

ZADANIA Z JĘZYKA C DLA GRUP 7. I 9.

Zestaw IV - listopad/grudzień 2017

13. **Szyfrowanie plików.** Napisać prosty program, który szyfruje zawartość pliku za pomocą hasła wpisywanego z klawiatury. Nazwę pliku do zaszyfrowania podajemy jako argument wywołania programu ("`$./szyfruj mojplik.dat`"); plik o podanej nazwie jest otwierany do odczytu binarnego za pomocą instrukcji

```
if (NULL==(file1=fopen(Arg[1],"rb"))) return 1;
```

gdzie tablica wskaźników `char *Arg[]` jest parametrem funkcji `main` (por. Zadanie 6. z Zestawu II). Ponadto, otwieramy plik tymczasowy do zapisu binarnego

```
if (NULL==(file2=fopen(Tymczasowy,"wb"))) return 1;
```

którego nazwa generowana jest wcześniej funkcją `tmpnam(Tymczasowy)` z biblioteki standardowej. Zasadnicza część programu to czytanie kolejnych znaków funkcją `znak=getc(file1)`, szyfrowanie za pomocą operacji binarnej:

```
znak ^= haslo[ j %(dlugosc_hasla) ];
```

(`j` to numer kolejnego czytanego znaku, pozostałe zmienne objaśniam niżej) oraz zapis do pliku tymczasowego: `putc(...)`. Jak wspomniano wyżej, hasło (przechowywane w tablicy znakowej `haslo[...]` o pewnym założonym rozmiarze maksymalnym) może być wczytywane z klawiatury bezpośrednio po uruchomieniu programu, jego faktyczną długość (`dlugosc_hasla`) można określić z pomocą funkcji `strlen`. Do sprawdzania, czy nie osiągnięto końca pliku wejściowego, służy funkcja `feof(file1)`, która powinna być wywoływana bezpośrednio po każdorazowym wczytaniu znaku. Na koniec, należy użyć funkcji `unlink(Arg[1])` i `rename(...)`, tak aby zastąpić plik wejściowy plikiem zaszyfrowanym. Powtórne wywołanie programu, z tym samym hasłem, powinno odszyfrować zakodowaną wiadomość.

14. **Kalkulator.** Tzw. *odwrotna notacja polska* to wygodna (dla komputera ...) forma zapisu działań arytmetycznych i nie tylko. Przykładowo, działanie $(3+7) \cdot (11-1)$ będzie miało w tej notacji postać:

```
3 7 + 11 1 - *
```

Komputer interpretuje taki zapis następująco: każdy operand (tutaj operandami są po prostu liczby) jest odkładany na stos, każdy operator dwuargumentowy (tutaj: "+", "-", "*") powoduje zdjęcie ze stosu dwóch elementów i położenie wyniku operacji. Jak widać, nawiasy nie są potrzebne, zaś notacja jest jednoznaczna pod warunkiem, że ustalimy ilu argumentów potrzebuje dany operator:

a zatem, jeśli np. "-" oznacza odejmowanie, to nie może równocześnie oznaczać zmiany znaku ostatniego argumentu.

Napisać prosty, obsługiwany za pośrednictwem terminalu tekstowego, kalkulator interpretujący działania arytmetyczne wprowadzane w notacji odwrotnej polskiej. Do implementacji stosu można użyć zmiennych globalnych: tablicy liczb typu `double` o pewnym założonym rozmiarze maksymalnym, zmiennej typu `int` przechowującej aktualną liczbę argumentów na stosie, oraz dwóch funkcji:

```
void push(double);      double pop(void);
```

odpowiednio do odkładania argumentu na stosie i zciągania argumentu ze stosu (funkcje te powinny sprawdzać, czy maksymalny rozmiar stosu nie został przekroczony, lub czy stos nie jest pusty, i generować odpowiednie komunikaty wypisywane na ekran). Operatory i operandy można wczytywać instrukcją `scanf("%s", oper)`; gdzie `oper` to zadeklarowana wcześniej tablica znakowa (taka konstrukcja zapewnia poprawną interpretację standardowych separatorów). Później, należy dokonywać identyfikacji co zostało wczytane.

Po pierwsze, wprowadzenie liczby można rozpoznać z pomocą funkcji `isdigit` z pliku `<ctype.h>` (warto pamiętać o możliwości wystąpienia wiodącego minusa, jeśli dopuścimy działania na dla liczbach ujemnych). W pozostałych przypadkach pomocna będzie funkcja `strcmp` z nagłówka `<string.h>`. Przykładowo, fragment kodu odpowiedzialny za dodawanie może wyglądać tak:

```
if (!strcmp(wyr, "+")) {
    y=pop();
    x=pop();
    push(x+y);
}
```

gdzie `x` i `y` to zadeklarowane wcześniej zmienne typu `double`.

W wersji podstawowej, kalkulator powinien obsługiwać operatory: "+", "-", "*", "/"; jak również operator "=", który zdejmuje ostatni argument ze stosu i wypisuje na ekran. Aby zakończyć działanie kalkulatora, wprowadzamy ustalony znak, np. "q". Proszę samodzielnie zaproponować proste rozszerzenia kalkulatora, tak aby np. obliczał potęgi, logarytmy, funkcje trygonometryczne. Proszę zwrócić uwagę, że funkcja `strcmp` jest bardzo elastyczna, np. konstrukcja:

```
if (!strcmp(wyr, "moja-specjalna-funkcja")) {...}
```

zachowa się jak trzeba, o ile tylko na stosie znajdować się będzie potrzebna liczba argumentów.