ISTOTNE UWAGI DO PONIŻSZYCH ZADAŃ:

- Nie należy stosować zmiennych globalnych;
- Kod winien być pisany przejrzyście z **uwzględnieniem wcięć** podkreślających poziom polecenia;
- Przed przystąpieniem do wykonania obu zadań proszę zapoznać się ze strukturą przykładowych plików nagłówkowych, na przykład: **`stdio.h`**, **`stdlib.h`** czy **`math.h`**
  Proszę zwrócić uwagę na ostatnią linijkę w każdym pliku nagłówkowym.
  Czym ona jest i z którymi wierszami tego pliku jest powiązana?
  Dlaczego stosujemy warunki przy dołączaniu plików nagłowkowych?
  Znając odpowiedzi na powyższe pytania zaproponuj odpowiednią strukturę własnych plików nagłówkowych tj. zawierających definicję odpowiedniej stałej symbolicznej[1].

Zad. 14.

W oparciu o definicję struktury **`POINT`** zawierającą informację o współrzędnych punktu na płaszczyźnie należy zdefiniować dwie funkcje **`area`** i **`perimeter`**, a zwracające odpowiednio wartość pola i odwodu trójkąta o wierzchołkach przekazanych do funkcji.

Odpowiednie deklaracje i definicję struktury przedstawiono poniżej:

```
struct POINT;

float area( struct POINT, struct POINT, struct POINT );
float perimeter( struct POINT, struct POINT, struct POINT );

struct POINT {
   int x;
   int y;
};
```

Dodatkowe uwagi:

- Deklaracje struktury i obu funkcji a także definicję struktury należy umieścić w pliku nagłówkowym o nazwie: **`point.h`**. Każdą definicję funkcji umieszczamy osobno w danym pliku o stosownych nazwach odpowiadających nazwie danej funkcji tj. **`area.c`** i **`perimeter.c`**,
- Następnie proszę napisać krótki program testujący (**`zad14.c`**), który pobiera od użytkownika współrzędne wierzchołków trójkąta i wypisuje wartość pola i obwodu trójkąta.
- Proszę zwrócić uwagę, że nie każda biblioteka jest dostępna i przed wykonaniem zadania należy zapoznać się opcjami: **`l`** i **`L`** kompilatora **`gcc`**.
- Proszę nie stosować **`typedef`** a nagłówki definiowanych funkcji muszą być w pełni zgodne z ich deklaracjami, które podano w zadaniu.
- Wartość pola trójkąta należy wyliczać bez korzystania z pierwiastka kwadratowego.
  (Proszę zastanowić się dlaczego...)
- W powyższych funkcjach nie należy używać żadnych funkcji wejścia-wyjścia (**`printf`**, czy **`scanf`**). Zaimplementowane funkcje mają odebrać wartość zmiennych strukturalnych, wykonać zadane obliczenia i zwrócić odpowiednią wartość – innymi słowy nie stosujemy pobierania i wypisywania wartości wewnątrz tych funkcji. *(Proszę zastanowić się, dlaczego tak powinno się postępować.)*

---

[1] Przykładowy ostatni wiersz z pliku `stdio.h`

pod macOS:

```
% tail -1 stdio.h
#endif /* _STDIO_H_ */
```

pod Debian Linux:

```
% tail -1 stdio.h
#endif /* !_STDIO_H */
```

Należy znaleźć inne wiersze, gdzie zdefiniowano stosowną stałą symboliczną oraz zastanowić się jak jest ona powiązana z nazwą pliku. Znalezienie stosownych wierszy pozwoli zrozumieć strukturę plików nagłówkowych i kompilację warunkową. Na przykład za pomocą:

```
% grep _STDIO_H /usr/include/stdio.h | head -2
% grep _STDLIB_H /usr/include/stdlib.h | head -2
% grep _MATH_H /usr/include/math.h | head -2
```

Zad. 15.

Proszę utworzyć stosowny plik nagłówkowy: **matrix.h** w którym należy umieścić deklaracje stosownych funkcji, definicję struktury **MATRIX_S** oraz wprowadzić za pomocą polecenia **typedef** nazwę **MATRIX** stanowiącą alias dla struktury **MATRIX_S**. Definicja struktury została przedstawiona w ramce. Deklaracje oznaczone nawiasem klamrowym należy potraktować jako pewne propozycje i zmodyfikować je zgodnie z potrzebami. Wykorzystując zdefiniowany alias proszę utworzyć również brakujące deklaracje funkcji a następnie definicje wszystkich funkcji oraz program testujący ich działanie (**zad15.c**). Opis działania funkcji przedstawiono w tabeli.

```
struct MATRIX_S {
   int x;  /* liczba wierszy */
   int y;  /* liczba kolumn */
   int * wsk;  /* adres tablicy x*y elementowej */
};

MATRIX m_create(int, int);
int m_remove(MATRIX * );

int m_scanf(MATRIX *, int, int);
int m_scanf_(MATRIX *);

int m_printf(MATRIX);
```

?

Proszę zastanowić się i zaproponować własne deklaracje funkcji, kiedy będzie to potrzebne, rozważając:

- *jak najlepiej przekazywać zmienną strukturalną do danej funkcji.*
- *czy w każdym języku (C/C++) można przeciążać funkcje*

m_create	Tworzy zmienną typu **MATRIX** na podstawie przekazanego rozmiaru. UWAGA: Wewnątrz zmiennej należy utworzyć dynamicznie tablicę jednowymiarową o rozmiarze stanowiącym iloczyn liczby wierszy i liczby kolumn. Funkcja zwraca wartość **MATRIX** z odpowiednio ustawionymi polami. Jeśli tablica nie została utworzona to wartości wszystkich pól mają być wyzerowane;
m_remove	Zwalnia pamięć i zeruje wartości w zmiennej typu **MATRIX**. UWAGA: Funkcja zwraca wartość 1 jeśli operacja została wykonana poprawnie lub 0 gdy nie udało się wykonać czynności – na przykład przekazany adres zmiennej typu **MATRIX** wynosi **NULL**;
m_get	Pozwala na odczytanie wartości znajdujących się w określonym miejscu (współrzędne) określonej macierzy;
m_put	Pozwala wstawić określoną wartość w określone miejsce danej macierzy;
m_printf	Wyświetla strukturę macierzy tj. wartości poszczególnych komórek macierzy. UWAGA: Funkcja zwraca wartość 1 jeśli operacja została wykonana poprawnie lub 0 gdy nie udało się wykonać czynności – na przykład wartość pola wsk wynosi **NULL** lub wartości dowolnego pola rozmiaru są mniejsze lub równe zero;
m_scanf ***m_scanf_***	Proszę zaproponować dwie różne implementacje funkcji. W pierwszym przypadku dana funkcja pozwala wypełnić wartościami (przekazanym od użytkownika w trakcie działania tej funkcji) całą istniejącą macierz. W drugim przypadku powinna sprawdzić, czy macierz istnieje, a jeśli tak, to czy rozmiary są zgodne i w zależności od tego utworzyć macierz, jeśli tablica wartości nie istnieje, wykorzystać, jeśli istnieje i rozmiar jej zgodny to ją wykorzystać, jeśli jest inny, to zwolnić pamięć i utworzyć na nowo właściwy obszar lub użyć **realloc** do zmiany rozmiaru pamięci. Po weryfikacji przekazanej macierzy i ewentualnym wykonaniu opisanych czynności funkcja ta powinna później zadziałać jak jej poprzedni odpowiednik *(może warto wywołać wtedy w odpowiedni sposób: **m_scanf**?)*; UWAGA: • Proszę zwrócić uwagę, że przeciążanie funkcji pojawia się dopiero w C++. • Funkcje zwracają wartość stanowiącą rozmiar tablicy w bajtach lub zero, jeśli pojawił się problem, na przykład przekazano niewłaściwy rozmiar macierzy, nie udało się zmienić rozmiaru tablicy, itp.

Dodatkowe uwagi:

- Tylko w funkcjach: **m_printf** i **m_scanf** można używać funkcji wejścia-wyjścia (**printf** i **scanf**). Pozostałe funkcji służą tylko i wyłącznie do wykonania pewnych działań i komunikacja z użytkownikiem bezpośrednio nie zachodzi.
- Jeśli funkcje nie mają zwracać specyficznej wartości np. wartości elementu macierzy, to należy przyjąć zasadę, że zwracają one wartość **0** w przypadku błędu. (Szczegóły opisano w tabelce powyżej)
- Należy zapoznać się z poleceniem **typedef** a następnie zdefiniować alias **MATRIX** dla struktury **MATRIX_S**;
- Deklaracje (oraz def. struktury i aliasu **MATRIX**) należy umieścić w pliku nagłówkowym: **matrix.h**, definicje funkcji w osobnych plikach (każda funkcja w osobnym pliku!!! o nazwie takiej jak nazwa funkcji + rozszerzenie **.c**)