## *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 plików bibliotecznych takich jak: **`stdio.h`**, **`stdlib.h`** oraz **`math.h`**
  Proszę zwrócić uwagę na ostatnią linijkę w każdym pliku nagłówkowym.
  *Czemu on służy i z którymi wierszami tego pliku jest powiązana?*
  *Jaka stała symboliczna odpowada za kompilację warunkową?*
  Znając odpowiedzi na powyższe pytania zaproponuj odpowiednią strukturę własnych plików nagłówkowych tj. zawierających definicję odpowiedniej stałej symbolicznej oraz stosując zasady kompilacji warunkowej [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 funkcji i definicję struktury przedstawiono poniżej:

```
struct POINT {
   int x;
   int y;
}
float area( struct POINT, struct POINT, struct POINT );
float perimeter( struct POINT, struct POINT, struct POINT
);
```

**Dodatkowe uwagi**:

- Deklaracje obu funkcji oraz definicję struktury należy umieścić w pliku nagłówkowym o nazwie: **`point.h`**, zaś każdą definicję funkcji umieszczamy osobno w danym pliku o stosownych nazwach odpowiadających nazwie danej funkcji tj. **`area.c`** i **`perimeter.c`**, a następnie napisać krótki program testujący (**`zad14.c`**).
- 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 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. Zaimplementowane funkcje mają odebrać wartość zmiennych strukturalnych, wykonać zadane obliczenia i zwrócić odpowiednią wartość – innymi słowy nie stosujemy pobierania i wypisywania wartości wewnątrz każdej z funkcji
  *(Proszę zastanowić się dlaczego tak powinno się postępować.)*

## Zad. 15.

Proszę utworzyć stosowny plik nagłówkowy: **`matrix.h`** w którym należy umieścić deklaracje stosownych funkcji, definicję struktury **`MATRIX_S`** oraz wprowadzić za pomocą polecenie **`typedef`** nazwę **`MATRIX`** stanowiącą alias dla struktury **`MATRIX_S`**. Definicja struktury została przedstawiona w ramce.
Następnie wykorzystując zdefiniowany alias proszę utworzyć brakujące deklaracje funkcji oraz definicje wszystkich funkcji oraz program testujący ich działanie (**`zad15.c`**). Opis działania funkcji przedstawiono w tabeli.

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

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

```
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);
```

?

*Należy zdecydować się jak najlepiej przekazywać zmienną strukturalną do każdej z funkcji.*

| | |
|---|---|
| **m_create** | Tworzy zmienną typu MATRIX na podstawie przekazanego rozmiaru.<br>UWAGA: Wewnątrz zmiennej należy utworzyć dynamicznie tablicę jednowymiarową o rozmiarze stanowiącym iloczyn liczby wierszy i liczby kolumn.<br>Funkcja zwraca wartość stanowiącą rozmiar tablicy w bajtach lub zero jeśli tablica nie została utworzona.; |
| **m_remove** | Zwalnia pamięć i zeruje wartości w zmiennej typu MATRIX.<br>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.<br>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** | Proszę wprowadzić dwie różne implementacje tej funkcji. W pierwszym przypadku funkcja pozwala wypełnić wartościami całą istniejącą macierz. W drugim przypadku powinna sprawdzić, czy macierz istnieje i rozmiary są zgodne i w zależności od tego zwolnić pamięć / utworzyć na nowo właściwy obszar, następnie powinna zadziałać jak jej poprzedni odpowiednik;<br>UWAGA: Funkcja zwraca wartość stanowiącą rozmiar tablicy w bajtach lub zero jeśli pojawił się problem, na przykład przekazano niewłaściwy rozmiar macierzy, 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 nazwę **MATRIX** dla struktury **MATRIX_S**;
- Deklaracje (oraz def. struktury i nazwy **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**)