Języki i paradygmaty programowania 1 – studia stacjonarne 2024/25

Lab 7. Program zapisany w kilku plikach. Pliki nagłówkowe (*.h). Słowa kluczowe static, const. Klasy pamięci auto, static, extern. Tablice liczbowe. Funkcje memcpy, memmove, memset. Wyrażenie warunkowe.

1. *Projekty składają się zazwyczaj z wielu, różnych plików. Zawartość każdego pliku programista wyznacza zgodnie z jego przeznaczeniem. Bezpośredni dostęp do danych jest możliwy wyłącznie z tego pliku, w którym dane te są zdefiniowane. Dostęp do danych z innych plików nie jest wspierany w języku C. Aby tworzone programy były bezpieczne należy:*
    a. *podstawowe dane w pliku definiować, jako **zmienne globalne** z modyfikatorem **static**, który nie pozwala na udostępnianie tych zmiennych w innych plikach;*
    b. *prototypy funkcji interfejsowych umieścić w odpowiednim **headerze** (**bez** modyfikatora **static**);*
    c. *prototypy funkcji wewnętrznych umieścić w pliku, jako obiekty **globalne z** modyfikatorem **static**;*
    d. *dane globalne, które mogą być „widoczne" w innych plikach:*
        i. *udostępniać poprzez funkcje*
        ii. *Definiować **bez** modyfikatora **static***
        iii. *Samo udostępnienie w innym pliku wprowadzić przy pomocy specyfikatora **extern***
        iv. *Nie poleca się umieszczania tych danych w pliku nagłówkowym **header**.*
    e. *Dane pomocnicze (indeksy tablic, zmienne tymczasowe itd.) używać jako dane lokalne (**auto**).*
2. ***Headery**, czyli pliki nagłówkowe, są używane do deklaracji typów danych (struktur, unii, typedef), makr, dyrektyw kompilatora (pragma), prototypów funkcji itd., które mają być widoczne w różnych plikach projektu.*
3. *Każdy **header** musi zawierać tak zwane straże, które w projektach złożonych chronią przed duplikacją definicji umieszczanych w headerze.*
4. *Klasy pamięci dzielą się na:*
    a. ***auto** – wszystkie zmienne i tablice zdefiniowane są w dowolnym bloku. Rezerwacja miejsca w pamięci następuje w momencie wejścia do bloku programu. Pamięć ta zostaje zwolniona w chwili wyjścia z bloku, w którym dana zmienna została zdefiniowana – następuje utrata danych. Widoczność (zasięg deklaracji) zmiennej w klasie auto to tzw. blok programu. Czas życia zmiennej – od momentu definicji do momentu wyjścia z bloku. Zmienne domyślnie nie są inicjowane - jeśli użytkownik nie przypisze im wartości, zawierają przypadkowe „śmieci".*
    b. ***static** – (zmienne i tablice zdefiniowane lokalnie – wewnątrz bloku programu). Rezerwacja pamięci następuje na początku działania programu. Obszar widoczności zmiennych klasy **static** – od miejsca definicji w bloku, w którym zostały zdefiniowane, do końca bloku. Czas życia zmiennej – czas trwania całego programu. Oznacza to, że wartości zmiennych tej klasy pozostają chronione nawet po wyjściu z bloku i są aktualne przy kolejnym wejściu w blok. Zmienne domyślnie inicjalizowane są wartością: zero.*

`void fun()`
`{`
`  int a = 0;`
`  a++;`
`}`
`pierwsze wejście:  a = 0`
`przed wyjściem:  a = 1`
`drugie wejście:  a = 0`
`przed wyjściem:  a = 1`
	`void fun()`
	`{`
	`  static int a;`
	`  a++;`
	`}`
	`pierwsze wejście:  a = 0`
	`wyjście:  a = 1`
	`drugie wejście:  a = 1`
	`wyjście:  a = 2`

c. ***static*** *– (definicja globalna – w pliku poza blokami). Rezerwacja pamięci dokonywana jest na początku działania programu. Obszar widoczności zmiennych – od miejsca definicji do końca pliku. Czas życia zmiennej – czas trwania całego programu. Zmienne domyślnie inicjalizowane są wartością: zero.* ***Są dostępne tylko w podanym pliku****.*

d. ***Zmienne zewnętrzne (globalne)*** *(bez specyfikatora static). Rezerwacja pamięci dokonywana jest na początku działania programu. Obszar widoczności zmiennych globalnych– od miejsca definicji do końca pliku. Czas życia zmiennej – czas trwania całego programu. Zmienne domyślnie inicjalizowane są wartością: zero.* ***Mogą być udostępnianie w innych plikach,*** *jeśli zostaną w nich zadeklarowane, tzn. deklaracja typu zostanie poprzedzona słowem kluczowym* ***extern****.*

e. *Klasa pamięci* ***register*** *oznacza, że dane zostaną umieszczone w rejestrze procesora, kiedy będzie to możliwe.*

f. *Specyfikator* ***static*** *używany przy deklaracji funkcji, ogranicza widoczność tej funkcji obszarem pliku. Nie wolno umieszczać deklaracji funkcji statycznych w plikach nagłówkowych.*

5. Do czego służą słowa kluczowe **static** i **const**? Czym się różnią?
6. Utwórz nowy projekt składający się z dwóch poniższych plików: plik_1.cpp, plik_2.cpp. Na podstawie informacji zawartych w pkt. 4 określ typ, klasę pamięci, obszar działania każdego występującego w programie identyfikatora. Przeanalizuj odwołania do różnych funkcji i znajdź te fragmenty programu, które na takie odwołania pozwalają. Sprawdź jakie teksty pojawią się na monitorze podczas wykonania programu.

```
//************************PLIK_1*****************************
#include <stdio.h>
double fun1();
static int fun2();
extern int fun3();
static int c = 5;
double a, b = 10;
char *xx[] = { "mama","tato","stryjek",(char *)0};
double aa[] = { 1,2,3,4,5,6 };

int main()
{
	double x, y = 5;
	int i, j, k;
	static double aa[] = { 11,12,13,14,15,16 };
	printf("%lf %lf \n", aa[0], aa[1]);
	j = fun2();
	k = fun3();
	printf("j = %d, k = %d\n", j, k);
}
```

```
double fun1(int x, int y)
{
	static char *xx[] = { "pies","kot","mysz",(char *)0 };
	int i = 0;
	i++;
}

char *xxxx[] = { "zima","wiosna","lato",(char *)0 };

static int fun2()
{
	static int k = 0;
	puts("ppp fun2");
	k++;
	return(k);
}

//************************PLIK_2*****************************
#include <stdio.h>
extern double fun1();
static double fun2();
int fun3();
static char *c[] = { "slon","lew","pantera",(char *)0 };
extern char *xx[];

static double fun2()
{
	static char *zz[] = { "krzeslo","szafa","tapczan",(char *)0 };
	puts("qqq fun2");
	return ((double) 2.0);
}
int fun3()
{
	double ff;
	puts("qqq fun3");
	ff = fun2();
	return (5);
}
```

7. Wybierz jeden spośród Twoich programów (np. projekt nr 1 lub nr 2) i podziel go na kilka plików, w taki sposób aby nie dzielić funkcjonalności, które tego nie wymagają.

8. Funkcje **memcpy, memmove, memset** znajdują się w bibliotece **#include <memory.h>.**

```
void *memcpy(void *dest, const void *src, size_t count);
```
funkcja kopiuje count bajtów z src do dest. Jako wynik zwraca wskaźnik do dest. Jeśli obszar pamięci src i dest pokrywa się działanie funkcji jest nieprzewidywalne.

```
void *memmove(void *dest, const void *src, size_t count);
```
funkcja kopiuje count bajtów z src do dest. Jeśli jakiś obszar pamięci src i dest pokrywa się funkcji gwarantuje, że bajty źródłowe w pokrywającym się regionie zostaną skopiowane zanim zostaną przepisane.

```
void *memset(void *dest, int c, size_t count);
```
Przypisuje symbol c pierwszym count symbolom tablicy dest. Zwraca wskaźnik do dest.

9. Utwórz nowy projekt, a następnie skopiuj i przeanalizuj poniższe przykłady użycia funkcji memcpy, memmove, memset.

a) Przykład 1.

```
#include <memory.h>
#include <string.h>
#include <stdio.h>
char str1[7] = "aabbcc";
int main(void)
{
	printf("The string: %s\n", str1);
	//Pokrywający się region! Kopiowanie może nie być poprawne
	memcpy(str1 + 2, str1, 4);
	printf("New string: %s\n", str1);
	strcpy_s(str1, sizeof(str1), "aabbcc"); // reset string
	printf("The string: %s\n", str1);
	//Pokrywający się region! Kopiowanie poprawne
	memmove(str1 + 2, str1, 4);
	printf("New string: %s\n", str1);
}
```

b) Przykład 2.

```
#include <memory.h>
#include <stdio.h>
int main(void)
{
	char buffer[] = "This is a test of the memset function";
	printf("Before: %s\n", buffer);
	memset(buffer, '*', 4);
	printf("After: %s\n", buffer);
}
```

Wynik działania:
Before: This is a test of the memset function
After : **** is a test of the memset function

c) Przykład 3.

```
#include <memory.h>
#include <stdio.h>
int main(void)
{
   char str[256];
   double aa[200];
   double cc[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
   double dd[5] = { 20, 30, 40, 50, 60 };
   memset((void *)str, 0, 256 * sizeof(char)); //OK
   memset((void *)str, ' ', 256 * sizeof(char)); //!OK-pisanie po pamięci
   memset((void *)str, ' ', 255 * sizeof(char)); //OK
   memset((void *)aa, 0, 200 * sizeof(double)); //OK
   memcpy((void *)&cc[3], (const void *)&dd[2], 2 * sizeof(double));
}
```

Wynik działania: cc: 1 2 3 40 50 6 7 8 9 10

10. Napisz program, który utworzy dynamicznie (za pomocą funkcji **malloc**) dwie tablice t1 i t2 liczb całkowitych o rozmiarze n wczytywanym z klawiatury. Jeżeli operacja przydziału pamięci dla t1 i t2 zakończyła się pomyślnie (wskaźniki t1 i t2 są różne od NULL), to zainicjuj tablicę t1 losowymi liczbami. Napisz funkcję, która korzystając z funkcji: **srand**, **time**, **rand** oraz operatora dzielenia modulo uzupełni przekazaną do niej tablicę typu int o losowe liczby z zakresu od 0 do 99. Tablica niech będzie przekazywana przez wskaźnik. Znaleźć minimalny i maksymalny element w tablicy oraz jego położenie. Wyprowadzić na ekran zawartość tablicy t1, a także znalezione wartości i położenie (indeksy) minimum i maksimum. Przekopiować tablicę t1 do t2 wykorzystując standardową funkcję memmove lub memcpy. Wyprowadzić zawartość tablicy t2 na ekran. Przed zakończeniem programu zwolnić pamięć przydzieloną t1 i t2.

11. Przykład. Napisz program, który dla danych dwóch wektorów $\vec{x}, \vec{y}$ policzy:
    i. Sumę tych wektorów: $z_i = x_i + y_i$;
    ii. Iloczyn skalarny wektorów: $s = \sum_{i=0}^{n-1} x_i \cdot y_i$;
    iii. Maksymalną współrzędną wektorów $\vec{x}, \vec{y}$

```
//==============================PLIK_1==================================
#pragma warning (disable:4996)
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define LL 200
extern void error(int, char *);
void argumenty(int, char **);

int main(int argc, char *argv[])
{
	double x[LL], y[LL], z[LL], s, mx, my;
	FILE *fw, *fd;
	int n, k;
	argumenty(argc, argv);
	if (!(fd = fopen(argv[1], "r"))) error(2, "dane");
	if (!(fw = fopen(argv[2], "w"))) error(2, "wyniki");
	fscanf(fd, "%d", &n);
	for (k = 0; k < n; k++)
		fscanf(fd, "%lf", &x[k]);
	for (k = 0; k < n; k++)
		fscanf(fd, "%lf", &y[k]);
	s = 0;
	mx = x[0];
	my = y[0];
	for (k = 0; k < n; k++)
	{
		z[k] = x[k] + y[k];
		mx = x[k] > mx ? x[k] : mx;
		my = y[k] > my ? y[k] : my;
		s += x[k] * y[k];
	}
	for (k = 0; k < n; k++)
	{
		fprintf(fw, "%lf ", z[k]);
		if (!((k + 1) % 5)) fprintf(fw, "\n");
	}
	fprintf(fw, "\nilocz.skal=%lf mx=%lf my=%lf\n", s, mx, my);
}
void argumenty(int argc, char *argv[])
{
	int len;
	char *usage;
	if (argc != 3)
	{
		len = strlen(argv[0]) + 19;
		if (!(usage = (char*)malloc((unsigned)len * sizeof(char))))
error(3, "tablica usage");
		strcpy(usage, argv[0]);
		strcat(usage, " file_in file_out");
		error(4, usage);
	}
}
```

```
/*************** plik util_1.cpp ****************************/
#include<stdio.h>
#include<stdlib.h>
#define MAX_ERR 5
static char *p[] = { "",
                                    " zle dane",
                                    " otwarcie pliku",
                                    " brak pamieci",
                                    " Usage : ",
                                    " nieznany ",
};

void error(int nr, char *str)
{
        int k;
        k = nr >= MAX_ERR ? MAX_ERR : nr;
        fprintf(stderr, "Blad(%d) -  %s %s\n", nr, p[k], str);
        exit(nr);
}
```

12. **Wyrażenie warunkowe** przyjmuje postać:
    `wyrażenie1 ? wyrażenie2 : wyrażenie3`
    wartością wyrażenia warunkowego jest:
    - Wartość wyrażenia2, o ile wyrażenie1 jest różne od zera,
    - Wartość wyrażenia3, o ile wyrażenie1 jest równe zero.

    Operator pytajnik, dwukropek (? :) jest prawostronnie łączny i ma priorytet wyższy jedynie od operatorów przypisania i operatora przecinkowego.

13. Zmodyfikuj program z przykładu w pkt.11 tak aby:
    - Zamiast tablic zdefiniować zmienne typu `double *`;
    - Zarezerwować dokładnie n miejsc na zmienne typu `double` (alokacja pamięci), np.:
      `x = (double*)malloc((unsigned)n * sizeof(double))`

14. Algorytm z pkt.11 podziel na kilka funkcji:
    a) Alokacja pamięci dla tablicy: `double *DajWekt(int n)`
    b) Czytanie elementów tablicy z pliku: `void CzytWekt(FILE *fd, double *we, int n)`
    c) Pisanie elementów tablicy do pliku: `void PiszWekt(FILE *fw, double *we, int n)`
    d) Obliczanie sumy dwóch wektorów:
    `void DodWekt(double *w1, double *w2, double *w3, int n)`
    e) Obliczanie iloczynu skalarnego: `double IloczynSkal(double *w1,double *w2,int n)`
    f) Obliczanie maksymalnej współrzędnej wektora: `double MaxElem(double *w, int n)`

    Funkcje realizujące zadania a), b), c) zapisz w pliku util_2.cpp.
    Funkcje realizujące zadania d), e), f) zapisz w pliku util_3.cpp.

15. Przeanalizuj w trybie pracy krokowej przykłady do wykładu 7:
    Wszystkie przykłady dostępne pod adresem:
    http://torus.uck.pk.edu.pl/~fialko.sergiy/text/CC/przykl/

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