Opgave 1: Adventure game

Leerdoelen

Na afloop van deze opdracht ben je in staat om:

- het verschil tussen een object en een pointer naar dat object te herkennen;
- met behulp van de address-operator, &, het adres van een object bepalen;
- met de dereferencing operator, $*$, een pointer omzetten in de het aangewezen object;
- de notatie `p->x` te gebruiken als afkorting van `(*p).x`;
- gebruik te maken van een datastructuur met pointers;
- te bepalen wanneer objecten *dynamische gealloceerd* moeten worden en wanneer niet;
- de relatie tussen een array index en pointers te gebruiken.

Achtergrond

Pointer zijn verwijzingen naar, of adressen van, objecten. Deze opdracht laat je oefenen met pointers, en dynamische geheugenallocatie. Om dit te doen dien je een datastructuur te realiseren voor een *adventure game* en de algoritmen om door deze datastructuur heen te lopen.

Probleemschets

De datastructuren in deze opgave representeren een netwerk van *kamers* en verbindingen tussen die kamers. Per kamer wordt de naam van kamer, een informatietekst (beide als `char []` van willekeurige lengte), een boolean die aangeeft of deze kamer een eindpunt is, en een reeks van uitgangen (toegangen tot andere kamers) bijgehouden. Iedere uitgang bestaat uit een tekst die een beschrijving geeft (de *sleutel*) en een verwijzing naar de kamer tot welke de betreffende deur toegang geeft. Ook hier is de beschrijving van de uitgang willekeurig lang en zijn er willekeurig veel uitgangen. Uitgangen worden opgeslagen en weergegeven in de volgorde waarin ze worden toegevoegd aan een kamer. De laatst toegevoegde uitgang komt dus achteraan. Het type `Kamer` representeren we met een klasse die, naast een constructor, tenminste één methode bevat om een uitgang aan de kamer toe te voegen. Breid zelf de klasse uit met de attributen en methoden, die je nodig hebt.

```
class Kamer
{
private: // de attributen van deze klasse hier toevoegen
public:
    Kamer (char _naam [], char _beschrijving [], bool _eind);
    void nieuweUitgang (char _sleutel [], Kamer* _naar);
};
```

De uitgangen van kamers worden gerepresenteerd door een structuur, genaamd `Uitgang`, die (1) de sleutel (de beschrijving van die uitgang), (2) de verwijzing van de kamer waar de uitgang heen gaat, en (3) een verwijzing naar de volgende uitgang bevat. Bij iedere instantie van `Kamer` hoort een reeks (lees: gelinkt lijst) van uitgangen. Ieder van die uitgangen wordt gerepresenteerd door een instantie van de structuur `Uitgang`. Het einde van deze reeks wordt op de gebruikelijke manier aangegeven, d.w.z. met een `NULL`-pointer.

Hiermee kunnen we bijvoorbeeld de volgende datastructuur definiëren:

```
Kamer k1 = Kamer ("Startkamer", "Dit is het begin van de reis.", false ) ;
Kamer k2 = Kamer ("Eindstation", "Dit is de laatste kamer.", true ) ;

k1.nieuweUitgang ("blijf hier", &k1 ) ;
k1.nieuweUitgang ("ga verder", &k2 ) ;
k2.nieuweUitgang ("opnieuw", &k1 ) ;
```

1 Tekenen van de datastructuur

Teken de datastructuur (maak het je gemakkelijk: 'gewoon op een stuk papier' of een tekenprogramma naar keuze) zoals die door bovenstaand programmafragment gedefinieerd wordt.

Ga na of je opzet voldoet aan de gegeven eisen. Kunnen alle teksten willekeurig lang zijn en is het altijd mogelijk om een extra `Kamer` te maken en een `Uitgang` toe te voegen aan een bestaande `Kamer`?

2 Ontwerp en implementeer de klasse Kamer

Ontwerp en implementeer de hele klasse `Kamer`. Belangrijk ontwerpcriterium hierbij is dat je geen aanname mag doen over het maximale aantal kamers, de lengte van de teksten of het aantal uitgangen. Komt de laatst toegevoegde keuze inderdaad achteraan?

De datastructuur doorlopen

Voor de adventure game heb je ook een algoritme om door deze datastructuur heen te lopen nodig. Iedere keer dat je in een kamer komt wordt de naam van die kamer afgedrukt. Als je voor het eerst in een kamer komt wordt ook de beschrijving van die kamer getoond. Vervolgens wordt de tekst `Uitgangen` : afgedrukt, gevolgd door de teksten die horen bij de uitgangen van deze kamer. Daarna vraagt het programma `Maak een keuze`, en leest de keuze van de gebruiker in. Het programma kiest de eerste mogelijkheid waarvoor de invoer een niet lege prefix (dus de eerste letter(s)) is van de tekst die bij die uitgang is opgeslagen. Indien geen van de uitgangen hieraan voldoet krijgt de gebruiker de melding `Onbegrepen keuze , je blijft hier.` Zodra de gebruiker op een eindpunt komt krijgt hij de mogelijkheid om te stoppen als hij dat wil.

3 De doolhof doorlopen

Ontwerp en implementeer de noodzakelijke methoden om interactief door deze datastructuur (de doolhof) heen te wandelen. Gebruik de tekening om te bepalen wat voor een variabelen dit algoritme nodig heeft om de huidige positie te kennen en de uitgangen te tonen. Je kunt je programma testen met de voorbeeld doolhof uit deze opgave.

4 Een doolhof inlezen

Als je met grotere doolhoven wil werken is het onhandig om die vast in je programma in te bakken. Het is beter om de doolhof uit een file te lezen. Om je op weg te helpen vind je op Bb code die een doolhof inleest en afdrukt. Ook staat daar een voorbeeld doolhof. Pas deze code aan zodat deze de doolhof opbouwt waar je programma doorheen loopt.

Inleveren van je producten

Als producten moet je een schets hebben van de gebruikte datastructuur en (uitgeteste!) C++-code. Neem de uitvoer van je programma (b.v. voor de kortste weg naar de uitgang) als commentaar op achter je code in het .cpp-bestand.

Lever vóór vrijdag 4 februari 13.30 uur via Blackboard jullie C++ code en de schets van de datastructuur in. De schets van de datastructuur kun je ook op het practicum aan de studentassistent geven. Vergeet niet om **in** je uitwerking duidelijk jullie namen te vermelden.