Zajęcia 4 – procedury i funkcje

1. Zdefiniuj funkcję
   `int count_occurrences(const char str[], char c, int &first, int &last)`
   która w podanym łańcuchu (tablicy) `str` zlicza i zwraca jako wynik liczbę wystąpień znaku `c`.

   Dodatkowo funkcja poprzez parametry `first` oraz `last` zwraca indeksy *pierwszego* i *ostatniego* wystąpienia znaku `c`. Jeżeli znak ten nie występuje w `str`, to obie zmienne powinny otrzymać wartość -1.

   Przykład użycia funkcji `count_occurrences`:

```
int main() {
    char str[] = "abrakadabra";
    int l, r;
    int num_b = count_occurrences(str, 'b', l, r);
    std::cout << "Znak 'b' znaleziono " << num_b << " razy w \""
              << str << "\"\n"
              << "pierwszy raz na pozycji " << l
              << " a ostatni na pozycji " << r << '\n';

    for (int i = l; i <= r; ++i) {
        str[i] = '*';
    }
    std::cout << str << '\n';

    const char addr[] = "Katowice, Bankowa 12";
    std::cout << "Znak 'x' występuje w \"" << addr
              << "\" " << count_occurrences(str, 'x', l, r) << " razy\n"
    return 0;
}
```

   Wynik:

```
Znak 'b' znaleziono 2 razy w "abrakadabra"
pierwszy raz na pozycji 1 a ostatni na pozycji 8
a********ra
Znak 'x' występuje w "Katowice, Bankowa 12" 0 razy
```

2. Napisz funkcję
`void print(int tab[], int size, const char sep[] = " ")` która drukuje na standardowym wyjściu `size` pierwszych elementów podanej tablicy oddzielając poszczególne wydruki łańcuchem `sep`.

   Przykładowo, następujący fragment programu:

   ```
   const int N = 5;
   int punkty[N] = { 12, 5, 0, 20, 10 };

   std::cout << "punkty: ";
   print(punkty, N);

   std::cout << "\npunkty: ";
   print(punkty, N, ", ");
   ```

   powinien wydrukować

   ```
   punkty: 12 5 0 20 10
   punkty: 12, 5, 0, 20, 10
   ```

3. Funkcję z poprzedniego zadania rozszerz o *dwa* dodatkowe *opcjonalne* parametry typu `const char[]` oznaczające, odpowiednio, łańcuch do wyświetlenia przed pierwszym oraz za ostatnim elementem tablicy.

   Przykładowo, dla następującego fragmentu programu

   ```
   const int N = 5;
   int punkty[N] = { 12, 5, 0, 20, 10 };
   print(punkty, N, "</li>\n<li>", "<ul>\n<li>", "</li>\n</ul>");
   ```

   wydruk to

   ```
   <ul>
   <li>12</li>
   <li>5</li>
   <li>0</li>
   <li>20</li>
   <li>10</li>
   </ul>
   ```

4. Zaimplementuj funkcję rekurencyjną
`void kombinacje(std::string tab[], int n, int i=0, std::string s="")`
działającą zgodnie z poniższymi instrukcjami:

   a) Jeżeli `i == n`, to funkcja wyświetla na konsoli parametr `s` i kończy działanie.

b) Jeżeli `i < n`, to wywoływana jest (rekurencyjnie) funkcja `kombinacje` z parametrami:
   - `i+1` (pozostałe param. bez zmian);
   - `i+1` oraz param. `s` równemu łańcuchowi powstałemu ze sklejenia łańcucha `tab[i]` oraz poprzedniej wartości `s`.

Działanie funkcji ilustruje następujący fragment programu:

```
std::string miasta[] = { "Warszawa", "Krakow", "Gdynia" };
kombinacje(miasta, 3);
```

wynik:

```
{}
{Gdynia}
...
{Krakow, Warszawa}
{Gdynia, Krakow, Warszawa}
```

Wskazówka. Klasa `std::string` znajduje się w nagłówku `string`, a operację sklejania można zrealizować za pomocą operatora „+”.

5. Napisz funkcję, która zwraca wartość n-tego wyrazu ciągu Fibonacciego. Funkcja powinna być napisana w dwóch wersjach: iteracyjnej i rekurencyjnej.

   Przykład deklaracji funkcji:

   ```
   unsigned fib(unsigned n);
   unsigned fibRekurencyjna(unsigned n);
   ```

6. Napisz funkcję `strfind`, która szuka w tekście (pierwszy parametr) podanej frazy (drugi parametr). Wynikiem funkcji ma być indeks znaku, od którego podana fraza zaczyna się w tekście lub -1, jeżeli tekst nie zawiera szukanej frazy. Wielkość liter w podanych ciągach nie ma znaczenia.

   Przykład:

   Dla podanego fragmentu programu:

   ```
   char zdanie[] = "Jutro jest egzamin z programowania.";
   char fraza[] = "Program";
   cout << "Szukam w \"" << zdanie << "\"" << endl;
   cout << "\t\"" << fraza << "\" : " << strfind(zdanie, fraza) << endl;
   cout << "\t\"jutro JEST\" : " << strfind(zdanie, "jutro JEST") << endl;
   cout << "\t\"WDI\" : " << strfind(zdanie, "WDI") << endl;
   cout << "Szukam w \"" << fraza << "\"" << endl;
   cout << "\t\"" << zdanie << "\" : " << strfind(fraza, zdanie) << endl;
   cout << "\t\"" << fraza << "\" : " << strfind(fraza, fraza) << endl;
   ```

na ekranie powinno zostać wyświetlone:

```
Szukam w "Jutro jest egzamin z programowania."
    "Program" : 21
    "jutro JEST" : 0
    "WDI" : -1
Szukam w "Program"
    "Jutro jest egzamin z programowania." : -1
    "Program" : 0
```

7. Dywan Sierpińskiego to fraktal otrzymany z kwadratu za pomocą podzielenia go na dziewięć (3x3) mniejszych kwadratów, usunięcia środkowego kwadratu i ponownego rekurencyjnego zastosowania tej samej procedury do każdego z pozostałych ośmiu kwadratów.

   Napisz *rekurencyjną* funkcję, która „rysuje" dywan Sierpińskiego w konsoli za pomocą znaków '#' oraz spacji. Funkcja przyjmuje parametr n określający stopień szczegółowości rysowanego dywanu. Rysowanie odbywa się w dwuwymiarowej tablicy znaków, przy czym zaczerniony kwadrat oznaczany jest znakiem '#', natomiast pusty – spacją. Długość boku dywanu dla stopnia n wynosi 3^{n-1} znaków.
   Przykładowe wydruki dla kilku początkowych wartości n przedstawiono poniżej.

```
n=1   n=2    n=3              n=4

#     ###    #########        ###########################
      # #    # ## ## #        # ## ## ## ## ## ## ## ## #
      ###    #########        ###########################
             ###   ###        ###   ######   ######   ###
             # #   # #        # #   # ## #   # ## #   # #
             ###   ###        ###   ######   ######   ###
             #########        ###########################
             # ## ## #        # ## ## ## ## ## ## ## ## #
             #########        ###########################
                              #########        #########
                              # ## ## #        # ## ## #
                              #########        #########
                              ###   ###        ###   ###
                              # #   # #        # #   # #
                              ###   ###        ###   ###
                              #########        #########
                              # ## ## #        # ## ## #
                              #########        #########
                              ###########################
                              # ## ## ## ## ## ## ## ## #
                              ###########################
                              ###   ######   ######   ###
                              # #   # ## #   # ## #   # #
                              ###   ######   ######   ###
                              ###########################
                              # ## ## ## ## ## ## ## ## #
                              ###########################
```