴.۳) و نسخه‌ی بازگشتی در الگوریتم (۵.۳) نشان داده شده است.

**الگوریتم ۴.۳** حذف تمامی عناصر یک لیست پیوندی یکطرفه به صورت غیربازگشتی

```
procedure MakeNull(L)
    while L ≠ NULL
        p = L
        L = L.next
        Free(p)
    end while
end procedure
```

**الگوریتم ۵.۳** حذف تمامی عناصر یک لیست پیوندی یکطرفه به صورت بازگشتی

```
procedure MakeNull(L)
    if L == NULL
        return
    else
        MakeNull(L.next)
        Free(L)
    end if
end procedure
```

با در نظر گرفتن پیاده‌سازی بازگشتی و غیربازگشتی زیربرنامه MakeNull، چه تفاوتی در ترتیب آزاد شدن فضای مصرفی عناصر لیست $L$ در این دو زیربرنامه وجود دارد؟ نسخه‌ی بازگشتی زیربرنامه را به گونه‌ای تغییر دهید که از نظر آزادسازی فضای مصرفی لیست $L$ دقیقاً مانند نسخه‌ی غیربازگشتی عمل کند.

◃ **پاسخ سوال ۴.**

در نسخه‌ی غیر بازگشتی عناصر لیست از ابتدای لیست شروع به حذف شدن می‌کنند در حالی که در نسخه بازگشتی عناصر از انتها شروع به حذف شدن می‌کنند. به بیانی دیگر ترتیب حذف عناصر در دو پیاده‌سازی برعکس یکدیگر است.

می‌توان نسخه‌ی بازگشتی را به صورتی بازنویسی کرد که عناصر از ابتدای لیست شروع به حذف شدن کنند. شبه کد تغییر یافته‌ی نسخه بازگشتی زیربرنامه MakeNull در الگوریتم (۶.۳) آمده است.

◂ **سوال ۵.** برای ایجاد یک کپی از یک لیست پیوندی یکطرفه به صورت بازگشتی، می‌توان از الگوریتم (۷.۳) استفاده کرد. یک تابع غیربازگشتی برای کپی کردن یک لیست پیوندی یکطرفه بنویسید. در نسخه‌ی

**الگوریتم ۶.۳** حذف تمامی عناصر یک لیست پیوندی یک‌طرفه به صورت بازگشتی

```
procedure MakeNull(L)
    if L == NULL
        return
    end if
    p = L
    L = L.next
    Free(p)
    MakeNull(L)
end procedure
```

غیربازگشتی ممکن است تفاوت‌هایی در ایجاد سلول‌ها، اتصال آن‌ها به یکدیگر و ... نسبت به نسخه‌ی بازگشتی وجود داشته باشد. این تفاوت‌ها را ذکر کنید.

**الگوریتم ۷.۳** کپی یک لیست پیوندی یکطرفه به صورت بازگشتی

```
function CopyList(L)
    if L == NULL
        return NULL
    end if
    New(p)
    p.data = L.data
    p.next = Copy(L.next)
    return p
end function
```

◁ **پاسخ سوال ۵.**

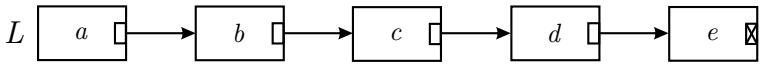
شبه کد نسخه‌ی غیربازگشتی تابع CopyList در الگوریتم (۸.۳) نشان داده شده است.

تفاوتی که نسخه‌ی غیربازگشتی تابع CopyList با نسخه‌ی بازگشتی خود دارد این است که ترتیب برقراری ارتباط سلول‌ها در نسخه‌ی بازگشتی از انتها به ابتدای لیست است در حالیکه در نسخه‌ی غیربازگشتی برقراری ارتباطات از ابتدا به انتهای لیست است. به بیان بهتر ترتیب انتساب مقادیر اشاره‌گر $next$ در دو نسخه‌ی بازگشتی و غیربازگشتی متفاوت است.

◀ **سوال ۶.** یک لیست پیوندی یکطرفه که دارای $n$ عنصر است را در نظر بگیرید. فرض کنید عناصر این لیست را از ابتدا به انتها و به ترتیب از ۱ تا $n$ شماره‌گذاری کرده‌ایم. تابعی به نام SplitList بنویسید که یک لیست پیوندی یکطرفه را دریافت و عناصر با شماره‌ی زوج را از آن حذف کرده و در لیست جدیدی درج کند. برای مثال اگر لیست ورودی به صورت شکل (۱.۳) باشد آنگاه پس از اجرای تابع باید به لیست‌های نشان داده شده در شکل (۲.۳) برسیم. (دقت داشته باشید که حذف عناصر از لیست ورودی و درج آنها در

**الگوریتم ۸.۳** کپی یک لیست پیوندی یکطرفه به صورت غیربازگشتی

```
function CopyList(L)
    if L == NULL
        return NULL
    end if
    p = L
    New(t)
    q = t
    while p ≠ NULL
        q.data = p.data
        if p.next ≠ NULL
            New(q.next)
            q = q.next
        end if
        p = p.next
    end while
    return t
end function
```



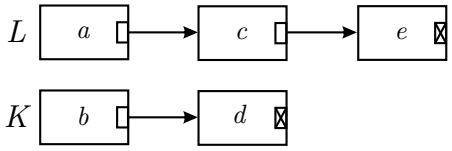

**شکل (۱.۳):** یک لیست پیوندی نمونه به عنوان ورودی تابع SplitList

لیست جدید باید تنها با تغییر اشاره‌گرهای عناصر انجام شود و نباید هیچ گره‌ی جدیدی ایجاد شود).

◁ **پاسخ سوال ۶.**

شبه کد الگوریتم جداسازی عناصر با شماره‌ی زوج از لیست ورودی در الگوریتم (۹.۳) آمده است.

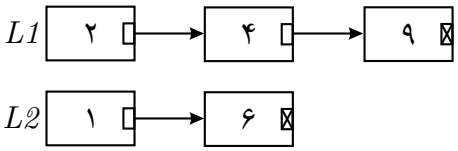
اگر لیست $L$ تهی باشد آنگاه تنها کافیست لیست $K$ را نیز برابر با تهی قرار دهیم. همچنین هنگامی که لیست $L$ دارای تنها یک عنصر است باید لیست $K$ را برابر با تهی قرار دهیم زیرا در حالتی که فقط یک عنصر وجود دارد این عنصر دارای شماره یک است لیست $L$ باید شامل آن عنصر بوده و لیست $K$ تهی باشد.




**شکل (۲.۳):** لیست‌های حاصل از اجرای تابع SplitList

**الگوریتم ۹.۳** جداسازی عناصر با شماره‌ی زوج یک لیست پیوندی یکطرفه

```
1: function SplitList(L)
2:     if L == NULL or L.next == NULL
3:         K = NULL
4:     else
5:         K = L.next
6:         L.next = L.next.next
7:         K.next = SplitList(L.next)
8:     end if
9:     return K
10: end function
```




**شکل (۳.۳):** دو لیست پیوندی نمونه به عنوان ورودی تابع Merge

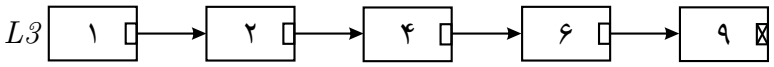
اگر لیست $L$ دارای بیش از یک عنصر باشد آنگاه ابتدا لیست $K$ به عنصر دوم لیست $L$ اشاره کرده و سپس عنصر بعد از عنصر اول در لیست $L$ به عنصر سوم لیست $L$ اشاره می‌کند. به این ترتیب $L$ شامل عناصر با شماره‌های ۱, ۳, ۴, ..., $n$ و $K$ شامل عنصر با شماره‌ی دو خواهد بود. سپس زیربرنامه‌ی SplitList به صورت بازگشتی فراخوانی می‌شود تا به همین ترتیب عناصر با شماره‌ی فرد در لیست $L$ و عناصر با شماره‌ی زوج در لیست $K$ قرار بگیرند.

◄ **سوال ۷.** دو لیست پیوندی یکطرفه‌ی L1 و L2 را در نظر بگیرید. مقادیر موجود در این دو لیست به صورت صعودی مرتب هستند. تابعی غیربازگشتی به نام Merge بنویسید که تمام عناصر دو لیست L1 و L2 را در لیست جدید L3 درج کند به شکلی که عناصر لیست L3 نیز به صورت صعودی مرتب باشند. برای مثال اگر دو لیست نشان داده شده در شکل (۳.۳) به عنوان ورودی تابع Merge داده شوند آنگاه پس از پایان اجرای زیربرنامه باید به لیست جدیدی مانند آنچه در شکل (۴.۳) نمایش داده شده است برسیم.

◃ **پاسخ سوال ۷.**

شبه کد تابع ادغام عناصر دو لیست پیوندی یکطرفه در الگوریتم (۱۰.۳) آورده شده است. بیان نحوه‌ی عملکرد این تابع در ادامه آمده است.

حلقه‌ی while موجود در خطوط ۶ تا ۱۶ در هر دور از اجرای خود از میان دو عنصر جاری لیست‌های L1 و




**شکل (۴.۳):** لیست پیوندی حاصل از اجرای تابع Merge بر روی لیست‌های پیوندی شکل (۳.۳)

**الگوریتم ۱۰.۳** ادغام عناصر دو لیست پیوندی یکطرفه

```
function MERGE(L1, L2)
    NEW(L3)
    p = L3
    q = L1
    r = L2
    while q ≠ NULL and r ≠ NULL
        NEW(p.next)
        if q.data < r.data
            p.data = q.data
            q = q.next
        else
            p.data = r.data
            r = r.next
        end if
        p = p.next
    end while
    if q == NULL
        while r ≠ NULL
            p.data = r.data
            if r.next ≠ NULL
                NEW(p.next)
                p = p.next
            end if
            r = r.next
        end while
    end if
    if r == NULL
        while q ≠ NULL
            p.data = q.data
            if q.next ≠ NULL
                NEW(p.next)
                p = p.next
            end if
            q = q.next
        end while
    end if
    return L3
end function
```

L2، عنصری را که دارای مقدار کمتری است انتخاب کرده و در لیست L3 درج می‌کند. این حلقه زمانی پایان می‌یابد که حداقل یکی از دو لیست L1 یا L2 به انتهای خود برسند. پس از پایان اجرای حلقه باید بررسی شود که کدام لیست به انتهای خود رسیده است تا بتوان عناصر باقیمانده در لیست دیگر را در لیست L3 درج کرد. اگر لیست L1 به انتهای خود رسیده باشد باید عناصر باقیمانده در لیست L2 را به لیست L3 اضافه کرد (خطوط ۱۷ تا ۲۶) و اگر لیست L2 به انتهای خود رسیده باشد آنگاه عناصر باقیمانده در لیست L1 باید در لیست L3 درج شوند (خطوط ۲۷ تا ۳۶).

◄ **سوال ۸.** زیربرنامه‌ای غیربازگشتی بنویسید که عناصر تکراری یک لیست پیوندی یکطرفه را حذف کند. برای مثال اگر ورودی زیربرنامه لیست $L = \langle a, b, a, c, b, d, d \rangle$ باشد آنگاه خروجی زیربرنامه باید لیست $L = \langle a, b, c, d \rangle$ باشد. نسخه بازگشتی چنین زیربرنامه‌ای را نیز طراحی کنید.

◃ **پاسخ سوال ۸.**

شبه کد نسخه‌ی غیربازگشتی زیربرنامه‌ی حذف عناصر تکراری یک لیست پیوندی یکطرفه در قالب الگوریتم (۱۱.۳) نمایش داده شده است.

**الگوریتم ۱۱.۳** حذف عناصر با مقادیر تکراری از یک لیست پیوندی یکطرفه

```
1:  procedure RemoveRedundant(L)
2:      if L == NULL
3:          return
4:      end if
5:      p = L
6:      while p.next ≠ NULL
7:          q = p
8:          while q.next ≠ NULL
9:              if q.next.data == p.data
10:                 r = q.next
11:                 q.next = r.next
12:                 Free(r)
13:             else
14:                 q = q.next
15:             end if
16:         end while
17:         p = p.next
18:     end while
19: end procedure
```

روش عملکرد زیربرنامه‌ی RemoveRedundant به این صورت است که در هر بار اجرای حلقه‌ی while بیرونی، یک عنصر از لیست $L$ که اشاره‌گر $p$ به آن اشاره دارد به عنوان عنصر جاری در نظر گرفته می‌شود و

عمل پیمایش در لیست $L$، از عنصر جاری به بعد، آغاز می‌شود. در حین انجام پیمایش اگر عنصری یافت شود که مقدار آن برابر با مقدار عنصر جاری باشد آنگاه این عنصر از لیست حذف می‌شود. اجرای حلقه‌ی while بیرونی زمانی متوقف می‌شود که به ازای تمام عناصر لیست $L$ عمل پیمایش انجام شده و تمام عناصر با مقادیر تکراری حذف شده باشند.

نسخه‌ی بازگشتی زیربرنامه‌ی حذف عناصر تکراری یک لیست پیوندی یکطرفه در الگوریتم (۱۲.۳) آمده است. نسخه‌ی بازگشتی به این صورت عمل می‌کند که فرض می‌شود عنصر ابتدایی لیست وجود ندارد و زیربرنامه به صورت بازگشتی با شروع از عنصر دوم لیست ورودی فراخوانی می‌شود. پس از بازگشت از فراخوانی خط ۵، دارای لیستی خواهیم بود که ممکن است تعدادی از عناصر آن دارای مقدار یکسان با مقدار عنصر اول لیست باشند. به این ترتیب با شروع از ابتدای لیست و با انجام عمل پیمایش، هر عنصری که دارای مقدار یکسان با مقدار عنصر اول بود از لیست حذف می‌شود.

**الگوریتم ۱۲.۳** حذف عناصر با مقادیر تکراری از یک لیست پیوندی یکطرفه به صورت بازگشتی

```
1:  procedure REMOVEREDUNDANT(L)
2:      if L == NULL or L.next == NULL
3:          return
4:      end if
5:      REMOVEREDUNDANT(L.next)
6:      p = L
7:      while p.next ≠ NULL
8:          if p.next.data == L.data
9:              r = p.next
10:             p.next = r.next
11:             FREE(r)
12:         else
13:             p = p.next
14:         end if
15:     end while
16: end procedure
```

◄ **سوال ۹.** فرض کنید دارای یک لیست پیوندی یکطرفه به نام $L$ و اشاره‌گر $p$ هستیم. اشاره‌گر $p$ به یکی از عناصر لیست $L$ که عنصر آخر لیست نیست اشاره دارد. به جز اشاره‌گر $p$ هیچ اشاره‌گر دیگری به هیچ یک از عناصر لیست $L$ وجود ندارد مگر اینکه از ابتدای لیست پیمایش انجام شود. قطعه کدی از مرتبه‌ی $O(1)$ بنویسید که مقدار موجود در عنصری که اشاره‌گر $p$ به آن اشاره دارد را از لیست $L$ حذف کند.

◃ **پاسخ سوال ۹.**

تکه کد لازم برای حذف مقدار مورد نظر در ادامه آمده است.

```
1: p.data = p.next.data
```

2: $t = p.next$
3: $p.next = p.next.next$
4: $\text{FREE}(t)$

◂ **سوال ۱۰.** فرض کنید از لیست پیوندی یکطرفه برای نگهداری اعداد صحیح مثبت استفاده شده است. روش ذخیره‌سازی یک عدد در چنین لیستی به این صورت است که رقم یکان در گره‌ی ابتدای لیست، رقم دهگان در گره‌ی دوم، رقم صدگان در گره‌ی سوم و به همین ترتیب تا رقم آخر که در گره‌ی آخر لیست قرار می‌گیرد. تابعی بنویسید که دو عدد صحیح، در قالب مطرح شده، را دریافت کرده و یک لیست پیوندی که حاوی حاصل جمع دو عدد ورودی است را بازگرداند.

◃ **پاسخ سوال ۱۰.**

## ۵.۳ لیست‌های پیوندی دوطرفه

◂ **سوال ۱۱.** لیست پیوندی دوطرفه $L$ را در نظر بگیرید. گره‌ای مانند $n$ در $L$ دارای دو اشاره‌گر است که یکی به گره‌ی قبلی و دیگری به گره‌ی بعدی اشاره دارد (اشاره‌گرهای $prev$ و $next$). با ترفندی خاص می‌توان لیست پیوندی دوطرفه $L'$ را تنها با یک اشاره‌گر به ازای هر گره پیاده‌سازی کرد که در این صورت فضای مصرفی به ازای هر گره در یک لیست پیوندی دوطرفه برابر با گره‌ای در یک لیست پیوندی یکطرفه خواهد بود. این ترفند به این صورت است که هر گره مانند $n'$ در $L'$ دارای اشاره‌گری به نام $np$ است. فرض کنید مقدار هر اشاره‌گر را می‌توان به صورت عددی $k$ بیتی تفسیر کرد. به ازای هر گره مانند $n$ در $L$ گره‌ای مانند $n'$ در $L'$ در نظر می‌گیریم و قرار می‌دهیم $n'.np = n.prev \text{ XOR } n.next$ و همچنین $n'.data = n.data$.

با توجه به توضیحات ارائه شده الگوریتمی بنویسید که لیستی از نوع $L'$، مقدار $x$ و اشاره‌گر $p$ را دریافت کرده و گره‌ای با مقدار $x$ را بعد از گره‌ای که $p$ به آن اشاره دارد درج کند (مقدار NULL را معادل با عدد صفر در نظر بگیرید).

◃ **پاسخ سوال ۱۱.**

## ۶.۳ لیست‌های اندیسی

◂ **سوال ۱۲.** پیاده‌سازی لیست توسط آرایه را در نظر بگیرید. زیربرنامه‌ای بنویسید که با در اختیار داشتن دو لیست L1 و L2، عنصر با موقعیت $p$ را از لیست L1 حذف کرده و در لیست L2 بعد از موقعیت $q$ درج کند. سرآیند زیربرنامه نوشته شده توسط شما باید به صورت زیر باشد:

$\text{MOVE}(L1, p, L2, q)$

◃ **پاسخ سوال ۱۲.**

شبه کد حذف عنصر با موقعیت $p$ از لیست L1 و درج آن بعد از موقعیت $q$ در لیست L2 در قالب الگوریتم (۱۳.۳) نمایش داده شده است. در این الگوریتم فرض شده است اگر مقدار $q$ برابر با NULL باشد آنگاه عنصر

حذف شده از لیست L1 باید در ابتدای لیست L2 درج شود.

**الگوریتم ۱۳.۳** حذف عنصری از لیستی اندیسی و درج آن در لیست اندیسی دیگر

```
1: procedure MOVE(L1, p, L2, q)
2:     if p ≠ L1
3:         temp = L1
4:         while space[temp].next ≠ p
5:             temp = space[temp].next
6:         end while
7:         if q ≠ NULL
8:             space[p].next = space[q].next
9:             space[q].next = p
10:        else
11:            space[p].next = L2
12:            L2 = p
13:        end if
14:    else
15:        if q ≠ NULL
16:            temp = space[L1].next
17:            space[L1].next = L2
18:            L2 = L1
19:            L1 = temp
20:        else
21:            temp = space[L1].next
22:            space[L1].next = space[q].next
23:            space[q].next = L1
24:            L1 = temp
25:        end if
26:    end if
27: end procedure
```

در ادامه به توضیح روند کلی کار الگوریتم پرداخته می‌شود. عملکرد این الگوریتم، با توجه به اینکه اشاره‌گر $p$ به کدام عنصر از لیست L1 اشاره می‌کند، به دو حالت تقسیم می‌شود. حالت اول در خطوط ۲ تا ۱۳ و حالت دوم در خطوط ۱۵ تا ۲۶ آمده است.

در حالت اول اشاره‌گر $p$ به عنصری غیر از عنصر اول لیست L1 اشاره دارد. در این حالت باید در لیست L1 عمل پیمایش را انجام داد تا به عنصرِ قبل از عنصری برسیم که $p$ به آن اشاره دارد (خطوط ۴ تا ۶). سپس با توجه به اینکه اشاره‌گر $q$ به ابتدای لیست L2 اشاره دارد یا خیر عملیات لازم برای درج عنصر در لیست L2

انجام می‌شود (خطوط ۷ تا ۱۳).

در حالت دوم اشاره‌گر $p$ به عنصر ابتدایی لیست L1 اشاره دارد. در این حالت نیازی به پیمایش لیست L1 وجود ندارد. تنها کافیست با توجه به اینکه اشاره‌گر $q$ به ابتدای لیست L2 اشاره دارد یا خیر عملیات لازم برای حذف عنصر ابتدایی لیست L1 و درج آن در لیست L2 انجام شود (خطوط ۱۵ تا ۲۶).

◂ **سوال ۱۳.** با فرض پیاده‌سازی لیست یک‌طرفه به کمک آرایه و در اختیار داشتن زیربرنامه‌های DISPOSE و ALLOCATE، زیربرنامه‌های زیر را پیاده‌سازی کنید:

- INSERT$(L, p, d)$: عنصری با مقدار $d$ ساخته و آن را در لیست $L$، بعد از عنصری که $p$ به آن اشاره دارد، درج می‌کند.
- DELETE$(L, p)$: عنصری که $p$ به آن اشاره دارد را از لیست $L$ حذف می‌کند.
- NEXT$(L, p)$: اشاره‌گر به عنصری از لیست $L$ که بعد از عنصری است که $p$ به آن اشاره دارد را برمی‌گرداند.
- PREV$(L, p)$: اشاره‌گر به عنصری از لیست $L$ که قبل از عنصری است که $p$ به آن اشاره دارد را برمی‌گرداند.

◃ **پاسخ سوال ۱۳.**

◂ **سوال ۱۴.** دانشگاهی با ۴۰۰۰ دانشجو و ۲۵۰۰ درس را در نظر بگیرید. این دانشگاه برای سیستم آموزشی خود قصد تولید دو نوع گزارش را دارد:

۱. گزارشی شامل افراد ثبت نام کننده برای هر درس به تفکیک درس.

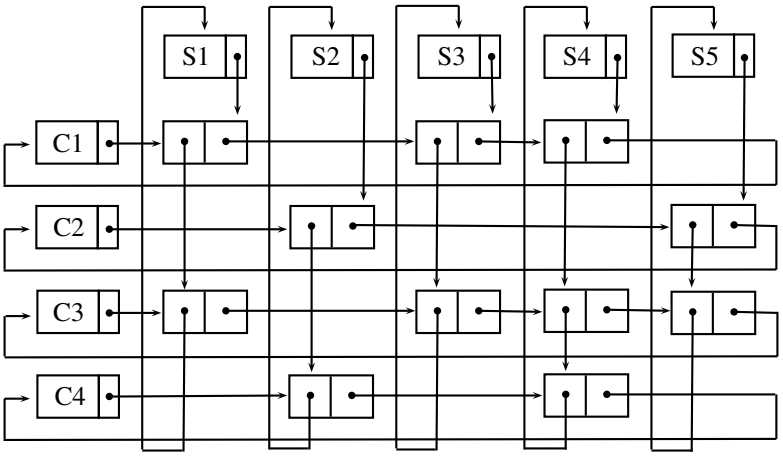
۲. گزارشی شامل دروس اخذ شده توسط هر دانشجو به تفکیک دانشجو.

یک راه برای نگهداری اطلاعات مربوط به دانشجویان و کلاس‌ها استفاده از یک آرایه دو بعدی است. چنین آرایه‌ای دارای صد میلیون خانه خواهد بود! در این دانشگاه هر دانشجو به طور متوسط در سه درس ثبت نام می‌کند و این یعنی به طور متوسط تنها ۱۲۰ هزار خانه از این آرایه دارای مقادیر مفید هستند (حدود ۰/۱ درصد از کل فضای آرایه). در چنین حالتی فضای بسیار زیادی به هدر می‌رود. به کمک لیست‌های پیوندی داده‌ساختاری طراحی کنید که فقط شامل اطلاعات مفید باشد و همچنین مناسب تولید دو نوع گزارش ذکر شده هم باشد.

◃ **پاسخ سوال ۱۴.**

شکل (۵.۳) نمایش دهنده‌ی داده‌ساختار مورد نظر است. توجه داشته باشید که این داده‌ساختار تنها داده‌ساختار ممکن نیست. همانطور که از شکل مشخص است، چندین لیست پیوندی با یکدیگر ترکیب و در قالب یک لیست پیوندی پیچیده به نمایش درآمده‌اند. تمام لیست‌ها دارای گره‌ی سرلیست هستند و به صورت حلقوی پیاده‌سازی شده‌اند. روش به دست آوردن گزارش‌های ذکر شده در ادامه بیان خواهد شد.

برای استخراج اسامی افراد ثبت‌نام کننده در یک کلاس، مثلاً کلاس C3 ، کافیست از گره با سرلیست C3 شروع کرده و به سمت راست این لیست حرکت کنیم. بعد از رفتن به اولین گره از لیست C3 ، مقدار جاری اشاره‌گر را ذخیره می‌کنیم و سپس با استفاده از اشاره‌گر به سمتِ پایین این گره به صورت عمودی در لیست




**شکل (۵.۳)**

حرکت کرده تا به سرلیست لیست عمودی برسیم (در اینجا لیست S1). به این ترتیب نام اولین دانشجو ثبت نام کننده در کلاس C3 به دست می‌آید. در گام بعد و برای به دست آوردن نام دانشجوی بعدی از اشاره‌گر ذخیره شده در گام قبل استفاده کرده و دوباره به سمت راست در لیست C3 حرکت کرده و روند حرکت عمودی در ساختمان داده را ادامه می‌دهیم. به این ترتیب می‌توان اسامی تمام افراد ثبت نام کننده در یک کلاس را به دست آورد.

برای به دست آوردن دروس ثبت نام شده توسط یک دانشجو به روشی مشابه با پاراگراف قبل عمل می‌کنیم با این تفاوت که در اینجا از گره با سر لیست حاوی نام دانشجو شروع کرده و به سمت پایین لیست حرکت می‌کنیم و برای یافتن هر درس به صورت افقی لیست‌ها را پیمایش می‌کنیم.

# کتاب‌نامه


[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3rd ed. , 2009.

[۲] محسن ابراهیمی مقدم، آذرخش کی‌پور و حسین عبدی. *ساختمان‌داده‌ها و الگوریتم‌ها*. انتشارات نصیر، ویرایش اول، ۱۳۹۲.

[3] E. Horowitz, S. Sahni, and S. Anderson-Freed. *Fundamentals of Data Structures in C*. Silicon Press, 2nd ed. , 2007.

[4] M. A. Weiss. *Data Structures and Algorithm Analysis in Java*. Pearson Education, 3rd ed. , 2012.