Linked List

برنامه سازی پیشرفته

استاد: دکترثبوتی

مسئول تمرین: محمدباقر مهدی زاده

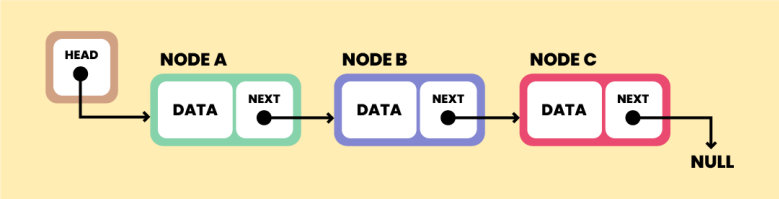
لیست پیوندی یکی از بنیادی‌ترین و در عین حال جذاب‌ترین ساختمان‌داده‌ها در علوم کامپیوتر است. برخلاف آرایه‌ها که عناصر آن‌ها به‌صورت پشت سر هم در حافظه ذخیره می‌شوند، در لیست پیوندی هر عنصر (یا همان «گره») به‌صورت مستقل در حافظه قرار می‌گیرد و تنها از طریق اشاره‌گرها (Pointers) به عنصر بعدی متصل می‌شود.

این ویژگی باعث می‌شود که:

- افزودن یا حذف عناصر از وسط لیست بسیار سریع‌تر از آرایه‌ها انجام شود، زیرا نیازی به جابه‌جایی سایر عناصر نیست.
- اندازه‌ی لیست بتواند به‌صورت پویا در زمان اجرا تغییر کند، بدون آن‌که از قبل اندازه‌ی آن مشخص باشد.
- درک نحوه‌ی کار اشاره‌گرها و تخصیص پویا (Dynamic Allocation) در حافظه، از طریق این ساختمان‌داده به‌صورت عمیق و کاربردی حاصل شود.

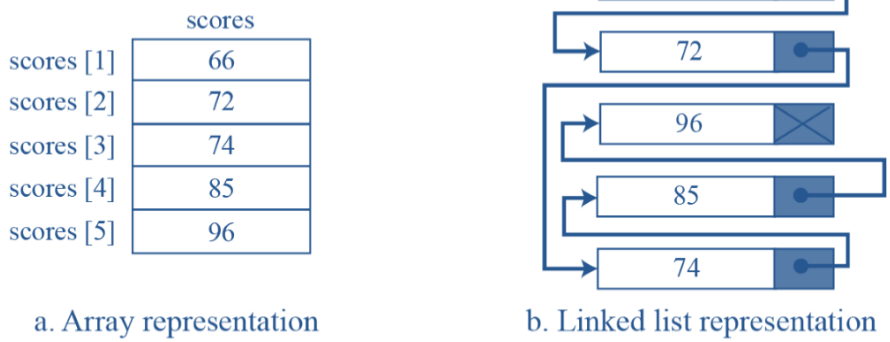
لیست‌های پیوندی در بسیاری از ساختارهای پیشرفته‌تر مانند **پشته (Stack)**، **صف (Queue)**، **درخت‌ها (Trees)** و حتی **سیستم‌عامل‌ها** و **کامپایلرها** نقش اساسی دارند. درک درست آن‌ها به معنی تسلط بر بخش مهمی از منطق داده‌ساختارها و حافظه است.

این ساختار داده، ساختاری از داده‌هاست که در آن عناصر به‌صورت زنجیروار به یکدیگر متصل‌اند. هر عنصر در این لیست شامل دو بخش اصلی است: بخشی برای نگهداری داده و بخشی برای برقراری ارتباط با گره های دیگر. برخلاف آرایه‌ها که عناصر در مکان‌های متوالی از حافظه قرار دارند، در لیست پیوندی موقعیت هر عنصر در حافظه ممکن است مستقل از بقیه باشد.



ساده ترین نوع لیست پیوندی

ساختار کلی گره



a. Array representation

b. Linked list representation

در این تمرین شما به کامل کردن پیاده سازی نوع خاصی از آن به نام لیست پیوندی دوطرفه یا همان doubly linked list. فرم کلی این ساختار به شکل زیر است:

```
20 struct List
21 {
22     size_t length;
23     node_ptr start;
24 };
```

گره یا Node:

برای دیدن یک نمونه از پیاده سازی گره می توانید به ساختار Node در کد مربوط مراجعه کنید. در این پیاده سازی یک گره به دو گره لینک می شود:

```
 8 struct Node //actual definition of Node;
 9 {
10     int value;
11     node_ptr left, right;
12 };
```

گره سمت راست خود

گره سمت چپ خود

اولین و آخرین گره:

پیاده سازی های گوناگون از این داده ساختار موجود است که هر یک ممکن است با اولین و آخرین گره به طور متفاوتی برخورد کند، در پیاده سازی مدنظر تمرین گره اول گره ای است که اشاره گر به سمت چپ در آن NULL/nullptr باشد. همچنین اگر اشاره گر به سمت راست NULL/nullptr باشد آن گره،آخرین گره از لیست می باشد.

واضح است که اگر لیست فقط شامل یک گره باشد، آن گره هم گره اول است و هم آخر. با توجه به این نکته واضح است که در ساخت گره ای جدید باید به این نکته توجه کرد.

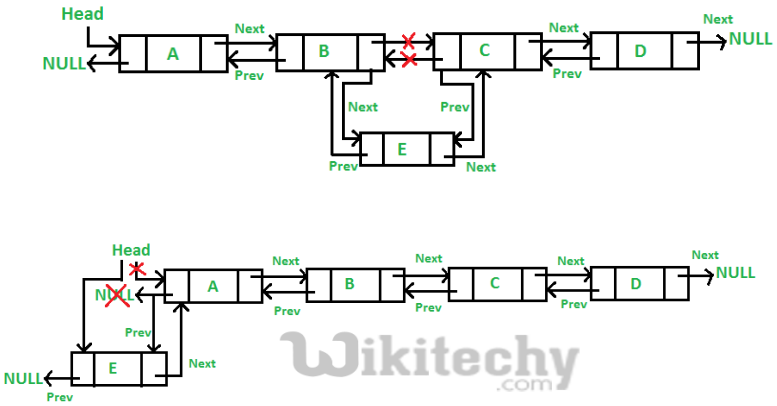
```
29 node_ptr make_node(int value = 0, node_ptr left = nullptr, node_ptr right = nullptr)
30 {
31     Node *tmp = new Node();
32     tmp->right = right;
33     tmp->left = left;
34     tmp->value = value;
35     return tmp;
36 }
37
```

پیمایش لیست:

یکی از نکات کلیدی در کار با لیست پیوندی، توانایی پیمایش آن است؛ یعنی بتوان از ابتدا تا انتهای لیست حرکت کرد و به مقادیر هر عنصر دسترسی یافت. همچنین باید امکان افزودن عناصر جدید در محل مناسب و حذف عناصر غیرضروری نیز وجود داشته باشد.

درج کردن:

زمانی که بخواهیم داده‌ای را در لیست اضافه یا حذف کنیم، کافی است ارتباط بین عناصر تغییر کند؛ نیازی به جابه‌جایی سایر داده‌ها وجود ندارد. این ویژگی باعث می‌شود لیست پیوندی انتخابی مناسب برای کاربردهایی باشد که در آن‌ها اندازه‌ی داده‌ها یا ترتیب آن‌ها به‌صورت پویا تغییر می‌کند.



درج کردن یک گره در doubly linked list

تصویر اول از geeks for geeks

مراحل درج گره:

. ساخت یک گره

. پیدا کردن مکان مناست درج

. درج گره، برقراری لینک های جدید

نکات مهم در درج:

. تقریبا در تمامی انواع لیست پیوندی درج کردن به سر لیست نیاز به دقت بیشتری دارد.(تصویر بالا)

. به برقراری یا تغییر ارتباط میان:

گره جدید و گره بعدی

گره جدید و گره قبل از آن

گره قبل و گره بعد از گره جدید

توجه کنید.(تصویر بالا)

.به اضافه کردن مقدار length یا طول لیست در این عملیات دقت کنید.

حذف گره:

در لیست پیوندی دوطرفه، هر گره علاوه بر مقدار داده، دارای دو اشاره‌گر است: یکی به گره قبلی و دیگری به گره بعدی. این ساختار امکان پیمایش لیست در هر دو جهت را فراهم می‌سازد و همچنین عملیات حذف را ساده‌تر از لیست پیوندی یک‌طرفه می‌کند.

هنگامی که قصد حذف یک گره از لیست را داریم، کافی است **ارتباط بین گره‌های مجاور** آن را به‌گونه‌ای تنظیم کنیم که گره مورد نظر از زنجیره خارج شود. به بیان دقیق‌تر:

- اشاره‌گر «بعدی» (right) گره قبلی باید به گره بعدیِ گره حذف‌شونده اشاره کند.
- اشاره‌گر «قبلی» (left) گره‌ی بعدی نیز باید به گره‌ی قبلیِ گره حذف‌شونده اشاره کند.

با این دو تغییر، گره حذف‌شده دیگر در مسیر پیمایش لیست قرار نخواهد گرفت و می‌توان حافظه‌ی مربوط به آن را آزاد کرد.

در صورتی که گره‌ی مورد نظر در ابتدا یا انتهای لیست قرار داشته باشد، لازم است اشاره‌گر آغاز یا پایان لیست نیز به‌روز شود تا از باقی‌ماندن اشاره‌گرهای نادرست جلوگیری شود.

در این تمرین، هدف از مشاهده و تحلیل این بخش از کد، درک نحوه‌ی مدیریت ارتباط میان گره‌ها در زمان حذف و اهمیت به‌روزرسانی صحیح اشاره‌گرها برای جلوگیری از خطاهای حافظه است.

آزاد کردن حافظه:

در برنامه‌هایی که از تخصیص پویا (Dynamic Allocation) استفاده می‌کنند، مدیریت صحیح حافظه از مهم‌ترین وظایف برنامه‌نویس است. در لیست پیوندی، هر گره به‌صورت جداگانه در حافظه ایجاد می‌شود و در نتیجه، لازم است پس از اتمام استفاده از لیست، تمام این گره‌ها به‌صورت منظم آزاد شوند.

تابع آزادسازی حافظه در این پروژه، وظیفه دارد با پیمایش لیست از ابتدا تا انتها، تک‌تک گره‌ها را حذف کرده و جلوگیری **(Memory Leak)** حافظه‌ی اشغال‌شده توسط آن‌ها را آزاد کند. این کار نه‌تنها از **نشت حافظه** می‌کند، بلکه تضمین می‌نماید که برنامه پس از پایان اجرا، هیچ منبعی از سیستم را بی‌دلیل در اختیار نداشته باشد.

اگر آزادسازی حافظه انجام نشود، فضای اشغال‌شده توسط گره‌های حذف‌نشده تا پایان اجرای برنامه در حافظه باقی می‌ماند و در برنامه‌های بزرگ‌تر یا در سیستم‌هایی با منابع محدود، این موضوع می‌تواند منجر به افت عملکرد یا حتی از کار افتادن برنامه شود.

و اجرای دقیق آن، بخشی ضروری از طراحی هر ساختمان‌داده‌ی پویا Deallocate به همین دلیل، درک عملکرد تابع که در C++، محسوب می‌شود. این تابع نمونه‌ای است از رعایت اصول صحیح مدیریت حافظه در زبان‌هایی مانند آن‌ها مسئولیت آزادسازی منابع مستقیماً بر عهده‌ی برنامه‌نویس است.

```
123 void deallocate(struct List &list)
124 {
125     /*
126      * It's important to release all the resources you have allocated
127      * through the program. By calling this function, it traverses the linked list and
128      * deallocates memory
129      */
130     node_ptr cur = list.start;
131     while(cur)
132     {
133         node_ptr tmp = cur;
134         cur = tmp->right;
135         delete tmp;
136     }
137     list.length = 0;
138     list.start = nullptr;
139 }
```

کد تمرین:

در این تمرین، هدف اصلی آشنایی با منطق این ساختار داده و درک ارتباط میان عناصر آن است، نه صرفاً پیاده‌سازی مکانیکی آن. به همین دلیل پیاده سازی بیشتر اجزای این لیست در اختیار شما قرار می گیرد و شما کافی است که با مطالعه این بخش ها و درک نحوه کارکرد این داده ساختار به پیاده سازی توابع درج (insert) و پیمایش(traverse) بپردازید.

نکات تکمیلی:

. تفاوت nullptr و NULL:

یکی از تفاوت های مهم این است که کامپایلر به صلاح دید خود NULL را به عدد(صفر) یا boolean(false)، cast می کند که ممکن است به خطا های منطقی منجر شود برای مثال:

```
1 void func(void *hello) cout << "pointer says hello";
2 void func(int hello) cout << "integer says hello";
```

با دادن NULL به عنوان ورودی به تابع بالا،خروجی شما بسته به کامپایلر ممکن است مورد دوم باشد یا اول! اما nullptr همواره مورد اول را صدا می زند.

. خطا در اختصاص حافظه:

در زبان C++، زمانی که از عملگر new برای اختصاص حافظه استفاده می‌کنیم، سیستم عامل باید فضایی متناسب با درخواست برنامه تخصیص دهد. اما در شرایطی خاص، مانند کمبود حافظه یا محدودیت‌های سیستم، ممکن است این درخواست **با خطا مواجه شود**. در چنین حالتی، عملگر new نمی‌تواند فضای لازم را اختصاص دهد و معمولاً یک استثنا (Exception) از نوع `std::bad_alloc` ایجاد می‌کند.

اگر برنامه‌نویس این امکان را در نظر نگیرد، ممکن است برنامه به‌صورت ناگهانی متوقف شود یا رفتاری غیرقابل پیش‌بینی از خود نشان دهد. به همین دلیل، توصیه می‌شود همیشه در پروژه‌های بزرگ‌تر یا حساس‌تر، احتمال شکست در تخصیص حافظه بررسی و مدیریت شود.