Język C i C++. Podstawy

Materiały do samodzielnego opanowania, ale także propozycja zadań na zajęcia laboratoryjne.

Zagadnienia do opanowania

1) Czym się różnią pliki tekstowe od binarnych?
2) Jakie znaki końca wiersza obowiązują w różnych systemach operacyjnych?
3) Jak zbudowana jest kompletna ścieżka do pliku? Czy jest katalog, nazwa, rozszerzenie?
4) Jaki znak separatora elementów ścieżki pliku jest stosowany w różnych systemach operacyjnych? Na co należy zwrócić uwagę w systemach Windows?
5) Czym jest wskaźnik?
6) Czym jest tablica jednowymiarowa (lub wektor lub tablica jednoindeksowa)? Jak ją definiujemy w języku C?
7) Chcemy przechować w pamięci napis 30-znakowy. Ile minimalnie elementów powinna zawierać tablica znaków?
8) Czym się różnią w języku C te konstrukcje: A, 'A', "A", 0xA ? Po ile bajtów w pamięci zajmują 3 ostatnie z nich?
9) Jak inicjalizować tablice w języku C? Podaj kilka sposobów.
10) Jak wyświetlać zawartość tablicy znakowej (napis w niej zawarty)?
11) Jak wyświetlać wartości kolejnych elementów tablicy?
12) Jak policzyć długość napisu w języku C?
13) Jak zabezpieczyć program przed wczytaniem nadmiernej liczby znaków do tablicy (inaczej: jak zabezpieczyć program przed przepełnieniem buforów pamięciowych)?
14) Jakie mogą być skutki uzyskania dostępu (zapisania lub odczytania) do elementów leżących poza tablicą?
15) Używając pętli for i operatora indeksowania [ ] zamień małe łacińskie litery napisu na wielkie.
16) Używając pętli while i wskaźnika znakowego zamień małe łacińskie litery napisu na wielkie.
17) Wczytaj do programu 2 wartości: swój wiek do zmiennej całkowitej ‘wiek’ oraz imię (do tablicy znakowej lub obiektu string). Przygotuj napis mający postać zdania następującej postaci: "Osoba [imię] za 5 lat będzie miała [wiek+5] lat.". Wyświetl ten napis w konsoli.
18) Zmodyfikuj kod (17) w taki sposób, aby wyniki były zapisywane w pliku tekstowym.

Uwaga użytkownicy C++ i klas/obiektów string, cin, cout. Szczególną uwagę zwróćcie na zadania 13, 15, 16, 17. Rozwiążcie je, używając wygodnych w użyciu w/w obiektów.

Dodatkowe funkcje, przydatne w języku C do zabawy z napisami ASCIIZ (tablicami znakowymi), dostępne są po dodaniu pliku nagłówkowego <string.h> :

```
#include <string.h>
```

Przykład 1.

```
char p[] = { 'A', 'b', 'c' };
char q[] = { "Abc" };

printf("sizeof(p) == %d\n", sizeof(p));
printf("sizeof(q) == %d\n", sizeof(q));
printf("strlen(p) == %d\n", strlen(p));
printf("strlen(q) == %d\n", strlen(q));
```

Dlaczego takie wyniki? Przeanalizuj, użyj debugera, wyciągnij wnioski.

Przykład 2.

```
FILE *f1 = fopen("c:\\test1.txt", "wt");
FILE *f2 = fopen("c:\\test2.txt", "wb");
FILE *f3 = fopen("c:/test3.txt", "w");
FILE *f4 = fopen("c:/test4.txt", "a");

if(f1) { fprintf(f1, "x\ny\nz\n"); fclose(f1); }
if(f2) { fprintf(f2, "x\ny\nz\n"); fclose(f2); }
if(f3) { fprintf(f3, "x\ny\nz\n"); fclose(f3); }
if(f4) { fprintf(f4, "x\ny\nz\n"); fclose(f4); }
```

Jakie długości plików (z dokładnością do pojedynczych bajtów) uzyskamy? Dlaczego?

Przykład 3. Zrelaksuj się i przeanalizuj następujący kod. Przy okazji podciągnij swoje umiejętności, przepisując go ręcznie i usuwając ewentualne błędy, a nie tylko ograniczaj się do Cytr+C, Ctrl+V.

```
     char p[] = { "Grzegorz Brzęczyszczykiewicz QWERTY123" };
     char q[] = {
          'G','r','z','e','g','o','r','z',' ','B','r','z',
          'ę','c','z','y','s','z','c','z','y','k','i','e',
          'w','i','c','z',' ','Q','W','E','R','T','Y','1',
          '2','3', '\0' };
//co oznacza symbol '\0' ? Po co go tu umieszczono?
     char * x;
     int i;

     printf("p = %s\nq = %s\n", p, q);
     printf("sizeof(p) == %d\n", sizeof(p));
     printf("sizeof(q) == %d\n", sizeof(q));
     printf("strlen(p) == %d\n", strlen(p));
     printf("strlen(q) == %d\n", strlen(q));
//Jakie wyniki się pojawią?


//Znajdź dokumentację funkcji strlen().
//Czy potrafiłbyś sam napisać jej odpowiednik?
//Jaką wartość ma 'a' - 'A' ? Dlaczego?
//Czy znasz już funkcję putch()? Czym ją można zastąpić?
     for(i=0; i<strlen(p); i++)   {
          if(p[i]>='a'  &&  p[i]<='z') //co oznacza ten warunek?
               putch(p[i] - ('a'-'A'));
          else
               putch(p[i]);
     }
     printf("\n\n");



     x = p;
     while(*x != '\0')
```

```
    {
        if(*x>='a'  &&  *x<='z')
            putch(*x - ('a'-'A'));
        else
            putch(*x);
        x++; //jak to działa?
    }
    printf("\n\n");


//Dlaczego pojawiło się *x zamiast dłuższej formy (*x != '\0') ?
    for(x=p; *x; x++)
        if(*x>='a'  &&  *x<='z')
            putch(*x - ('a'-'A'));
        else
            putch(*x);
    printf("\n\n");


//wersja dla ‘hardkorów’
//zwykli śmiertelnicy mogą sobie odpuścić kolejne 2 przykłady:
    for(x=p; *x; x++) *x = *x>='a' && *x<='z' ? *x - ('a'-'A') : *x;
    printf("%s\n\n", p);

    for(x=p; *x; x++) *x -= (*x>='a' && *x<='z') * ('a'-'A');
    printf("%s\n\n", p);
```