Zaawansowane programowanie komputerowe

Zadania przygotowawcze do kolokwium (2014)

A1. Napisać funkcję, która zwraca wartość true wtedy i tylko wtedy, gdy tablica t o długości n jest uporządkowana (rosnąco lub malejąco).

```
bool f(int* t, int n);
```

A2. Napisać funkcję, która zwraca medianę elementów z tablicy t o rozmiarze n, tj. dowolną liczbę m taką, że zbiory {i: t[i]>m} i {i:t[i]<m} są równoliczne.

```
int f(int* t, int n);
```

A3. Napisać funkcję

```
int f(int a, int b, int n);
```

która zlicza liczbę sposobów, na jakie można zapisać liczbę n w postaci sumy liczb a i b (utożsamiamy różne kolejności składników)

A4. Napisać funkcję

```
bool f(char* a, char* b);
```

która zwraca true wttw jeden napis jest przesunięciem drugiego. Np. przesunięciami napisu "ABCD" są napisy "BCDA", "CDAB", "DABC" i oczywiście "ABCD".

A5. Napisać funkcję

```
int f(int*t, int rozmiar);
```

która zwraca długość najdłuższego ciągu kolejnych elementów tablicy t tworzących ciąg arytmetyczny.

A6. Napisać funkcję
bool f(int n, int a, int b, int c)
która zwraca true wtedy i tylko wtedy, gdy n można przedstawić w postaci sumy, której wszystkie składniki są równe jednej z liczb a, b, c. Oszacować złożoność napisanej funkcji.

A7. Napisać funkcję
int g(int* t, int n)
która zwraca długość najdłuższego ciągu kolejnych elementów tablicy t o dodatniej sumie. Jeśli żaden z elementów tablicy nie jest dodatni, funkcja powinna zwrócić 0.

A8. Napisać funkcję
int f(int n, int a, int b)
która zwraca liczbę przedstawień liczby n w postaci sumy składników równych a lub b. Zakładamy, że wszystkie parametry są dodatnie. Rozróżniamy różne kolejności składników. Oszacować złożoność napisanej funkcji.

A9. Napisać funkcję
void g(double* t1, double* t2, double* t3, double* x, int n)
Parametry t1, t2, t3 to posortowane rosnąco tablice o rozmiarze n, a tablica t ma rozmiar 3n. Funkcja powinna przepisać elementy z tablic t1, t2, t3 do tablicy t tak, aby zachować porządek rosnący. Należy zadbać o to, aby funkcja działała jak najszybciej.

B1. Dana jest funkcja

```
int f(int n) {
      if(n<3) return 1;
      return f(n-1)+f(n-2)+f(n-3);
}
```

a. Obliczyć przybliżoną złożoność tej funkcji ze względu na parametr n.

b. Napisać równoważną funkcję nierekurencyjną.

B2. Dana jest funkcja

```
int f(int a, int b) {
	if(a<=0 || b<=0) return 1;
	return f(a-1, b)+f(a,b-1)+a*b;
}
```

a. Obliczyć przybliżoną złożoność tej funkcji.
b. Napisać równoważną funkcję nierekurencyjną.

B3. Napisać funkcję, która wypisuje na ekran wszystkie możliwe przedstawienia liczby n w postaci nierosnącej sumy liczb całkowitych dodatnich. Oszacować złożoność tej funkcji.

B4. Dana jest funkcja

```
int f(int k) {
	if(k) return 1+k%10+f(k/10);
	return 0;
}
```

Obliczyć złożoność tej funkcji w zależności od k. Napisać równoważną funkcję, która nie używa rekurencji.

B5. Dana jest funkcja

```
int f(int n, int k) {
	if(n==1) return 2;
	if(n%k==0)
		return k*k*f(n/k, k);
	else
		return f(n, k+1);
}
```

Obliczyć złożoność wywołania tej funkcji w zależności od n, jeśli parametr k jest równy 2. Napisać równoważną funkcję, która nie używa rekurencji.

C1. Stworzyć klasę Tablica przechowującą elementy typu double.

```
class Tablica {
public:
	Tablica(int n, double x); // tworzy nową tablicę wypełnioną
elementami x
	void ustaw(int i, double v); // ustawia wartość o wskazanym
indeksie
	double wartosc(int i); // zwraca wartość o podanym indeksie
	Tablica& dolacz(Tablica& t); // dołącza do tablicy elementy z
tablicy będącej parametrem
	int rozmiar(); // zwraca rozmiar tablicy
};
```

C2. Stworzyć klasę Czas, która przechowuje aktualny czas (godzinę i minutę)

```
class Czas {
public:
	Czas();
	Czas(int h, int m);
	Czas& dodajGodziny(int ileGodzin);
	Czas& dodajMinuty(int ileMinut);
	void drukuj(); // drukuje godzinę w formacie 12-godzinnym (np.
```

```
7:30AM lub 1:14PM)
      int zaIleMinut(Czas t); // podaje, za ile minut nastąpi podany
czas
};
```

C3. Napisać klasę ZbiorNapisow, która przechowuje zbiory napisów

```
class ZbiorNapisow {
public:
      ZbiorNapisow(); // tworzy pusty zbiór
      void dodaj(char* napis); // dodaje napis
      void usun(char* napis); // usuwa napis
      bool nalezy(char* napis); // sprawdza, czy podany napis należy do
zbioru
      int ile(); // zwraca liczbę napisów
      double sredniaDlugosc(); // zwraca średnią długość napisu ze
zbioru
};
```

C4. Labirynt to prostokątny budynek podzielony na kwadratowe pomieszczenia jednakowej wielkości. Pomieszczenia są ponumerowane parami liczb całkowitych (a,b), 0<=a<s, 0<=b<w, gdzie s i w są rozmiarami labiryntu. Pomiędzy sąsiednimi pomieszczeniami może znajdować się przejście, pomieszczenia skrajne nie mogą mieć wyjść na zewnątrz. Napisać klasę Labirynt, której obiekty reprezentują labirynty opisane powyżej.

```
class Labirynt {
public:
      Labirynt(int s, int w); // tworzy nowy Labirynt o rozmiarach s,w
bez przejść pomiędzy polami
      void wstawPrzejscie(int x, int y, char k); // wstawia przejście z
pomieszczenia (x,y) w kierunku k
      void usunPrzejscie(int x, int y, char k); // usuwa przejście z
pomieszczenia (x,y) w kierunku k
      bool jestPrzejscie(int x, int y, char k); // zwraca informację
czy jest przejście
}
```

C5. Napisać klasę T której obiekty reprezentują uporządkowane ciągi liczb typu double o długości nie większej niż 100.

```
class T {
public:
      T(); // tworzy nowy pusty ciąg
      void dodaj(double x); // dodaje nową liczbę x
      bool nalezy(double x); // zwraca przynależność
      void usun(double x); // usuwa podaną liczbę (tylko jedno wystąpienie)
      void usunWszytkie(double x); // usuwa wszystkie wystapienia
      double& operator[](int i); // zwraca i-ty najmniejszy element lub 0 jeśli indeks jest
niepoprawny
};
```

C6. Napisać klasę Drzewa, którek obiekty reprezentują zbiory drzew w lesie. Każde drzewo jest reprezentowane przez dwie współrzędne i literę kodującą gatunek drzewa. Zakładamy, że drzew jest nie więcej niż 100.

```
class Drzewa {
public:
      Drzewa(); // tworzy pusty las
```

```
        void dodaj(double x, double y, char gatunek); // dodaje nowe drzewo
        void usun(char gatunek); // usuwa wszystkie drzewa o podanym gatunku
        int ile(char gatunek); // zwraca liczbę drzew o podanym gatunku
        char najblizszy(double x, double y); // zwraca gatunek drzewa rosnącego najbliżej
zadanego punktu ('?' jeśli nie ma żadngo drzewa)
}
```