Wskaźniki do funkcji

Trochę teorii

Definicja wskaźnik do funkcji przyjmującej dwie wartości typu integer i zwracającej wartość typu integrer:

```
int(*f)(int, int);
```

Przypisanie funkcji do wskaźnika:

```
int func(int a, int b)
{
	return a + b;
}
f = &func;
```

Zadanie 1.

Napisz program, który będzie pozwalał na wykonanie operacji matematycznych (dodawania, odejmowanie, mnożenie, dzielenie). W programie ma być napisana funkcja, która będzie wywoływała funkcję odpowiedzialną za wykonanie konkretnej operacji matematycznej. Jako parametry ma przyjąć wskaźnik do funkcji oraz dwie liczby. Wynik działania ma zostać wypisany na konsoli. Do programu ma zostać dodane menu, które pozwoli użytkowniki wybrać jaką operacje chce wykonać i podać liczby, dla których ma zostać ona wykonana.

Zadanie 2.

Napisz prosty kalkulator, który pobiera od użytkownika dwie liczby (dowolny typ) i rodzaj operacji (np. dodawanie), określony zaproponowanym typem enum. Napisz funkcję, która na podstawie typu operacji zwraca wskaźnik do funkcji wykonującej tę operację. Następnie wywołaj tę funkcję (przez jej wskaźnik), a wynik wyświetl.

Zadanie 3.

Napisz program, który będzie pozwalał na wykonanie następujących działań: znajdowanie minimum, maksimum, mediany, dominanty, obliczanie średniej, odchylenia standardowego. W programie ma być napisana funkcja, która będzie wywoływała funkcję odpowiedzialną za wykonanie konkretnej operacji. Jako parametry ma przyjąć wskaźnik do funkcji oraz tablicę. Wynik działania ma zostać wypisany na konsoli. Do programu ma zostać dodane menu, które pozwoli użytkowniki wybrać jaką operacje chce wykonać i podać liczby, dla których ma zostać ona wykonana.

Zadanie 4.

Napisz funkcję, która przetwarza znaki w tekście za pomocą funkcji o podanym wskaźniku. Wskaźnik ma być zbudowany na prototypie char func_name(char ch). Wykorzystaj napisaną funkcję przetwarzającą do zamiany znaków małych na duże, dużych na małe oraz spacji na myślniki w dowolnych, wybranych przez siebie tekstach. Wyniki dla wszystkich operacji wyświetlaj oddzielnie.

Zadanie 5.

Napisać funkcję wyznaczającą miejsce zerowe dowolnej funkcji *f(x)* w przedziale [a,b] metodą bisekcji. Funkcja powinna przyjmować wskaźnik do funkcji, której miejsce zerowe należy wyznaczyć. Zaprezentować działanie funkcji w programie. Wartości **a** i **b** oraz dokładność przybliżenia miejsca zerowego epsilon przekazać jako parametry funkcji main(). Zapewnić kontrolę błędów.

Zadanie 6.

Zdefiniować w programie trzy funkcje typu void bez parametrów, które za pomocą dowolnego symbolu narysują na ekranie prostokąt, trójkąt oraz romb (o dowolnych rozmiarach). Wewnątrz funkcji main() zadeklarować trzyelementową tablicę wskaźników do funkcji i zainicjować ją funkcjami wymienionymi powyżej. Zaimplementować menu tekstowe, które po na ciśnięciu odpowiedniego przycisku uruchomi przypisaną do niego akcję. Menu powinno posiadać na stępujące opcje wyboru przypisane do podanych przycisków:

Przycisk	Akcja
h	Wyświetla dostępne opcje (czyli osobna funkcja wyświetlająca to menu).
1	Wywołuje funkcję rysującą prostokąt, jako pierwszy element zdefiniowanej tablicy wskaźników do funkcji (zapis wskaźnikowy).
2	Wywołuje funkcję rysującą trójkąt, jako drugi element zdefiniowanej tablicy wskaźników do funkcji (zapis wskaźnikowy).
3	Wywołuje funkcję rysującą romb, jako trzeci element zdefiniowanej tablicy wskaźników do funkcji (zapis wskaźnikowy).
ESC	Kończy działanie programu.

Dynamiczna alokacja pamięci

Zadanie 1.

Napisz program, który alokuje tablicę 100 liczb typu float. Wypełnij ją wartościami od 0 do 99 i wyświetl. Pamiętaj o zwolnieniu pamięci przed zakończeniem programu.

Zadanie 2.

Alokuj bloki pamięci po 1kB w nieskończonej pętli. Sprawdź, jak zachowa się system operacyjny (skorzystaj z menedżera zadań).

Zadanie 3.

Zaalokuj pamięć dla liczby typu double. Do zaalokowanej pamięci zapisz liczbe PI. Wyświetla wartość tej pamięci (zaalokowanej zmiennej) oraz jej adres.

Zadanie 4.

Napisz funkcję alokującą tablice typu float o podanej liczbie elementów. Wykorzystaj prototyp:

```
float* alokuj(int N);
```

Zadanie 5.

Dana jest tablica: `int tab[] a = { 1,2,3,4,5,6,7,8,9,10 };`

Napisz funkcję wyświetlającą tablice typu int o następującym prototypie:

```
void wyswietl(const int* ptr, int N);
```

Następnie napisz funkcje, która wykonuje kopię tablicy w parametrze. Prototyp funkcji kopiującej:

```
int* kopia(const int* ptr, int N);
```

Wykonaj kopię tablicy a. Wyświetl zawartość tej tablicy oraz jej kopii.

Struktury

Zadanie 1 - PUNKT.

Zaprojektuj strukturę `point_t`, która będzie reprezentowała punkt w przestrzeni dwuwymiarowej. Przygotuj do niej funkcje pozwalające na:

- `void przesun_w_poziomie(point_t* p, int dx);`
- `void przesun_w_pionie(point_t* p, int dy);`
- `void oblicz_odleglosc(const point_t* p1, const point_t* p2);`

W programie zadeklaruj 10 elementową tablicę struktur `point_t`, każdemu punktowi przypisz losowe współrzędne. Napisz funkcje zapisujące i wczytujące tablice do i z pliku.
Napisz funkcje, która posortuje tablicę punktów według współrzędnych x i y.
Napisz funkcję, która z pośród wszystkich punktów w tablicy wybierze ten, który jest najbardziej oddalony od pozostałych i go zwróci.

Zadanie 2 - STUDENCI.

Utwórz strukturę student_t, zawierającą trzy pola: imie (100 znaków), nazwisko (100 znaków), album (10 znaków). Napisz funkcję, która zwraca wskaźnik na strukturę wypełnioną danymi, podanymi z klawiatury przez użytkownika. Przykładowy prototyp:

```
student_t* wypelnij(void);
```

Dodaj funkcję wyświetlającą oraz zwalniającą pamięć.

Zadanie 3.

Zmodyfikuj poprzednie zadanie tak, aby pola nie były tablicami znaków a wskaźnikami. Ponownie napisz funkcje

```
student_t* wypelnij(void);
```

oraz wyświetl i zwolnij. Pamiętaj, że zwolnić należy nie tylko pamięć struktury student_t, ale też i tekstów w poszczególnych polach.

Zadanie 4 – LICZBY ZESPOLONE.

Zapoznaj się z podstawowymi operacjami na liczbach zespolonych a następnie zaproponuj strukturę zesp_t, przechowującą liczbę zespoloną. Napisz następujące funkcje:

```
zesp_t* lz(double rzeczywista, double urojona);
void ustaw(zesp_t* liczba, double rzeczywista, double urojona);
void wyswietl(zesp_t* liczba);
void dodaj(zesp_t* a, zesp_t* b, zesp_t* wynik);
void odejmij(zesp_t* a, zesp_t* b, zesp_t* wynik);
void pomnoz(zesp_t* a, zesp_t* b, zesp_t* wynik);
double modul(zesp_t* liczba);
double argument(zesp_t* liczba);
```

Każda z funkcji musi sprawdzać, czy przekazywany parametr wskaźnikowy posiada wartość NULL. Zaproponuj operację matematyczną z wykorzystaniem w/w funkcji, a następnie wyświetl oraz sprawdź uzyskane wyniki.

Zadanie 5 - ODCINKI.

Zaprojektuj strukturę, która będzie reprezentowała odcinek w przestrzeni dwuwymiarowej (struktura musi przechowywać współrzędne początku i końca odcinka). Przygotuj do niej funkcje pozwalające na (wszystkie funkcje powinny przyjmować wskaźnik/wskaźniki do struktur, jeżeli to konieczne powinny również zwracać wskaźnik do struktury):

- utworzenie nowego odcinka,
- obliczenie długości odcinka,
- porównanie dwóch odcinków.

Napisz program, który będzie tworzyl tablicę odcinków w sposób dynamiczny. Użytkownik ma wprowadzić z klawiatury liczbę odcinków, jakie mają zostać utworzone. Każdemu odcinkowi przypisz losowe współrzędne. Napisz funkcje zapisujące i wczytujące tablice do i z pliku.
Napisz funkcje, która posortuje odcinki według ich długości.
Napisz funkcję, która zwróci punkt przecięcia się dwóch odcinków.

Zadanie 6 – ODCINKI 2.

Zmodyfikuj strukturę z poprzeniego zadania w taki sposób, aby oprócz początku i końca odcinka przechowywała również współrzędne wszystkich punktów wchodzących w sklad odcinka. Punkty mają być przechowywane w strukturze jako wskaźnik do tablicy. Napisz funkcję, która będzie wyznaczała wszystkie punkty należące do zadanego odcinka oraz wypełniała odpowiednie pole w strukturze. Użyj dynamicznej alokacji pamięci.

Zadanie 7 - PROSTOKĄT.

Zaprojektuj strukturę, która będzie reprezentowała prostokąt w przestrzeni dwuwymiarowej. Przygotuj do niej funkcje pozwalające na:

- utworzenie nowego prostokąta,
- obliczenie obwodu prostokąta,
- obliczenie pola prostokąta,
- przesuwający odcinek o 1 w lewo, prawo, górę i dół,
- skalującą prostokąt (w paramterach ma przyjąć wskaźnik do struktury prostokąt oraz skalę (float)),
- porównanie dwóch prostokątów.

Napisz program, który będzie tworzyl tablicę prostokątów w sposób dynamiczny. Użytkownik ma wprowadzić z klawiatury liczbę prostokątów, jakie mają zostać utworzone. Każdemu prostokątowi przypisz losowe współrzędne. Napisz funkcje zapisujące i wczytujące tablice do i z pliku.
Napisz funkcje, która posortuje prostokąty według ich obwodu i pola.
Napisz funkcję, która sprawdzi czy dwa prostokąty się przecinają.
Utwórz nową tablicę, w której będą przechowywane pary prostokątów przecinających się.
Napisz funkcję, która sprawdzi czy prostokąt i odcinek przecinają się.
Korzystając z danych wygenerowanych w ramach zadania 5 sprawdź, które z par odcinek – prostokąt się przecinają. Utwórz odpowiednią strukturę, która będzie przechowywała wskaźniki do przecinającego się odcinka i prostokąta. Zapisz te pary do pliku.
Napisz funkcję, która sprawdzi czy zadany punkt leży w środku prostokąta.
Korzystając z danych wygenerowanych w ramach zadania 1 sprawdź, który z punktów należy do prostokąta. Utwórz odpowiednią strukturę, która będzie przechowywała wskaźniki do punktu i prostokąta. Zapisz te pary do pliku.
Napisz program, który będzie testował wszystkie napisane przez Ciebie funkcje.

Zadanie 8 - HIGHSCORE.

Zaprojektuj strukturę do przechowywania tablicy wyników (highscore). Struktura przechowująca pojedyńczy wynik powinna zawierać dwa pola: wynik i nick gracza. Wszystkie wyniki powinny być przechowywane w tablicy struktur. Przygotuj funkcje pozwalające na:

- odczyt i zapis wyników do pliku binarnego (wyniki powinny być posortowane od nawyższego do najniższego),
- dodawanie nowego rekordu do tablicy wyników,
- sortowania tablicy według wyników,
- wyświetlania tablicy wyników na konsoli (wyniki zawsze muszą być wyświetlane od najlepszego do najgorszego).

Napisz program, który będzie testował opracowane przez Ciebie funkcjonalności. Program ma umożliwić następujące rzeczy:

- tworzenie nowej tablicy wyników (użytkownik podaje ile chce rekordów mieć w tablicy, program ma uzupełnić ją losowymi danymi),
- wyświetlenie tablicy rekordów na konsoli,
- dodanie pojedynczego rekordu (dane wpisuje użytkownik),
- usunięcie pojedynczego rekordu (rekord wskazuje użytkownik),
- zapisanie tablicy rekordów do pliku,
- odczytanie tablicy rekordów z pliku.

Zadanie 9 - BIBLIOTEKA.

Zaprojektuj strukturę do przechowywania informacji o książkach, takich jak: imię i nazwisko autora, tytuł książki, gatunek (w tym celu wykorzystaj typ ENUM), rok wydania. Utwórz tablicę książek oraz napisz funkcje pozwalające na:

- odczyt i zapis bazy bazy danych do pliku (zaproponuj swój format przechowywania danych w pliku),
- dodawanie rekordu do bazy danych,
- usuwanie rekordu z bazy danych,
- wyświetlanie bazy danych,
- funkcje sortujące bazę danych według poszczególnych pól,
- funkcje wyszukujące książki danego autora, rodzaju.

Napisz program, który będzie testował opracowane przez Ciebie funkcjonalności. Program ma umożliwić następujące rzeczy:

- tworzenie nowej bazy książek,
- wyświetlenie tablicy rekordów na konsoli,
- usunięcie pojedynczego rekordu (rekord wskazuje użytkownik),
- zapisanie tablicy rekordów do pliku,
- odczytanie tablicy rekordów z pliku,
- wyświetlenie książek zadanego autora,
- wyświetlenie wszystkich książek danego gatunku.

Stos

Stos jest liniową strukturą danych, w której dane odkładane są na wierzch stosu i z wierzchołka stosu są pobierane (bufor typu LIFO, Last In, First Out; ostatni na wejściu, pierwszy na wyjściu). W przypadku stosu możliwy jest dostęp tylko do ostatnio dodanej wartości. Dostęp do elementów znajdujących się na 'dole' stosu możliwy jest dopiero po zdjęciu elementów znajdujących się nad nimi.

Zadanie 1 – Stos pierwsze podejście.

Zaimplementuj api obsługujące stos wartości typu int. Zaproponuj strukturę `stack_t`.

Zaimplementuj następujące funkcje API:

Funkcja tworzy nowy stos o wielkości size elementów typu `int` i zwraca wskaźnik do tego stosu.

```
stack_t* stack_create(int size);
```

Funkcja kładzie na stos nową wartość.

```
void stack_push(stack_t* pstack, int value);
```

Funkcja zdejmuje ze stosu istniejącą wartość oraz ją zwraca.

```
int stack_pop(stack_t* pstack);
```

Funkcja sprawdza, czy stos jest pusty. Jeśli tak, zwraca wartość prawdy (1).

```
bool stack_empty(const stack_t* pstack);
```

Funkcja zwalnia cała pamięć związaną ze stosem `pstack`.

```
void stack_free(stack_t* pstack);
```

Funkcja wyświetla wszystkie elementy stosu, bez jego modyfikowania.

```
void stack_print(const stack_t* pstack);
```

Pamiętaj, że do pół struktury `stack_t` mogą odwoływać się jedynie funkcje API stosu.
Podpowiedź: nie utożsamiaj stosu z tablicą. Stos to nie tylko zbiór odpowiednio ułożonych elementów, ale też informacja o tym, ile ich tam jest i jak dużo może być.

Przykład uzycia:

```
struct stack_t* stos = stack_create(10);
stack_push(stos, 10);
stack_push(stos, 20);
stack_push(stos, 30);
stack_print(stos);
printf("%d\n", stack_pop(stos)); // 30
stack_print(stos); // 20 10
```

Zadanie 2 – Stos – mała modyfikacja.

Zmodyfikuj API stosu z poprzedniego zadania tak, aby:

- funkcja `stack_push` była w stanie informować program wywołujący o tym, że stos jest przepełniony (np. zwracając **true**/**false**).
- funkcja `stack_pop` była w stanie informować program, że na stosie nie zastała żadnego elementu, który można by z niego zdjąć.

Uwaga! Przy tak dość różnym sposobie działania nowych wersji push/pop dobrze jest nie modyfikować istniejących a dodać nowe, np: `stack_try_push` (próbuj położyć wartość na stos, jak się nie uda - zwróć informacje o tym fakcie).

Zadanie 3 – Stos – modyfikacja na Arduino.

Napisz program, który umożliwi dodawanie do stosu nowego elementu (o losowej wartości) po naciśnięciu dowolnego przycisku oraz zdejmowanie elementu, po naciśnięciu przycisku. Za każdym razem jak do stosu dodawany będzie element ma się zapalić kolejna dioda z odpowiednią jasnością. W momencie zdejmowania elementu dioda ma gasnąć a na wyświetlaczu LCD ma zostać wyświetlona wartość usuniętego elementu.

Zadanie 4 – Stos – dalsze modyfikacje.

Dodaj funkcje realizujące następujące operacje:

- funkcję usuwającą wszystkie elementy ze stosu,
- funkcję zapisującą zawartość stosu do pliku binarnego,
- funkcję wczytującą zawartość stosu z pliku binarnego

Uwaga 1! Wczytanie/zapisanie stosu to nie tylko wczytanie/zapisanie bufora danych, ale również wszystkich informacji koniecznych do skorzystania ze stosu po jego wczytaniu.
Uwaga 2! Utrzymaj nomenklaturę funkcji.

Zadanie 5 – Stosy – różne oblicza.

Do tej pory kod obsługi stosu uwzględniał tylko jeden typ danych: `int`.

Zaimplementuj wersje stosu dla następujących typów:

- `char`, `unsigned char`,
- `short`, `unsigned short`,
- `int`, `unsigned int`,
- `long int`, `unsigned long int`,
- `long long int`, `unsigned long long int`,
- `float`, `double`.

Utworzone wersje API stosu muszą pozwalać pracować z nimi w ramach tego samego kodu źródłowego (np. program wykorzystujący dwa stosy typu `int` oraz 1 typu `float`). Przykład:

```
stack_int_t* stos1 = stack_int_create(32);
stack_int_t* stos2 = stack_int_create(32);
stack_float_t* stos3 = stack_float_create(123);
```