Zadania przygotowujące do kolokwium z Informatyki

Przygotowanie poniższych programów nie jest obowiązkowe, jednak warto zapoznać się z nimi, a najlepiej napisać wybrany przez siebie program w celu sprawdzenia swoich umiejętności.

1. Szyfrowanie tekstu

Wersja 1- dla dowolnych znaków.

Szyfrowany tekst jest wczytywany z klawiatury, a w postaci zaszyfrowanej jest zapisywany do pliku. Każdemu znakowi szyfrowanego tekstu odpowiada ciąg bajtów:
- losowy (jego kod, odczytany jako `unsigned char`, oznaczmy jako *n*, np. dla bajtu w zapisie binarnym: `00000011` *n*=3),
- *n* znaków (bajtów) losowych, (np. `10100111 01101110 01010011`),
- znak szyfrowanego tekstu, (np. `01000001` – znak `'A'`).

Tego typu zestawy znaków - o różnej liczebności, odpowiadający kolejnym znakom szyfrowanego tekstu, zapisywane jeden po drugim, bez żadnych ograniczników, odstępów itp. – tworzą ciąg zaszyfrowanego tekstu.
Druga faza szyfrowania polega na cyklicznym przesunięciu każdego znaku w lewo o *p* bitów (*p* jest liczbą, która odpowiada (przesuniętemu już) znakowi poprzedniemu (odczytanemu jako `unsigned char`), czyli pierwszy znak nie jest przesuwany, drugi jest przesuwany o liczbę bitów równą pierwszemu znakowi, trzeci znak jest przesuwany o liczbę bitów równą drugiemu (przesuniętemu) znakowi itd. (Przesunięcie cykliczne w lewo oznacza wpisanie bitu „wypychanego" (najstarszego) na pozycję najmłodszą). Dla ww. przykładu efekt 2. fazy wyglądałby następująco: `00000011 11110100 11100110 11010100 00010100` (liczba *p* miała wartości kolejno: 3, 244, 230, 212).
Proszę napisać:
– program szyfrujący i program deszyfrujący w wersji podstawowej - bez 2. fazy szyfrowania,
– program szyfrujący i program deszyfrujący w wersji rozszerzonej z 2. fazą szyfrowania i odczytać załączony plik `wiadomosc.dat`.
Uwaga:

1. *Jeżeli chcemy zminimalizować czas deszyfracji, to liczba przesunięć powinna być mała - do jakiej wartości możemy ją zmniejszyć?*
2. *Losowane znaki należą do przedziału 0 – 255. Można użyć funkcji rand().*

Wersja 2 - dla tekstu pisanego dużymi literami.

Zakładamy, że szyfrowany tekst oraz postać zaszyfrowana zawierają tylko duże litery lub znak @. Znak @ pełni rolę spacji.
1. faza szyfrowania przebiega tak jak w wersji 1. z tą różnicą, że losowane znaki są dużą literą lub znakiem @.
W 2. fazie bitowe przesunięcie cykliczne jest zastąpione „dodaniem" do znaku pewnej liczby *k* w taki sposób, aby wynik też był dozwolonym znakiem (dużą literą lub @). Np. dla *k*=2: 'A'+*k*->'C', 'N'+*k*->'P', 'X'+*k*->'Z', 'Y'+*k*->'@', 'Z'+*k*->'A', '@'+*k*->'B'. Można to nazwać cyklicznym przesunięciem liter w alfabecie (z dołączonym do alfabetu znakiem @).
W programie w wersji podstawowej nie ma fazy 2.
W wersji rozszerzonej *k* jest stałą, zadaną liczbą np. 5.
W wersji maksymalnej – liczba *k* jest „przesuniętym" już znakiem poprzednim (ew. liczbę *k* można każdorazowo pomniejszyć o kod znaku @, czyli o 64).

2. Test struktury nawiasów – stos (2a #if #elif #else #endif)

Program czyta z klawiatury ciąg znaków, wśród których są – dowolnie zagnieżdżone nawiasy (), [] oraz {}. Program ma sprawdzać, czy struktura nawiasów jest prawidłowa. Test może polegać na sprawdzaniu, czy w momencie odczytania nawiasu zamykającego ostatnio wczytanym nawiasem otwierającym był nawias właściwego typu. (*Czy to wystarczy?*) Należy napisać taki program z zastosowaniem stosu. W wersji podstawowej można zbudować stos w tablicy automatycznej, a w wersji rozszerzonej – w pamięci przydzielanej dynamicznie.
Program mógłby mieć opcję sprawdzania pliku źródłowego innego programu napisanego w języku C.

Jak zmodyfikować ten program aby mógł sprawdzać poprawność kolejności użytych w innym programie (pliku źródłowym) dyrektyw preprocesora: `#if #elif #else #endif` (załóżmy dla uproszczenia, dyrektywy te pisane byłyby od lewego marginesu)?

3. Kolejka zleceń w tablicy (po przeczytaniu fragmentu ostatniego rozdziału o tablicy w roli kolejki)

Należy napisać program symulujący kolejkę zleceń. Przyjęte, a jeszcze nie obsłużone zlecenia oczekują w kolejce – są zapisane w tablicy zleceń. Zlecenia są obsługiwane w kolejności ich przyjmowania – (first in first out). Program powinien zawierać funkcje obsługujące kolejkę: dopisanie nowego zlecenia, wyrejestrowanie obsłużonego zlecenia, informowanie o przepełnieniu bufora oraz o braku zleceń w kolejce.
Po każdej zmianie stanu kolejki program wyprowadza na ekran stan tablicy (bufora) zawierającej informacje o nieobsłużonych zleceniach. Puste miejsce w buforze jest zaznaczane znakiem _, np.: _ _ 3 4 5 _ _ _ _ _ oznacza, że tablica ma 10 elementów, w kolejce oczekują zlecenia nr 3, 4 i 5.
Symulacja w wersji podstawowej:

- wczytanie z klawiatury znaku I – przyjęcie zlecenia (każde zlecenie otrzymuje kolejny numer – identyfikator),
- wczytanie z klawiatury znaku O – obsługa zlecenia.

Symulacja w wersji rozszerzonej *(do realizacji w systemie Windows, z użyciem funkcji np. : kbhit(), getch(), getche() z biblioteki conio.h oraz time() z biblioteki time.h.)*:

- przyjęcie zlecenia – naciśnięcie dowolnego znaku (ten znak jest identyfikatorem zlecenia),
- obsługa każdego zlecenia trwa 5 sekund. Należy zadbać o to, aby przyjmowanie zleceń (wczytywanie znaków) nie wstrzymywało obsługi zleceń (program nie może czekać na zakończenie operacji wejścia i wstrzymywać przez ten czas obsługi innego zlecenia).

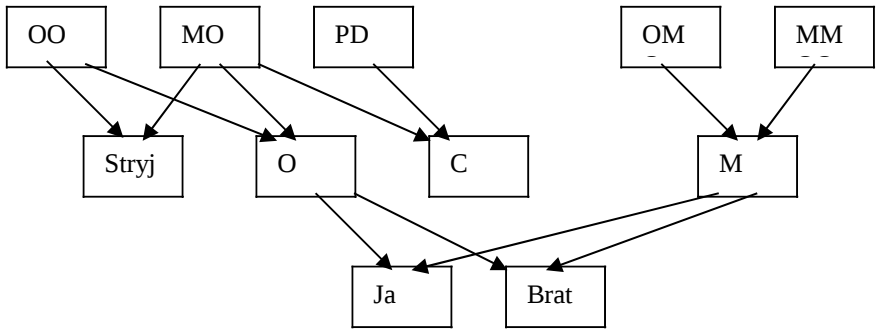
4. Test pokrewieństwa

Załóżmy, że mamy zgromadzić dane spokrewnionych osób. Dla każdej osoby należy zdefiniować zmienną strukturalną zawierającą:
- nazwisko,
- imię,
- PESEL

oraz informacje dotyczące pokrewieństwa w postaci wskaźników do:
- ojca,
- matki,
- dzieci (w wersji podstawowej struktura zawiera tablicę - np. 5 elementową – wskaźników do dzieci, a wersji rozszerzonej – wskaźnik do takiej tablicy alokowanej dynamicznie).

Ponadto w strukturze pamiętane jest:
- dla osoby żyjącej – miejscowość zamieszkania,
- dla osoby zmarłej – data zgonu.

Proszę napisać program, w którym te zmienne będą (w jak najprostszy sposób) wypełnione dla rodziny przedstawionej na poniższym schemacie (OM – ojciec matki, MO – matka ojca, strzałka biegnie od rodzica do dziecka). W przypadku braku danych o pokrewieństwie należy wstawić `NULL`.



Proszę napisać funkcję, której argumentem jest jedna ze struktur (lub jej adres). Funkcja ta wyprowadza na ekran dane rodzeństwa danej osoby (w wersji rozszerzonej – z rozróżnieniem rodzone / przyrodnie) – imię, nazwisko oraz miejsce zamieszkania albo datę zgonu. Funkcję należy testować wywołując ją dla struktury np. Ja lub Stryj.
Uwaga: proponowana struktura danych jest daleka od optymalnej – na dalszych ćwiczeniach poznamy lepsze.

5. Sortowanie tablicy łańcuchów

Należy napisać funkcję, która alfabetycznie sortuje długie łańcuchy znakowe.
Dla przetestowania tej funkcji należy w segmencie głównym zdefiniować dwuwymiarową tablicę znakową, np.: 2000x30. Do każdego wiersza należy wpisać losowo wygenerowany ciąg liter (i uzupełnić zerem tak, aby tworzył łańcuch). Następnie należy wywołać funkcję sortującą metodą np. bąbelkową. Po wyjściu z funkcji, w segmencie głównym należy wyprowadzić łańcuchy na ekran w porządku alfabetycznym.
W wersji rozszerzonej zminimalizować liczbę przepisywania łańcuchów z jednego miejsca na inne i ew. porównać czas wykonywania programów w wersji bez i z minimalizacją liczby przepisywań łańcuchów.
Uwaga: Należy zwrócić uwagę kiedy można zakończyć algorytm sortowania - nie należy przeglądać uporządkowanej już tablicy.

6. Znajdowanie miejsca zerowego funkcji –

wskaźniki do funkcji w argumentach funkcji

Metoda Newtona numerycznego znajdowania miejsca zerowego funkcji rzeczywistej $f(x)$ jednej zmiennej x polega na iteracyjnym obliczaniu kolejnych przybliżeń wg wzoru

$$x_{i+1} = x_i - \frac{f(x)}{f'(x)}$$

Chcemy uzupełnić bibliotekę matematyczną o procedurę (funkcję) `Newton` realizującą ten algorytm.
Wymagania dla użytkownika tej procedury byłyby następujące:
- napisać procedurę obliczającą wartość $f(x)$,
- napisać procedurę obliczającą wartość pochodnej funkcji $f(x)$,
- napisać segment `main`, w którym wywoływana będzie procedura `Newton` z nazwami ww. procedur jako parametry.

Dla przykładowej funkcji $f(x) = e^{-x} - x$ procedury te mogłyby mieć postać następującą:

```
#include <stdio.h>
#include <math.h>

double funkcja(double x) {  /* definicja funkcji f(x) */
      return exp(-x)-x;
}
double pochodna(double x) { /* definicja pochodnej funkcji f(x) */
      return -exp(-x)-1.0;
}

int main(void) {
int n_max=10;          /* maksymalna liczba iteracji */
double x0=0.0;      /* punkt startowy */
double x;
    x = Newton(x0,funkcja,pochodna,n_max);
    printf("Pierwiastek %20.15f, iteracji %d\n",x0,n);
    return 0;
}
```

Proszę napisać definicję funkcji `Newton`.