بسم الله الرحمن الرحیم

گزارش تمرین سری سوم درس برنامه نویسی پیشرفته

سیده معصومه سجادی

ش.د:۹۳۲۳۰۸۷

فروردین ۱۳۹۸

تمرینات C++

✓ سوال ۱:

در این سوال قصد داریم MaxHeapرا پیاده سازی کنیم. همانطور که در صورت سوال آمده است بع تعدادی تابع و تعریف counstructor هایی برای رسیدن به منظور سوال نیازمندیم ؛ در ابتدا یک کلاس Maxheap تعریف کرده و برایش Makefile را نیز میسازیم سپس در Maxheap.h به تعریف کانستراکتورها میپردازیم:

```
class Maxheap
{
   public : //constructors
        Maxheap();
        Maxheap(int arr[] , int n);
        Maxheap(const Maxheap& Maxheap);
```

در فایل Maxheap.cpp بدنه ی این توابع را مینویسیم:

```
Maxheap:: Maxheap(){
    std::cout<<"defult constructor created"<<std::endl;
    arr.push_back(0);
    }

Maxheap:: Maxheap(int arr[] , int n){
     std::cout<<"array condtructor created"<<std::endl;
     arr.push_back(0);
     arr.insert(arr.end(), input.begin(), input.end());
     for(int i =heapsize() / 2; i > 0; --i)
          Heapify(i);
    }

Maxheap:: Maxheap(const Maxheap& Maxheap){
      std::cout<<"copy condtructor created"<<std::endl;
      for(size_t i = 0; i < Maxheap.heapsize(); i++)
         arr.push_back(Maxheap.arr[i]);

    }
```

همانطور که در کدها کامنت گذاری شده است انواع دیفالت کانستراکتور و کانستراکتور با ورودی آرایه و همچنین کپی کانستراکتور را تعریف کرده ایم و برای هر کدام متناسب با عمل خواسته شده عمل کرده ایم.

تابع بعدی Heapify است که برای قسمت الگوریتم اصلی این روش به کار رفته و حالت باینری و منطقی روش ساخت یک درخت را با این روش نشان میدهد و همواره مقدار بیشتر را در ریشه ی درخت قرار میدهیم:

```
void Maxheap::Heapify(int arr[], int i){
    int largest = i; // setting largest to i first time
    int L = LeftChild(i);
    int R = RightChild(i);
    if (L <= heapSize && arr[L] > arr[i]) // compare with arr[i] is not wrong but doesn't express the intent
        largest = L;
    else{
        largest = i; // setting largest to i second time
    }

    if (R <= heapSize && arr[R] > arr[largest]){
        largest = R;
    }
    if(largest != i){
        int temp = arr[i];      // these 3 lines are the std::swap
        arr[i] = arr[largest];  // or you could roll your own function that does the same.
        arr[largest] = temp;    // better expressing the intent.
        Heapify(arr, largest);
    }
}
```

سایر توابع زیر هم تعریف میشوند که در کد ها به بیان تعاریف آنها پرداخته شده و از تکرار مجدد آن پرهیز میکنیم.

add(key) که گره با مقدار keyرا به درخت در جایگاه مناسب، اضافه می نماید.

Delete () که در هربار فراخوانی، گره با مقدار maxرا از درخت حذف کرده و درخت را مرتب می نماید تا دوباره Maxheapساخته شود.

Max () که کلید گره با بزرگترین مقدار را باز می گرداند.

getHeight () که ارتفاع درخت را بازگردانی می کند.

Parent(i) که مقدار کلید والد گره iم را باز می گرداند.

RightChild(i) ،LeftChild(i) که به ترتیب مقادیر فرزند سمت راست و فرزند سمت چپ را بازمی گرداند.

printArray () که در کد های ما با show نوشته شده است.

نمونه خروجی:

```
64,
53, 59,
42, 34, 56, 24,
18, 27, 25, 22, 40, 20, 10, 5,
8, 9,

Press any key to continue . . .
```

- ✓ سوال ۲:
- ✓ در این برنامه قصـد داریم تا مرحله به مرحله، کلاس vector که در کتابخانه اسـتاندارد std در C++ وجود دارد را پیاده سـازی کنیم. به همین منظور، مراحل تعیین شـده در صـورت سـوال را گام به گام پیش می بریم تا خروجی مطلوب حاصل شود.
- ✓ (a) : در این بخش، دو تابع display و push_back را تعریف می کنیم. تابع display برای نشـان دادن عناصـر بردار استفاده می شود. در هر بار استفاده از تابع push_back، یک آرایه دینامیک جدید با طول یکی بیشتر تولید می شود. سپس آرایه پیشین حذف شده و آرایه جدید تولید شده جایگزین آن می گردد.
- ✓ (b): در این مرحله، تابع display را به کد main منتقل می کنیم. به منظور کارکرد صـحیح کد، لازم اسـت که اپراتور "[]" و تابع ()size را تعریف نماییم.
- ✓ (c): پس از انجام مراحل فوق، یک تابع ()Show تعریف می کنیم برای اینکه عنصـری از بردار را طبق نمونه گفته شده در تابع main نشان دهد.
- ✓ (d): در این بخش همانند تعریف تابع push_back، یک تابع pop_back تعریف می نماییم ک عنصـر آخر بردار را بر حسب تعداد دفعات فراخوانی این تابع در main پاک کند.
- ✓ (e): در این مرحله، دو Constructor جدید برای کلاس تعریف می کنیم. Constructor دو ورودی، یک آرایه به ابعاد ورودی اول و با مقداری برابر با مقدار ورودی دوم برای هر عضـو آرایه، ایجاد می کند. Constructor تک ورودی نیز یک آرایه تک عضوی با مقداری برابر با مقدار ورودی ایجاد می نماید.
- ✓ (f): در این بخش، اپراتور ضرب * ، جمع + < = > (با کارکردی مطابق خواسته صورت مسئله) تعریف می گردد. در داخل تابع این اپراتور یک آرایه موقت دینامیک تعریف می نماییم و خروجی مورد نظر را در آن تولید می کنیم. سپس آن شی را به کمک Copy Constructor به v۲ اعمال می کنیم.
- ✓ (g) move constructorرا نیز برای این کلاس تعریف میکنیم.

- ✓ (h): در مرحله آخر، کار را با تعریف تابع ()max به اتمام می رسانیم. در این تابع هر عنصر بردار با عنصر قبلی مقایسه می شود و پس از چند مرحله عنصر بزرگتر در خانه آخر بردار به صورت صعودی قرار می گیرد و در نهایت ماکزیمم مقدار بردار برگردانده می شود.

تفاوت بین Copy constructor و Move counstructor و نحوه ی تعریف و استفاده ی آنها:

- اگر در تعریف Copy constructor از آدرس دهی ارجاعی (reference) استفاده نکنیم، هنگام فراخوانی عملیات کپی برداری، شیء مورد نظر به تابع ار سال نمی شود و نیاز ا ست یک کپی از آن ایجاد شود. این کپی برداری مجددا تابع Copy constructor را فرا می خواند. بنابراین برنامه وارد یک حلقه بی نهایت از اجرای Copy constructor شده و عملکرد آن با مشکل مواجه خواهد شد. بهتر است از عبارت const در تعریف آرگومان به منظور جلوگیری از تغییر ناخواسته ورودی در بدنه تابع، استفاده نماییم.

- در Move constructor از کپی برداری از متغیر های شیء مبدا و ایجاد متغیر های موقت جلوگیری می گردد. این نوع constructor در صورت مواجهه با یک آرایه دینامیک، پوینتر آن را برای خود می کند. به عبارت دیگر، اعضای آرایه دینامیک در خانه دیگری از حافظه کپی نمی شوند بلکه همان اعضای فعلی، با پوینتری جدید شنا سایی خواهند شد. این کار منجر به بهینه سازی قابل توجهی در زمان و حافظه مورد نظر خواهد شد. همچنین این constructor با اجزای استاتیک نیز به عنوان rvalue رفتار می کند(در مواقعی که برنامه نمی تواند از RVO استفاده کند)، که این موضوع هم کد را بهینه می نماید. در مورد Move semantics هم می توان گفت که باید دو مورد ورژن Move اپراتور مساوی و همچنین Move constructor را در برنامه خود داشته باشیم.

هنگام استفاده از آرایه های دینامیک، اگر از Copy constructor استفاده نکنیم، تنها پوینتر آرایه کپی برداری می شود. در نتیجه این آرایه یک بار زمان اجرای Destructor برای شیء اول و یک بار هم زمان اجرای Destructor شیء دوم پاکسازی می شود. این مو ضوع منجر به آن خواهد شد که در بار دوم، اقدام به پاک کردن بخ شی از حافظه کنیم که دیگر در اختیار ما نیست و پیش از این پاک شده ا ست. در نتیجه با خطای پاکسازی مجدد رو به رو خواهیم شد. در این حالت نو شتن Copy constructor ضروری می باشد.

- ✓ سوال ۳: