Podstawy Programowania Komputerów

Struktury

2 listopada 2018

1

Zdefiniowano strukturę

```
struct punkt
{
   double x, y, z;
};
```

przechowującą punkt w przestrzeni trójwymiarowej. Proszę zaimplementować funkcje:

```
/** Funkcja wyznacza odległość między dwoma punktami wg pewnej metryki (np.
    euklidesowej).
@param a pierwszy z punktów
@param b drugi z punktów
@return odległość między punktami */
double odleglosc (const punkt & a, const punkt & b);
```

```
/** Funkcja losuje wartości współrzędnych punktów.
@param N liczba punktów do wylosowania
@return wektor N wylosowanych punktów */
std::vector<punkt> wylosuj_punkty (const int N);
```

```
/** Funkcja wyznacza parę najbardziej odległych punktów w chmurze punktów tr
    ójwymiarowych.
@param chmura zbiór punktów
@return para najbardziej odległych punktów */
std::pair<punkt,punkt> najdalsze_punkty (const std::vector<punkt> & chmura);
```

```
/** Funkcja sprawdza, czy trzy punkty leżą na jednej prostej.
@param a pierwszy z punktów
@param b drugi z punktów
@param c trzeci z punktów
@return true, gdy punkty są współliniowe;
        false, gdy punkty nie są współliniowe */
bool wspolliniowe (const punkt & a, const punkt & b, const punkt & c);
```

```
/** Funkcja wypisuje wartości współrzędncych punkty do strumienia.
@param str strumień do wypisania punktu
@param a punkt do wypisania */
void wypisz (std::ostream & str, const punkt & a);
```

```
/** Funkcja wypisuje chmurę punktów do strumienia.
@param str strumień do wypisania punktu
@param chmura zbior punktów do wypisania */
void wypisz (std::ostream & str, const std::vector<punkt> & a);
```

```
/** Funkcja wyznacza wartość pola trójkąta utworzonego przez trzy punkty.
@param a pierwszy z punktów
@param b drugi z punktów
@param c trzeci z punktów
@return pole trójkąta */
bool pole_trojkata (const punkt & a, const punkt & b, const punkt & c);
```

```
/** Funkcja wyznacza wartość najmniejszego niezerowego pola trójkąta
    utworzonego przez trzy punkty z chmury punktów.
@param chmura zbior punktów
@return pole najmniejszego niezerowego trójkąta */
double najmniejsze_pole_trojkata (const std::vector<punkt> & chmura);
```

```
/** Funkcja wyznacza wartość kąta utworzonego przez trzy punkty.
@param a pierwszy z punktów
@param b drugi z punktów
@param c trzeci z punktów
@return kąt utworzony przez trzy punkty w radianach */
double kat (const punkt & a, const punkt & b, const punkt & c);
```

```
/** Funkcja wyznacza odleglosc punktu od płaszczyzny utworzonej przez trzy
    punkty.
@param a pierwszy z punktów tworzących płaszczyznę
@param b drugi z punktów tworzących płaszczyznę
@param c trzeci z punktów tworzących płaszczyznę
@param z punkt, którego odległość od płaszczyzny wyznacza funkcja
@return kąt punktu z od płaszczyzny */
double odleglosc (const punkt & a, const punkt & b, const punkt & c, const
    punkt & z);
```

2

Zdefiniowano następujące typy

```
enum class kolor { pik, kier, karo, trefl };
enum class wartosc { as, krol, dama, walet, dziesiatka, dziewiatka, osemka,
    siodemka, szostka, piatka, czworka, trojka, dwojka};

/** karta do gry */
struct karta
{
    /** kolor karty */
    kolor _kolor;
    /** wartość karty */
    wartosc _wartosc;
};
```

```
/** ręka, czyli karty jednego gracza. */
typedef std::vector<karta> reka;
```

Proszę zaimplementować funkcje:

```
/** Funkcja tworzy 52-kartową talię kart. Na karty składają się karty
    czterech kolorów (pik, kier, karo, trefl), w każdym kolorze wartości od
    2 do 10, walet, dama, król, as.
@return utworzona talia */
std::vector<karta> utworz_talie_brydz ();
```

```
/** Funkcja tworzy 32-kartową talię kart. Na karty składają się karty
    czterech kolorów (pik, kier, karo, trefl), w każdym kolorze wartości od
    7 do 10, walet, dama, król, as.
@return utworzona talia */
std::vector<karta> utworz_talie_skat ();
```

```
/** Funkcja tworzy 24-kartową talię kart. Na karty składają się karty
    czterech kolorów (pik, kier, karo, trefl), w każdym kolorze wartości: 9,
     10, walet, dama, król, as.
@return utworzona talia */
std::vector<karta> utworz_talie_tysiac ();
```

```
/** Funkcja tasuje przekazaną talię kart.
@param talia talia kart do potasowania
@return potasowana talia */
std::vector<karta> potasuj (const std::vector<karta> & talia);
```

```
/** Funkcja przydziela karty z talii graczom.
@param talia talia kart do rozdania
@param liczba_graczy liczba graczy
@return zbiór rąk */
std::vector<reka> rozdaj (const std::vector<karta> & talia, const int
    liczba_graczy);
```

```
/** Funkcja wypisuje kartę.
@param str strumień do wypisania karty
@param k karta do wypisania  */
void wypisz (std::ostream & str, const karta & k);
```

```
/** Funkcja wypisuje karty w ręku.
@param str strumień do wypisania ręki
@param r reka do wypisania  */
void wypisz (std::ostream & str, const reka & r);
```

```
/** Funkcja wypisuje rozdanie.
@param str strumień do wypisania rozdania
@param rozdanie rozdanie do wypisania  */
void wypisz (std::ostream & str, const std::vector<reka> & rozdanie);
```