Języki i paradygmaty programowania 1 – studia stacjonarne 2025/26

Lab 4. Podstawowe biblioteki. Pętle. Operatory inkrementacji, dekrementacji, przypisania. Instrukcje goto, continue, break. Operacje na plikach. Projekt nr 1.

1. Pliki nagłówkowe dołączamy dyrektywą `#include`. Do najczęściej wykorzystywanych bibliotek możemy zaliczyć:
   i. **<stdio.h>** Definiuje podstawowe funkcje wejścia/wyjścia.
   ii. **<stdlib.h>** Zawiera najbardziej podstawowe funkcje związane z konwersją numeryczną, generacją liczb pseudolosowych, alokacją pamięci czy sterowaniem wykonywanych instrukcji.
   iii. **<math.h>** Zawiera podstawowe funkcje matematyczne.
   iv. **<stdbool.h>** Definiuje typ logiczny (bool).
   v. **<string.h>** Dostarcza funkcji dla ciągów znakowych.

2. W językach imperatywnych instrukcje wykonywane są sekwencyjnie - jedna po drugiej. W celu zmiany takiego porządku stosuje się instrukcje sterujące. Do takich instrukcji zaliczamy instrukcje warunkowe (lab 2) czy pętle. W języku C wyróżniamy trzy podstawowe pętle:
   i. **for** - pętla składająca się z trzech wyrażeń: warunku początkowego (zwykle inicjalizacji zmiennej będącej licznikiem wykonywanych iteracji), warunku stopu (określającego kiedy nasza pętla ma się zatrzymać), wyrażenie wykonywane po każdej iteracji (zwykle inkrementacja licznika);
   ii. **while** - pętla przyjmująca jedno wyrażenie wykonywane zawsze przed instrukcjami w ciele pętli. Jeśli wyrażenie zwraca 0 (logiczny fałsz), pętla jest przerywana;
   iii. **do-while** - pętla podobna do while z tą różnicą, że wpierw wykonywane są instrukcje podane w ciele pętli, a następnie wyrażenie warunkowe. Pętla ta gwarantuje, że instrukcje wykonają się co najmniej jeden raz.

   Zapoznaj się ze strukturą i sposobem użycia wyrażeń w dokumentacji MSDN (https://msdn.microsoft.com/en-us/library/015az3wz.aspx).

3. Język C dostarcza skróconych zapisów przypisania łączonego z daną operacją arytmetyczną. Poniższa tabela przedstawia zapisy skrócone i ich pełne odpowiedniki.

Lp.	Zapis skrócony	Zapis pełny
1.	x += 2;	x = x + 2;
2.	x -= 2;	x = x - 2;
3.	x *= 2;	x = x * 2;
4.	x /= 2;	x = x / 2;
5.	x++;	x = x + 1;
6.	++x;	x = x + 1;
7.	x--;	x = x - 1;
8.	--x;	x = x - 1;

   Zapisy 5. i 6. nazywamy inkrementacją, natomiast 7. i 8. dekrementacją. Przetestuj poniższy kod - czym różnią się zapisy 5. - 8.?

```
int a, b, c, d;
a = 0;
b = a++;
c = 0;
d = ++c;
printf("a = %i\n", a);
printf("b = %i\n", b);
printf("c = %i\n", c);
printf("d = %i\n", d);
```

4. Utwórz nowy projekt. Napisz program, który wypisuje określoną ilość razy tekst `"Hello world!"`. Przed wykonaniem pętli poproś o podanie ilości powtórzeń.

5. Napisz program, który oblicza silnię dla podanego n. Algorytm zaimplementuj tworząc nową funkcję z pętlą.

6. Do instrukcji sterujących zaliczamy również poniższe polecenia:
   i. **break** - pozwala na opuszczenie wykonywania pętli w dowolnym momencie;
   ii. **continue** - powoduje przejście do kolejnej iteracji pętli;
   iii. **goto** etykieta - powoduje przejście do miejsca oznaczonego etykietą.

7. Napisz nieskończoną pętlę wczytującą wprowadzane z klawiatury przez użytkownika znaki do momentu podania znaku 'e'. Do przerwania wykorzystaj instrukcję **break**.

8. Edytuj powyższy program tak, aby zliczał wystąpienia liter: a, b, c - każdej osobno oraz wszystkich pozostałych znaków razem - do momentu podania znaku 'e'. Skorzystaj z instrukcji **continue** i **break**.

9. Edytuj powyższy program tak, aby w przypadku wystąpienia litery 'c' była ona zarówno zliczana jako litera 'c' jak i wliczana do sumarycznej liczby wystąpień pozostałych znaków. Skorzystaj z instrukcji **goto**.

10. Funkcje operacji na plikach dostarcza biblioteka **stdio.h**. Wyróżniamy dwie metody dostępu do plików: niskopoziomową (binarną, np. open()) i wysokopoziomową (tekstową, np. fopen()). Aby otrzymać dostęp do pliku musimy zadeklarować pusty wskaźnik na obiekt typu **FILE**. Następnie funkcją **fopen**(*filename*, *mode*), gdzie *filename* jest ścieżką dostępu (względną lub bezwzględną) wraz z nazwą pliku, natomiast *mode* określa nam uprawnienia, otrzymujemy dostęp do naszego pliku. Wyróżniamy następujące podstawowe metody dostępu (*mode*) do pliku:
    i. **"r"** - otwiera na czytanie; plik musi istnieć;
    ii. **"w"** - otwiera na pisanie; jeśli plik istnieje to jego zawartość zostaje zniszczona;
    iii. **"a"** - otwiera na pisanie; jeśli plik istnieje to zawartość jest dopisywana do końca pliku; jeśli plik nie istnieje to zostaje utworzony;
    iv. **"r+"** - otwiera na czytanie i pisanie; plik musi istnieć;
    v. **"w+"** - otwiera na czytanie i pisanie; jeśli plik istnieje to jego zawartość zostaje zniszczona;

vi. **"a+"** - otwiera na czytanie i pisanie; jeśli plik istnieje to zawartość jest dopisywana do końca pliku; jeśli plik nie istnieje to zostaje utworzony.

Do parametru *mode* można również dołączyć opcje określające, czy mamy do czynienia z plikiem zwykłym (t) czy binarnym (b). Domyślnie wybierany jest tryb dla plików zwykłych (tekstowych).

Więcej informacji nt. metod dostępu na wykładzie i w dokumentacji MSDN (https://msdn.microsoft.com/en-us/library/yeby3zcb.aspx).

11. Operacje odczytu i zapisu do pliku są bardzo podobne do znanych już nam operacji **printf** i **scanf**, z tą różnicą, że jako pierwszy argument podajemy wskaźnik do pliku. Do odczytu możemy wykorzystać funkcję **fscanf**, a do zapisu **fprintf**. Po każdej zakończonej pracy z plikiem należy go zamknąć poleceniem: **fclose**. Opis wymienionych i dodatkowych funkcji - na wykładzie i w dokumentacji MSDN.

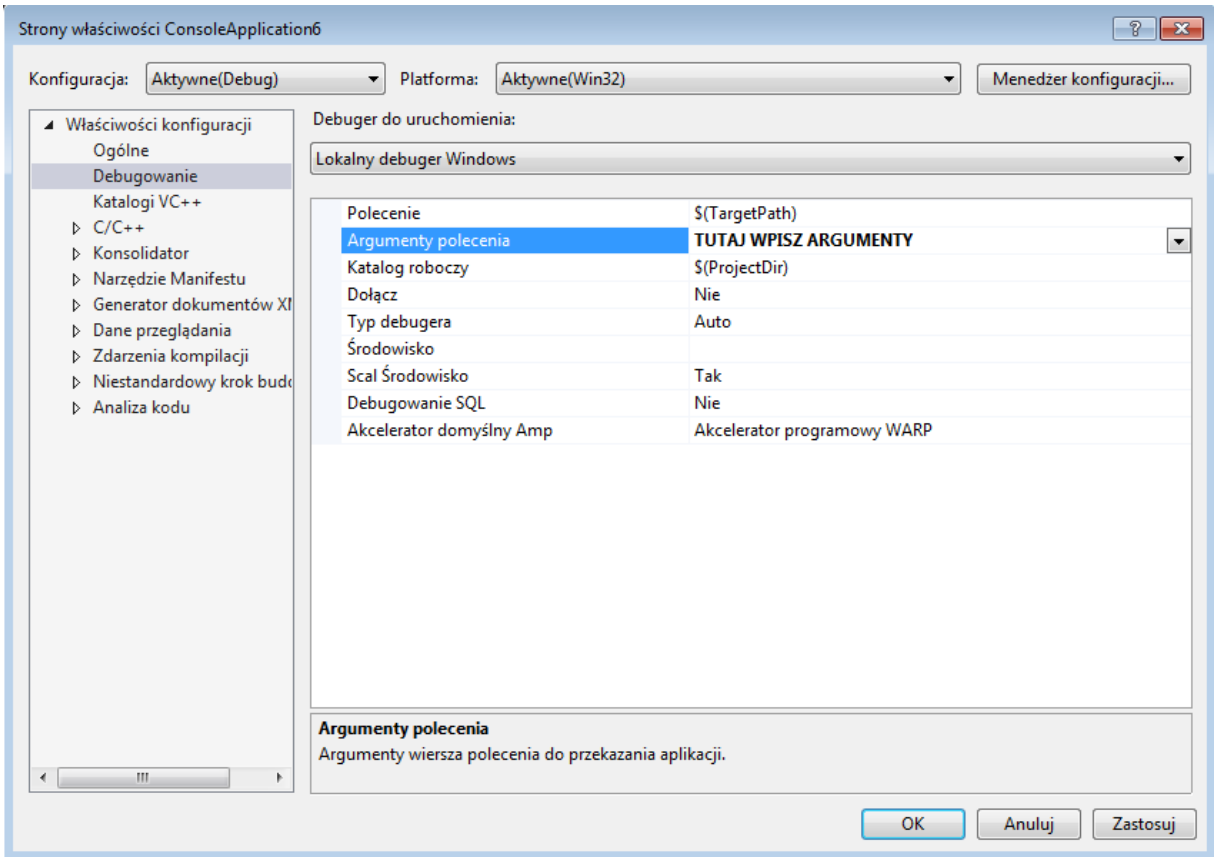
12. Utwórz nowy projekt. Przekopiuj i przeanalizuj poniższy kod:

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
	FILE *fp = NULL;
	fp = fopen("lab4.txt", "w");
	if (fp == NULL) {
		printf("Nie mogę otworzyć pliku lab4.txt do zapisu!\n");
		getchar();
		exit(1);
	}
	char tekst[] = "Hello world w pliku :)";
	fprintf(fp, "%s", tekst);
	fclose(fp);
	fp = NULL;
	getchar();
	return 0;
}
```

13. Utwórz plik tekstowy z wiadomością: "`I <3 programming`". Napisz program, który będzie odczytywał wiadomość z pliku po jednym znaku i wyświetlał ją na ekranie. Skorzystaj z pętli **while**. Podpowiedź: sprawdź jakie wartości zwraca funkcja **fscanf** w przypadku, gdy dojdzie do końca pliku.

14. Utwórz nowy projekt. W ustawieniach projektu (Projekt/nazwa_projektu Właściwości) wpisz argumenty przekazywane do programu wraz z jego wywołaniem w trybie Debug (tak jak na poniższym zrzucie ekranu).



Przeanalizuj wywołanie poniższego kodu:

```
#include<stdio.h>
#include<stdlib.h>
#pragma warning(disable:4996)
int main(int argc, char *argv[]){
	int i;
	printf("Liczba argumentow: %i\n", argc);
	for (i = 0; i < argc; i++)
		printf("%s\n", argv[i]);
	getchar();
	return 0;
}
```

15. Tablicowanie funkcji w przedziale [a,b] z krokiem dx.

Policzyć wartości funkcji

$$f(x) = \frac{1}{x^2 + \sqrt{1 + 2x + 3x^2}}$$

Dla x zmieniającego od a do b z krokiem dx. Wartości funkcji $f(x)$ liczone w funkcji

```
double ff(double x){}
```

Wyniki $(x, f(x))$ wyświetl na ekranie i zapisz do pliku.

```
fprintf(fw, "\t%10.2lf\t\t%15.4lf\n", x, y);
printf("\t%10.2lf\t\t%15.4lf\n", x, y);
```

16. Tablicowanie funkcji w przedziale [a,b] z krokiem dx. Do wyznaczenia wartości funkcji w danym punkcie wykorzystaj rozwinięcie funkcji w szereg Taylora. Dla sprawdzenia poprawności rozwiązania wypisz wartość ścisłą funkcji w danym punkcie.
Policzyć dla dowolnego argumentu x wartość funkcji $y = e^x$ korzystając z jej rozwinięcia w szereg Taylora

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, \qquad -\infty < x < \infty$$

Obliczenie sumy wykonaj z zadaną dokładnością ε.

Sumę nieskończoną zastępujemy sumą skończoną:

$$e^x \cong \sum_{i=0}^{n} \frac{x^i}{i!}$$

W której n przyjmujemy tak, by

$$\left|\frac{x^n}{n!}\right| < \delta$$

dla dowolnie małego δ. Natomiast δ dobierzemy tak, by

$$|f_d - f_p| < \varepsilon$$

gdzie, f_d – dokładna wartość funkcji e^x, f_p – przybliżona wartość funkcji e^x, ε – żądana dokładność. Zależność między ε i δ można wyznaczyć ze znanej postaci reszty szeregu Taylora.

```c
#pragma warning (disable: 4996)
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
FILE *fw;
double sz(double x, double delta);
int main()
{
    /* Tablicowanie funkcji (szereg potegowy) */
    double a, b, dx, x, y, delta, z;
    printf("Podaj przedzial [a,b], krok dx oraz dokladnosc: ");
    if (scanf("%lf %lf %lf %lf", &a, &b, &dx, &delta) != 4)
    {
           printf("Blad danych\n");
           system("pause");
           exit(1);
    }
    if (!(fw = fopen("wyniki.txt", "w")))
    {
           printf("Blad otwarcia zbioru\n");
           exit(2);
    }

    for (x = a; x <= b + 0.5*dx; x += dx)
    {
           y = sz(x, delta);
           z = exp(x);
           fprintf(fw, "\t%10.2lf\t\t%15.4le\t\t%15.4le\n", x, y, z);
    }

    fclose(fw);
    exit(0);
    system("pause");
}
```

```
double sz(double x, double delta)
{
    /* Obliczanie sumy szeregu potegowego */
    double d = 1, s = 1;
    int n = 1;
    do
    {
            d *= x / n;
            s += d;
            n++;
    } while (fabs(d) >= delta);
    return s;
}
```

17. **PROJEKT NR1.**

Policz wartości funkcji $y = f(x)$ we wszystkich punktach podziału na n części przedziału $[a, b]$. Funkcja f dana jest w postaci rozwinięcia w szereg potęgowy i w postaci wzoru analitycznego. Obliczanie sumy szeregu wykonaj z dokładnością ε. Algorytm obliczania sumy szeregu zapisz w oddzielnej funkcji.

Uzupełnij funkcję obliczającą sumę szeregu tak, by sumowanych było co najwyżej M wyrazów szeregu. Oznacza to, że przerwanie sumowania może nastąpić również wtedy, gdy nie została osiągnięta żądana dokładność. Informacja o tym, czy została osiągnięta dokładność czy też nie winna być znana w funkcji **main().**

Uzupełnij funkcję obliczającą sumę szeregu tak, by w funkcji **main()** znana była dodatkowo liczba sumowanych wyrazów szeregu.

Wyniki wyświetl na ekranie i zapisz do pliku. Wyniki przedstaw w następującym układzie:

x	f_szereg(x)	f_ścisłe(x)	liczba wyrazów szeregu	warunek stopu

18. Przeanalizuj w trybie pracy krokowej następujące przykłady do wykładu 4:
    http://torus.uck.pk.edu.pl/~fialko.sergiy/text/CC/przykl/WW4.pdf
    http://torus.uck.pk.edu.pl/~fialko.sergiy/text/CC/przykl/WW4_1.pdf
    http://torus.uck.pk.edu.pl/~fialko.sergiy/text/CC/przykl/WW4_2.zip
    Wszystkie przykłady dostępne pod adresem:
    http://torus.uck.pk.edu.pl/~fialko.sergiy/text/CC/przykl/

**Treści oznaczone kursywą pochodzą z różnych źródeł internetowych.*