Języki i paradygmaty programowania 1 – studia stacjonarne 2019/20

Lab 7. Kolokwium nr 1. Program zapisany w kilku plikach. Pliki nagłówkowe (*.h). Słowa kluczowe static, const. Klasy pamięci (auto), static, extern. Tablice liczbowe.

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
```

```
void fun()
{
       static int a;
       a++;
}
pierwsze wejście:    a = 0
```

przed wyjściem:	a = 1	wyjście:	a = 1
drugie wejście:	a = 0	drugie wejście:	a = 1
przed wyjściem:	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. Napisz procedurę, 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 10. Tablica niech będzie przekazywana przez referencję.
9. Do poprzedniego programu dopisz procedurę, która będzie wyświetlać zawartość tablicy oraz informację ile razy funkcja była wywoływana (skorzystaj ze zmiennej statycznej). Tablica niech będzie przekazywana przez wskaźnik.
10. Operatorem **sizeof** sprawdź rozmiar tablicy w trzech miejscach (po jej deklaracji i w dwóch procedurach z powyższych zadań). Wytłumacz otrzymane wyniki.
11. Zadeklaruj i uzupełnij liczbami tablicę 5 elementową. Wypisz wartość 3 elementu BEZ KORZYSTANIA Z INDEKSÓW (złe rozwiązanie: `printf("%d\n", tablica[2]);`).

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

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