Zasady ogólne

Zadanie projektowe powinno być realizowane w zespołach dwuosobowych. W szczególnych przypadkach możliwa jest realizacja projektu w zespole jednoosobowym, jednak taki fakt musi być zawsze zgłaszany prowadzącemu.

Należy użyć właściwych algorytmów i starannie zaprojektować obiekty i powiązania pomiędzy nimi.

Implementacja powinna wykorzystywać odpowiednie mechanizmy dostarczane przez bibliotekę standardową oraz biblioteki Boost (np. obsługa błędów przez wyjątki, sprytne wskaźniki do zarządzania zasobami, hierarchie klas, jeżeli trzeba to szablony, odpowiednie wzorce projektowe itd.).

Należy w przemyślany sposób podzielić źródła na moduły.

Należy zapewnić możliwość kompilacji i uruchamiania dostarczanej aplikacji na różnych platformach, w szczególności powinna być ona kompilowana przez g++ pod Linux-em oraz przez Visual Studio.

Zasady oceniania

dokumentacja wstępna	1 pkt
projekt, algorytmy, klasy i powiązania pomiędzy nimi	15 pkt
styl kodowania	10 pkt
przenośność	4 pkt
RAZEM	30 pkt

Terminy

ustalenie składu zespołów, przydział zadań	20.04 godz. 10.00
oddanie dokumentacji wstępnej (max. 2 strony A4)	27.04 godz. 10.00
dostarczenie implementacji i dokumentacji końcowej (max. 4 strony A4)	1.06 godz. 10.00
termin rezerwowy (max. ocena z projektu 20 pkt)	15.06

Zadanie 1 - zachłanne przeszukiwanie stanów

Zaproponować abstrakcyjny interfejs dla klas reprezentujących stany. Stan tutaj jest rozumiany jako węzeł w grafie skierowanym, który ma powiązania do węzłów potomnych i do węzła rodzicielskiego. Najważniejsze metody tej klasy są następujące:

- badanie, czy węzeł jest stanem końcowym,
- operator porównania,
- generowanie stanów następnych (tworzenie potomków).

Zaimplementować następujące algorytmy przeszukiwania grafów (wykorzystujące powyższy interfejs): wgłąb, wszerz, od końca (dokładniejsze wyjaśnienie algorytmów - ustnie).

Wykorzystując powyższe klasy i algorytmy napisać program, który rozwiązuje zadanie przelewania.

Zadanie przelewania : W jaki sposób można uzyskać pojemność **C** płynu, przy użyciu naczyń o pojemnościach **A** i **B** (**A**, **B**, **C** są całkowite)? Można dowolnie dolewać, wylewać, przelewać płyn.

Zadanie 2 - heurystyki przy przeszukiwaniu stanów

Zaproponować abstrakcyjny interfejs dla klas reprezentujących stany. Stan tutaj jest rozumiany jako węzeł w grafie skierowanym, który ma powiązania do węzłów potomnych i do węzła rodzicielskiego. Najważniejsze metody tej klasy są następujące:

- szacowanie 'odległości' danego węzła od stanu końcowego. Dla stanu końcowego ta metoda zwraca 0, dla stanów, które nie są końcowe - wartość większą od 0.
- operator porównania,
- generowanie stanów następnych (tworzenie potomków),

Zaimplementować następujące algorytm A* przeszukiwania grafu. Algorytm ten analizuje stany (poczynając od początkowego), wybierając ten spośród nierozpatrzonych, który ma najmniejszą wartość f(n)=g(n)+h(n), gdzie g(n) jest odległością od stanu początkowego (dla bezpośrednich potomków stanu początkowego g(n)=1), zaś h(n) jest szacowaną odległością od danego węzła do stanu końcowego.

Wykorzystując powyższe klasy i algorytmy napisać program, który układa 'ósemkę', 'piętnastkę', itp. (ang. 8-puzzle, 15-puzzle). Do szacowania 'odległości' można wykorzystać np. ilość liczb, które nie są na swoich miejscach.

n-puzzle Dany jest kwadrat (o boku n), który posiada n^2 komórek zawierających liczby $1, 2, ..., n^2-1$. Jedna z komórek jest pusta. Należy danego początkowego ustawienia komórek podać sekwencję ruchów (jeżeli taka istnieje), która doprowadzi do ustawienia końcowego (liczby rozmieszczone kolejno rzędami, prawa dolna komórka pusta). Dozwolone jest 'przesuwanie' liczby na puste pole.

Przykładowy (dla $n = 3$) początkowy układ jest pokazany z lewej strony. Z tego stanu dozwolone są 3 ruchy: przesunięcie '6' w dół, przesunięcie '7' w prawo, przesunięcie '8' w górę. Ustawienie końcowe

1	4	6
2	7	-
3	5	8

1	2	3
4	5	6
7	8	-

Zadanie 3 - komenda dla edytora

Napisać prosty edytor dla gry przygodowej. Gra taka składa się z pomieszczeń, zawierających przedmioty, które są połączone ze sobą. Edytor pozwala na następujące operacje:

- operacje edytorskie:
  - dodawanie, usuwanie pomieszczeń, tworzenie połączeń (przejść) pomiędzy pomieszczeniami.
  - dodawanie, usuwanie przedmiotów w danym pomieszczeniu
  - wycofania wcześniejszej operacji (dowolna liczba zagłębień);
- możliwość poruszania się na aktualnej planszy: proste operacje przechodzenia do kolejnych pomieszczeń oraz wyświetlanie znajdujących się tam przedmiotów,
- zapis planszy do pliku, odczyt planszy z pliku.

Wykorzystać wzorzec projektowy komendy (Command, patrz np. wikipedia http://en.wikipedia.org/wiki/Design_pattern). Komenda pozwala między innymi na tworzenie kolekcji wykonanych operacji, co można wykorzystać np. do wycofywania operacji (undo).

Zadanie 4 - kompozycja i XML

Napisać edytor pozwalający obrabiać dokumenty o strukturze hierarchicznej, np. uproszczone dokumenty XML-owe (w razie potrzeby dostarczę opis formatu). Edytor pozwala na następujące operacje:

- dodawanie, usuwanie elementów, atrybutów,
- modyfikacja elementów: dodawanie, usuwanie podelementów, zmiana tekstu,
- zapis do pliku, odczyt z pliku.

Wykorzystać wzorzec projektowy kompozycji (Composite, opisany np. w wikipedii http://en.wikipedia.org/wiki/Design_pattern). Wzorzec ten pozwala reprezentować drzewiaste struktury obiektów.

Zadanie 5 - parser i kalkulator

Napisać program kalkulatora dla liczb rzeczywistych. Działania: dodawanie, odejmowanie, mnożenie, dzielenie, podnoszenie do potęgi, funkcje trygonometryczne, funkcje logarytmiczne. Priorytety operacji naturalne (tzn. mnożenie przed dodawaniem). Istnieje możliwość użycia nawiasów do zmiany priorytetów. Możliwość deklarowania i używania zmiennych. Dane czytane ze strumienia wejściowego. Wynikiem jest wartość użytych zmiennych.

Przykładowo, dla wejścia:

```
a = 2 + 2^(2+1)
b = a^2
```

wynik jest następujący

```
a = 10
b = 100
```

Problemy występujące w zadaniu:

- obliczanie wartości wyrażeń zawierających nawiasy, gdy operatory mają różne priorytety;
- możliwość używania zmiennych w wyrażeniach.

Wykorzystać algorytm rekurencyjny do obliczania wyrażeń.

Zadanie 6 - strumienie RSA

Napisać bibliotekę (zestaw klas), które pozwalają zapisywać dane do pliku. Dane pojawiające się w plikach powinny być zakodowane RSA (klucz 64 bitowy). Założenia upraszczające: strumienie obsługują tylko napisy (typ napisowy std::string), które zawierają znaki ASCII.

Klucze są dostarczane jako parametr programu. Napisać prosty program testujący możliwości biblioteki.

Problemy występujące w zadaniu:

- klasy dla arytmetyki w ciele Zp,
- klasy pozwalające kodować informację i rozkodowywać dla RSA,
- obsługa bufora,
- klasy zapisujące / odczytujące informację z plików.

Zadbać o to, aby interfejs był zgodny ze standardowym interfejsem dla strumieni (odpowiednie klasy powinny być pochodnymi std::ostream, std::istream).

Zadanie 7 - ustawienia i rekurencja

Zaprojektować interfejs dla klas reprezentujących figurę na planszy (figurą konkretną może być wieża, hetman, pion itd.). Interfejs powinien dostarczać m. in. metodę badającą, czy dane pole jest 'bite' przez figurę. Zaimplementować algorytm ustawiający zbiór figur w ten sposób, aby nie było 'bić'.

Algorytmem ustawiania figur może być np. rekurencja z nawrotami (Wirth, Algorytmy + struktury danych = programy). Proszę zwrócić uwagę, aby algorytm wykorzystywał jedynie metody interfejsu figury.

Prosty program testowy powinien pozwalać ustawiać k hetmanów, l wież oraz m gońców na planszy $n \times n$, gdzie k, l, m, n są parametrami wejściowymi programu. Wynikiem powinna być informacja, czy ustawienie jest możliwe dla zadanych parametrów oraz, jeżeli tak jedno z rozwiązań.

Zadanie 8 - uczenie się gry

Zaimplementować program uczący się grać w kółko i krzyżyk (plansza 3 na 3). Program powinien przechowywać drzewo gry i powinien „uczyć się na błędach", tzn. w przypadku przegranej eliminować niewłaściwe ścieżki.

Aby zmniejszyć rozmiar drzewa należy wykorzystać wzorzec proxy (opisany np. na stronie http://en.wikipedia.org/wiki/Design_pattern), który pozwala na „leniwe" tworzenie obiektów.

Program powinien przechowywać zdobytą wiedzę w pliku (najlepiej, gdy zapisuje drzewo gry po otrzymaniu polecenia zapisu od użytkownika). Powinna być możliwa gra z użytkownikiem oraz trening (gra z „trenerem" – sztucznym przeciwnikiem, który gra losowo).

Zadanie 9 - wizytacja figur

Napisać prosty kalkulator dla figur geometrycznych płaskich. Hierarchia figur jest ustalona i składa się z następujących bytów: klasa bazowa Figura, klasy pochodne: Okrąg, Prostokąt, Trójkąt, FiguraZłożona.

Zbiór operacje, które będą implementowane na obiektach wchodzących w skład hierarchii nie jest dokładnie ustalony (będzie się często zmieniał podczas tworzenia kolejnych wersji), dlatego zdecydowano na wykorzystanie wzorca wizytatora.

Zaimplementować wzorzec wizytatora (Design Pattern Visitor, opisany np. w wikipediihttp://en.wikipedia.org/wiki/Design_pattern). Wzorzec ten pozwala na przeglądanie obiektów należących do ustalonej hierarchii klas bez konieczności modyfikacji (dodawania metod) do tej hierarchii.

Kalkulator powinien (stosując wzorzec wizytatora) dostarczać następujących operacji:

- badanie sumarycznego pola figur w danym zbiorze,
- znajdowanie figury o największym polu,
- znajdowanie prostokąta o największym polu,
- znajdowanie okręgu o najmniejszym polu.

Ponadto należy dostarczyć następujących operacji pomocniczych:

- dodawanie, usuwanie figur ze zbioru,
- wyświetlanie informacji o figurach należących do danego zbioru,
- odczyt, zapis danego zbioru figur.

Zadanie 10 - singleton i obserwator

Zaimplementować bibliotekę (zestaw klas) pozwalających na prostą obsługę komunikatów, które pojawiają się w złożonych aplikacjach.

Biblioteka zapewnia istnienie jednej instancji obiektu w aplikacji, do którego będą wysyłane wiadomości tekstowe. Zapoznać się ze wzorcem singletona (informacje np. na stronie http://en.wikipedia.org/wiki/Design_pattern). Wykorzystać ten wzorzec do zapewnienia powyższej właściwości.

Wiadomości, które są wysyłane do wyżej opisanego obiektu mogą być zapisywane w różnych miejscach: mogą być wysyłane na standardowe wyjście (cout), mogą być wysyłane na wyjście błędów (cerr) lub na wyjście logów (clog), lub zapisywane do pliku (lub kilku plików). Aby zapewnić taką elastyczność należy użyć wzorca projektowego Obserwator (informacje o nim są dostępne na tej samej stronie).

Napisać klasę, umożliwiającą generowanie napisów z argumentami, podobna do `printf`, ale z kontrolą typów. Dla argumentów powinno się wykorzystywać operator strumieniowy.

Przykład:

```
struct Foo { };

std::ostream& operator<<(std::ostream& os, const Foo& f) {
  return os << "Foo";
}

Foo foo;
std::cout << print("Ha! %1%, x=%2% %1% %3%") % "Hej" % 1 % foo;
```

Spowoduje wypisanie na ekranie napisu

```
Ha! Hej, x=1 Hej Foo
```

Wykorzystać operator modulo (%).

Prosty program testowy powinien demonstrować możliwości wykorzystania biblioteki.

Zadanie 11 - prototypowanie dla hierarchii figur

Napisać program pozwalający obsługiwać figury płaskie, które mogą być rysowane np. na ekranie terminala w trybie tekstowym. Figury są następujące: Prostokąt, Trójkąt, Okrąg, FiguraZłożona (zawiera dowolną liczbę innych figur).

Ponieważ istnieje tutaj figura pozwalająca na grupowanie wielu obiektów (FiguraZłożona) tworzenie tzw. głębokiej kopii obiektów tego typu wymaga pewnego wsparcia, które dostarcza wzorzec Prototypu (Prototype, opisany np. na stronie http://en.wikipedia.org/wiki/Design_pattern). Należy zapoznać się z tym wzorcem i zaimplementować go.

Program obsługujący figury powinien dostarczać następujących metod:

- dodawanie, usuwanie figur ze zbioru,
- odczyt, zapis danego zbioru figur (do / z pliku),
- tworzenie kopii danej figury,
- modyfikacja danego obiektu (zmiana jego położenia),
- wyświetlanie istniejących obiektów (w trybie tekstowym na terminalu).

Zadanie 12 - drogi w grafach

Napisać aplikację pozwalającą znajdować trasę podróży. Dany jest zbiór miejscowości oraz zbiór dróg (każda droga łączy bezpośrednio dwie miejscowości). Aplikacja pozwala:

- znaleźć najkrótszą drogę pomiędzy zadanymi miejscowościami,
- znaleźć drogę pomiędzy miejscowościami, przechodzącą przez zadany zbiór innych miejscowości (które podróżny także chce odwiedzić)

Dane wejściowe (zbiór miejscowości, zbiór dróg, długości dróg) są umieszczone w pliku (plikach).

Zadanie 13 - strumienie i kompresja

Napisać bibliotekę (zestaw klas), które pozwalają zapisywać dane do pliku. Dane pojawiające się w plikach powinny być kompresowane. Założenia upraszczające: strumienie obsługują tylko napisy (typ napisowy std::string), które zawierają znaki ASCII.

Algorytm kompresji jest dowolny, proponuję wykorzystać kodowanie Huffmana. Ponieważ nie jest znana długość strumienia, należy dane wejściowe podzielić na tzw. paczki o ustalonej długości i kompresować niezależnie każdą paczkę.

Zadbać o to, aby interfejs był zgodny ze standardowym interfejsem dla strumieni (odpowiednie klasy powinny być pochodnymi std::ostream, std::istream).

Problemy występujące w zadaniu:

- klasy pozwalające kompresować dany napis,
- obsługa bufora,
- klasy zapisujące / odczytujące informację z plików.

Zadanie 14 - rekurencja i krzyżówki

Napisać program wspomagający układanie krzyżówek. Kształt krzyżówki oraz lista słów są parametrem wejściowym programu.

Problemy występujące w zadaniu:

- zapisywanie i odczyt informacji (tworzenie klas na podst. informacji zapisanej w pliku),
- efektywna implementacja algorytmu układającego krzyżówki.

Zadanie 15 - szablony grafów

Obsługa grafów. Dostarczyć kontenerów (generycznych, tak jak stl) przechowujących grafy. Szablon grafu jest parametryzowany typem danej przechowywanej w krawędzi oraz typem danej przechowywanej w wierzchołku. Wewnętrznie graf reprezentowany przez listę wierzchołków (lista sąsiedztwa). Umożliwić reprezentację grafu skierowanego oraz grafu nieskierowanego. Dostarczyć iteratorów umożliwiających przeglądanie grafu: iteratora dla wierzchołków, iteratora dla krawędzi.

Dostarczyć algorytm znajdowania najkrótszej ścieżki (Dijkstry) oraz oraz algorytm znajdowania ścieżki Hammiltona (problem komiwojażera).

Zadanie 16 - zarządca pamięci

Zarządca pamięci.

Napisać klasę, która przydziela pamięć dla obiektów o ustalonym rozmiarze (np. 4 bajtowych), gdzie rozmiar jest parametrem szablonu. Dla takiego zarządcy nie jest potrzebne „uciąglanie" pamięci. Wykorzystując odpowiedni zbiór powyższych obiektów napisać klasę, która przydziela pamięć dla małych obiektów (o rozmiarze od 4 do 32 bajtów); dla każdego rozmiaru istnieje niezależny zarządca.