Podstawy programowania
Semestr letni 2018/19
Materiały z laboratorium i zadania domowe

Przemysław Olbratowski

5 czerwca 2019

Spis treści

1 Podstawy: 19 lutego 2019
Wejście-wyjście, zmienne, warunki i wybory

1.1 Przykłady laboratoryjne z działu Podstawy

1.1.1 Area: Pola figur płaskich

Napisz program `area` obliczający pole koła lub trójkąta. Po uruchomieniu program wczytuje ze standardowego wejścia liczbę 1 lub 2. Jeżeli wpisano 1, wczytuje promień koła i wypisuje na standardowe wyjście jego pole. Jeżeli wpisano 2, wczytuje długości trzech boków trójkąta i wypisuje jego pole. Pole A trójkąta o bokach a, b, c, dane jest wzorem Herona:

$$A = \sqrt{s(s-a)(s-b)(s-c)} \quad \text{gdzie} \quad s = \frac{a+b+c}{2},$$

Jeżeli ktorąkolwiek daną podano niepoprawnie, na przykład trzy boki nie spełniają warunku trójkąta, program wypisuje komunikat `error`.

Przykładowe wykonanie

```
 In: 2
 In: 3 4 5
Out: 6
```

Rozwiązanie

```
#include <math.h>
#include <stdio.h>

#define M_PI 3.14159265358979323846

int main() {
    int figure;
    scanf("%i", &figure);
    switch (figure) {
    case 1: {
        double r;
        scanf("%lg", &r);
        if (r >= 0.) {
            double p = M_PI * pow(r, 2);
            printf("%lg\n", p); }
        else {
            printf("error\n"); }
        break; }
    case 2: {
        double a, b, c;
        scanf("%lg%lg%lg", &a, &b, &c);
        if (a >= 0. && b >= 0. && c >= 0. && a <= b + c && b <= c + a && c <= a + b) {
            double s = (a + b + c) / 2.;
            double p = sqrt(s * (s - a) * (s - b) * (s - c));
            printf("%lg\n", p); }
        else {
            printf("error\n"); }
        break; }
    default: {
        printf("error\n"); }}
    return 0; }
```

1.2 Zadania laboratoryjne z działu Podstawy

1.2.1 Age: Wiek użytkownika

Napisz program `age`, który wczytuje ze standardowego wejścia rok bieżący oraz rok urodzenia użytkownika i wypisuje na standardowe wyjście wiek użytkownika w latach. Jeżeli rok bieżący wypada przed rokiem urodzenia, program wypisuje komunikat `error`. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 2017 2000
Out: 17
```

Rozwiązanie

```
#include <stdio.h>

int main() {
    int current, birth;
    scanf("%i%i", &current, &birth);
    if (!(current < birth)) {
        printf("%i\n", current - birth); }
    else {
        printf("error\n"); }
    return 0; }
```

1.3 Zadania domowe z działu Podstawy na 12 marca

1.3.1 Temperature: Skale temperatury

Temperatura w stopniach Celsjusza jest o 273.15 niższa niż w Kelwinach. Temperatura w stopniach Rankina jest dziewięć piątych razy większa niż w Kelwinach. Temperatura w stopniach Réaumura to temperatura w stopniach Celsjusza pomnożona przez cztery piąte. Napisz program `temperature`, który wczytuje ze standardowego wejścia temperaturę w Kelwinach i wypisuje na standardowe wyjście odpowiadające jej temperatury w stopniach Celsjusza, Rankina i Réaumura. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 200
Out: -73.15 360 -58.52
```

1.3.2 Barometric: Wzór barometryczny

Wyrażoną w metrach wysokość h nad poziomem morza można obliczyć ze wzoru barometrycznego

$$h = -\frac{RT}{\mu g} \log\left(\frac{p}{p_0}\right)$$

gdzie $R = 8.3144598$, $\mu = 0.0289644$, $g = 9.80665$, $p_0 = 1013.25$, zaś p oraz T są odpowiednio ciśnieniem atmosferycznym w hektopaskalach i temperaturą powietrza w kelwinach. Napisz program `barometric`, który wczytuje ze standardowego wejścia ciśnienie w hektopaskalach oraz temperaturę w Kelwinach i wypisuje na standardowe wyjście wysokość w metrach. Program załącza tylko pliki nagłówkowe `math.h` i `stdio.h`.

Przykładowe wykonanie

```
 In: 955 290
Out: 502.596
```

1.3.3 Dice: Rzut dwiema kostkami

Napisz program `dice` symulujący rzut dwiema sześciennymi kostkami do gry. Program wypisuje na standardowe wyjście liczby oczek na obu kostkach oraz ich sumę. Przy każdym uruchomieniu wyniki powinny być inne. Program załącza tylko pliki nagłówkowe `stdio.h`, `stdlib.h` i `time.h`.

Przykładowe wykonanie

```
Out: 5 2 7
```

1.3.4 Leap: Lata przestępne - 1 bitcoin

Według kalendarza gregoriańskiego przestępne są lata podzielne przez 4 z wyjątkiem lat podzielnych przez 100 ale niepodzielnych przez 400. Napisz program `leap`, który wczytuje ze standardowego wejścia rok i wypisuje na standardowe wyjście `true` jeśli jest on przestępny, albo `false` w przeciwnym razie. Program załącza tylko plik nagłówkowy `stdio.h`.

Wykonanie

```
 In: 2000
Out: true
```

Uwaga Spróbuj napisać program bez użycia instrukcji wyboru ani instrukcji warunkowej.

1.3.5 Triangle: Warunek trójkąta

Napisz program `triangle`, który wczytuje ze standardowego wejścia długości trzech odcinków i wypisuje na standardowe wyjście `true` jeśli można z nich zbudować trójkąt, albo `false` w przeciwnym razie. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 2.1 7 4.3
Out: false
```

Uwaga Z trzech odcinków można zbudować trójkąt jeśli długość każdego z nich jest mniejsza od sumy długości dwóch pozostałych. Spróbuj napisać program bez użycia instrukcji warunkowej.

1.3.6 Shop: Godziny otwarcia sklepu

Pewien sklep jest czynny od 10:30 włącznie do 18:30 wyłącznie. Napisz program `shop`, który wczytuje ze standardowego wejścia godzinę w formacie `hh mm`, i wypisuje na standardowe wyjście `true`, jeśli sklep jest wtedy otwarty, albo `false` jeśli jest zamknięty. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 11 45
Out: true
```

Uwaga Spróbuj napisać program bez użycia instrukcji warunkowej.

1.3.7 Signum: Gra w znaki

Signum to gra dla dwóch osób, z których jedna przyjmuje rolę pozytywnej, a druga negatywnej. Każda osoba zapisuje w ukryciu liczbę jeden lub minus jeden. Następnie osoby odkrywają swoje liczby i jeżeli ich iloczyn jest dodatni, to wygrywa gracz pozytywny, zaś w przeciwnym razie wygrywa gracz negatywny. Napisz program `signum` grający z użytkownikiem. Po uruchomieniu program losowo wybiera swoją liczbę, ale nie wyświetla jej. Następnie wczytuje ze standardowego wejścia liczbę użytkownika. Potem wypisuje na standardowe wyjście swoją liczbę oraz `true` jeśli wygrywa gracz pozytywny albo `false` jeśli negatywny. Program załącza tylko pliki nagłówkowe `stdio.h`, `stdlib.h` i `time.h`.

Przykładowe wykonanie

```
 In: -1
Out: -1
Out: true
```

Uwaga Spróbuj napisać program bez użycia instrukcji wyboru ani instrukcji warunkowej.

1.3.8 BMI: Indeks masy ciała

Indeks Masy Ciała BMI, z angielskiego *Body-Mass Index*, to masa wyrażona w kilogramach dzielona przez kwadrat wzrostu wyrażonego w metrach. Wagę człowieka można ocenić według następującej tabelki:

BMI	Waga
Poniżej 18.5	Niedowaga
18.5 - 25	Norma
25 - 30	Nadwaga
Powyżej 30	Otyłość

Napisz program `bmi`, który wczytuje ze standardowego wejścia masę w kilogramach oraz wzrost w centymetrach i wypisuje na standardowe wyjście komunikat `underweight`, `normal`, `overweight`, lub `obese`. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 80 178
Out: overweight
```

1.3.9 History: Test z historii

Napisz program `history` przeprowadzający test z historii świata. Program wypisuje na standardowe wyjście trzy pytania o rok jakiegoś wydarzenia i po każdym wczytuje ze standardowego wejścia odpowiedź. Po poprawnej odpowiedzi wypisuje `true`, a po niepoprawnej `false`. Na końcu wypisuje liczbę poprawnych odpowiedzi. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
Out: foundation of rome  In: -753
Out: true
Out: discovery of america  In: 1975
Out: false
Out: first airplane flight  In: 1903
Out: true
Out: 2
```

1.3.10 Holidays: Dni wolne od pracy

W Polsce obowiązują następujące święta stałe wolne od pracy:

1 stycznia	Nowy Rok	New Year
6 stycznia	Trzech Króli	Epiphany
1 maja	Święto Pracy	Worker's Day
3 Maja	Święto Konstytucji 3 Maja	Constitution Day
15 sierpnia	Święto Wojska Polskiego	Armed Forces Day
1 listopada	Wszystkich Świętych	All Saints' Day
11 listopada	Święto Niepodległości	Independence Day
25 grudnia	Boże Narodzenie	Christmas
26 grudnia	Boże Narodzenie	Chirstmas

Napisz program `holidays`, który wczytuje ze standardowego wejścia datę w formacie `mm dd`. Jeżeli wypada wtedy święto stałe wolne od pracy, program wypisuje na standardowe wyjście małymi literami angielską nazwę święta. W przeciwnym razie wypisuje słowa `ordinary day`. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 8 15
Out: armed forces day
```

Uwaga Spróbuj napisać program bez użycia instrukcji warunkowej ani operatora warunkowego.

1.3.11 Countdown: Odliczanie przed startem

Napisz program `countdown` przeprowadzający odliczanie przed startem. Program wczytuje ze standardowego wejścia liczbę całkowitą. Jeżeli należy ona do przedziału od zera do dziesięciu włącznie, to wypisuje na standardowe wyjście angielskie nazwy liczb od podanej w dół do zera, a na końcu słowo `start`. W przeciwnym razie wypisuje słowo `stop`. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 7
Out: seven six five four three two one zero start
```

Uwaga Użyj instrukcji wyboru. Spróbuj napisać program tak, aby nazwa każdej liczby wystąpiła w nim tylko raz.

1.3.12 Calculator: Kalkulator pięciodziałaniowy - 1 bitcoin

Napisz program `calculator` wykonujący cztery podstawowe działania arytmetyczne oraz wyciągający pierwiastek kwadratowy. Po uruchomieniu program wczytuje ze standardowego wejścia liczbę 1, 2, 3, 4, lub 5, co odpowiada dodawaniu, odejmowaniu, mnożeniu, dzieleniu i pierwiastkowaniu. Następnie wczytuje argumenty wybranego działania i wypisuje na standardowe wyjście jego wynik. Program załącza tylko pliki nagłówkowe `math.h` i `stdio.h`.

Przykładowe wykonanie

```
 In: 2
 In: 7.5 5.2
Out: 2.3
```

Uwaga Użyj instrukcji wyboru.

2 Pętle: 26 lutego 2019
Pętle, format wydruku

2.1 Przykłady laboratoryjne z działu Pętle

2.1.1 Loops: Pętle

Napisz program `loops`, który wczytuje ze standardowego wejścia liczbę naturalną n i wypisuje na standardowe wyjście w kolejnych liniach:

1. wszystkie liczby całkowite od 0 włącznie do n wyłącznie, w kolejności rosnącej,
2. wszystkie liczby niepodzielne przez 3 od 0 włącznie do n wyłącznie, w kolejności rosnącej,
3. wszystkie liczby podzielne przez 3 od 0 włącznie do n wyłącznie, w kolejności rosnącej,
4. wszystkie liczby parzyste większe lub równe $-n$ i mniejsze lub równe n, w kolejności malejącej.

Przykładowe wykonanie

```
 In: 10
Out: 0 1 2 3 4 5 6 7 8 9
Out: 1 2 4 5 7 8
Out: 0 3 6 9
Out: 10 8 6 4 2 0 -2 -4 -6 -8 -10
```

Rozwiązanie

```
#include <stdio.h>

int main() {
    int n;
    scanf("%i", &n);
    for (int i = 0; i < n; ++i) {
        printf("%i ", i); }
    printf("\n");
    for (int i = 0; i < n; ++i) {
        if (i % 3) {
            printf("%i ", i); }}
    printf("\n");
    for (int i = 0; i < n; i += 3) {
        printf("%i ", i); }
    printf("\n");
    for (int i = n / 2 * 2; i >= -n; i -= 2) {
        printf("%i ", i); }
    printf("\n");
    return 0; }
```

2.1.2 Multi: Drukowanie tabliczki mnożenia

Napisz program `multi`, który drukuje na standardowe wyjście tabliczkę mnożenia liczb od 1 do 10 sformatowaną jak w poniższym przykładzie.

Wykonanie

```
Out:    1   2   3   4   5   6   7   8   9  10
Out:    2   4   6   8  10  12  14  16  18  20
Out:    3   6   9  12  15  18  21  24  27  30
Out:    4   8  12  16  20  24  28  32  36  40
Out:    5  10  15  20  25  30  35  40  45  50
Out:    6  12  18  24  30  36  42  48  54  60
Out:    7  14  21  28  35  42  49  56  63  70
Out:    8  16  24  32  40  48  56  64  72  80
Out:    9  18  27  36  45  54  63  72  81  90
Out:   10  20  30  40  50  60  70  80  90 100
```

Rozwiązanie

```
#include <stdio.h>

int main () {
    for (int row = 1; row <= 10; ++row) {
        for (int column = 1; column <= 10; ++column) {
            printf("%4i", row * column); }
        printf("\n"); }
    return 0; }
```

2.2 Zadania laboratoryjne z działu Pętle

2.2.1 Factorial: Silnia

Napisz program `factorial`, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą i wypisuje na standardowe wyjście jej silnię. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 5
Out: 120
```

Rozwiązanie

```
#include <stdio.h>

int main() {
    int number, factorial = 1;
    scanf("%i", &number);
    while (number) {
        factorial *= number--; }
    printf("%i\n", factorial);
    return 0; }
```

2.3 Zadania domowe z działu Pętle na 19 marca

2.3.1 Fibonacci: Ciąg Fibonacciego - 1 bitcoin

Ciąg Fibonacciego zaczyna się od wyrazów 0 i 1, a każdy następny jest sumą dwóch poprzednich. Napisz program `fibonacci`, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i drukuje na standardowe wyjście n pierwszych wyrazów ciągu Fibonacciego. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 10
Out: 0 1 1 2 3 5 8 13 21 34
```

2.3.2 Section: Losowy podział odcinka

Napisz program `section`, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą n, po czym wypisuje na standardowe wyjście n losowych liczb nieujemnych, które dają w sumie jeden. Program załącza tylko pliki nagłówkowe `stdio.h`, `stdlib.h` i `time.h`.

Przykładowe wykonanie

```
 In: 3
Out: 0.54095 0.345354 0.113696
```

2.3.3 Pi: Oszacowanie liczby pi metodą Monte-Carlo

Przybliżoną wartość liczby π można wyznaczyć następująco. Rozważmy kwadrat o wierzchołkach w punktach $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$ oraz zawartą w nim ćwiartkę koła o środku w punkcie $(0,0)$ i promieniu 1. Pola kwadratu i ćwiartki koła wynoszą odpowiednio $A_s = 1$ i $A_c = \pi/4$. Wybierzmy losowo dużo punktów tak, aby równomiernie wypełniały cały kwadrat. Stosunek liczby punktów wewnątrz ćwiartki koła do liczby wszystkich punktów w kwadracie jest w przybliżeniu równy stosunkowi pól tych figur, $N_c/N_s \approx A_c/A_s$. Dostajemy stąd

$$\pi \approx 4N_c/N_s$$

Napisz program `pi`, który czyta ze standardowego wejścia liczbę punktów do wylosowania i wypisuje na standardowe wyjście otrzymane przybliżenie liczby π. Program załącza tylko pliki nagłówkowe `stdio.h`, `stdlib.h` i `time.h`.

Przykładowe wykonanie

```
 In: 10000
Out: 3.144
```

2.3.4 Sum: Suma skończona

Napisz program `sum`, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą n i wypisuje na standardowe wyjście sumę

$$4\sum_{k=1}^{n} \frac{(-1)^{k+1}}{2k-1}$$

Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 1000
Out: 3.14059
```

2.3.5 Nominals: Nominały

Napisz program `nominals`, który wczytuje ze standardowego wejścia nieujemną liczbę liczbę całkowitą i wypisuje na standardowe wyjście wszystkie mniejsze od niej liczby postaci 1, 2, 5, 10, 20, 50,... Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 1000
Out: 1 2 5 10 20 50 100 200 500
```

2.3.6 Collatz: Hipoteza Collatza

Jeżeli liczba jest parzysta, to dzielimy ją przez dwa, a jeśli jest nieparzysta, to mnożymy przez trzy i dodajemy jeden. Z otrzymanym wynikiem postępujemy tak samo. Hipoteza Collatza mówi, że w końcu zawsze otrzymamy jeden. Do tej pory nie wiadomo, czy jest to prawda. Napisz program `collatz`, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą i wypisuje na standardowe wyjście kolejne liczby otrzymane w opisany sposób, od podanej aż do jedynki. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 1
Out: 1 4 2 1
```

2.3.7 Precision: Dokładność maszynowa

Dokładnością maszynową nazywamy najmniejszą potęgę dwójki, która dodana do jedynki daje wynik numerycznie różny od jednego. Pojęcie to odnosi się tylko do liczb zmiennoprzecinkowych i chodzi tu o potęgę z wykładnikiem ujemnym. Dokładność maszynowa jest różna dla zmiennych różnego typu. Napisz program `precision`, który doświadczalnie wyznacza i wypisuje na standardowe wyjście dokładności maszynowe typów `float`, `double`, oraz `long double`. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
Out: 5.960464e-08 1.110223e-16 5.421011e-20
```

Uwaga Dokładność maszynową można wyznaczyć doświadczalnie w następujący sposób. Zaczynamy od zerowej potęgi dwójki równej jeden, a następnie rozpatrujemy coraz mniejsze potęgi. Każdą kolejną potęgę dodajemy do jedynki i sprawdzamy, czy wynik jest numerycznie równy jedności. Trzeba jednak pamiętać, że procesor nie wykonuje operacji zmiennoprzecinkowych bezpośrednio na zmiennych, lecz w swoich wewnętrznych rejestrach. Może się więc zdarzyć, że zamiast wyznaczyć dokładność żądanego typu znajdziemy dokładność rejestrów procesora, która jest na ogół większa. Aby temu zapobiec, wynik każdego dodawania należy najpierw wpisać do zmiennej żądanego typu i dopiero tę zmienną porównać z jedynką. Każdą kolejną potęgę dwójki najprościej wyznaczyć dzieląc poprzednią potęgę przez dwa.

2.3.8 Guess: Odgadywanie liczby

Napisz program `guess` odgadujący pomyślaną przez użytkownika liczbę. Przed uruchomieniem użytkownik wybiera losowo liczbę całkowitą z przedziału od zera włącznie do stu wyłącznie. Po uruchomieniu program wypisuje na standardowe wyjście pewną liczbę z tego przedziału i wczytuje ze standardowego wejścia odpowiedź użytkownika równą `-1`, `0`, lub `+1`. Odpowiedzi te oznaczają odpowiednio, że pomyślana liczba jest mniejsza, równa, lub większa od liczby wyświetlonej przez program. Program kontynuuje zgadywanie aż do odgadnięcia właściwej liczby. Program powinien odgadnąć tę liczbę w nie więcej niż siedmiu próbach. Program załącza tylko plik nagłówkowy `stdio.h`.

Uwaga Zastosuj tak zwane wyszukiwanie binarne. Pomyślana przez użytkownika liczba leży w przedziale od 0 do 100. Na początku program wyświetla liczbę leżącą w połowie tego przedziału, czyli 50. Jeżeli użytkownik odpowie -1, to jego liczba leży w przedziale od 0 do 50, i w następnej próbie program wyświetla

liczbę w połowie tego przedziału, czyli 25. Jeżeli natomiast użytkownik odpowie +1, to jego liczba leży w przedziale od 50 do 100, i w następnej próbie program wyświetla liczbę w połowie tego przedziału, czyli 75. W ten sposób w każdej próbie program dwukrotnie zawęża przedział, w którym może leżeć pomyślana przez użytkownika liczba. Zapewnia to zgadnięcie liczby w nie więcej niż siedmiu próbach.

2.3.9 Factor: Rozkład na czynniki pierwsze - 1 bitcoin

Napisz program `factor`, który wczytuje ze standardowego wejścia liczbę całkowitą większą od jednego i wypisuje na standardowe wyjście jej rozkład na czynniki pierwsze drukując kolejne liczby w kolejności niemalejącej. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 517
Out: 11 47
```

Uwaga Liczbę naturalną można rozłożyć na czynniki pierwsze w następujący sposób. Sprawdzamy, czy liczba jest podzielna przez 2. Jeżeli tak, to 2 dopisujemy do rozkładu, a samą liczbę dzielimy przez 2. Czynność tę powtarzamy aż liczba przestanie być podzielna przez 2. Następnie w taki sam sposób badamy podzielność przez 3, 4 i tak dalej, aż rozważana liczba stanie się równa 1.

2.3.10 Quiz: Test z tabliczki mnożenia

Napisz program `quiz`, który wypisuje na standardowe wyjście dziesięć losowych pytań z tabliczki mnożenia liczb od jednego do dziesięciu i po każdym pytaniu wczytuje ze standardowego wejścia odpowiedź. Każde pytanie zadaje aż do uzyskania poprawnej odpowiedzi. Program załącza tylko pliki nagłówkowe `stdio.h`, `stdlib.h` i `time.h`.

Przykładowe wykonanie

```
Out: 2 8  In: 16
Out: 5 1  In: 3
Out: 5 1  In: 5
Out: 10 5 In: 50
Out: 9 9  In: 81
Out: 3 5  In: 15
Out: 6 6  In: 36
Out: 2 8  In: 16
Out: 2 2  In: 4
Out: 6 3  In: 18
Out: 8 7  In: 56
```

2.3.11 Stars: Wzorki z gwiazdek

Napisz program `stars`, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowe wyjście wybrany wzorek z gwiazdek złożony z $2n + 1$ wierszy i kolumn. Program załącza tylko pliki nagłówkowe `stdio.h` i `stdlib.h`.

Przykładowe wzorki

```
   *     *******  *     *     *     *******     *
   *     *     *   *   *     * *     *****     ***
   *     *     *    * *     *   *     ***     *****
*******  *     *     *     *     *     *     *******
   *     *     *    * *     *   *     ***     *****
   *     *     *   *   *     * *     *****     ***
   *     *******  *     *     *     *******     *
```

2.3.12 Tri: Liczby trzycyfrowe

Napisz program `tri` wypisujący na standardowe wyjście w kolejności rosnącej wszystkie liczby trzycyfrowe, których cyfra setek to 1, 2, 5, 6, 7, lub 9, cyfra dziesiątek jest potęgą dwójki, cyfra jedności jest parzysta, a suma cyfr dzieli się przez 7. Program wypisuje te liczby przy użyciu odpowiednich pętli i załącza tylko plik nagłówkowy `stdio.h`.

Wykonanie

```
Out: 124 142 214 248 284 518 520 588 610 626 644 680 716 786 914 948 984
```

2.3.13 Polyhedron: Wielościany foremne

W każdym wielościanie foremnym całkowita liczba krawędzi, e_t, liczba krawędzi jednej ściany, e_f, oraz liczba krawędzi wychodzących z jednego wierzchołka, e_v, spełniają zależność

$$2e_t(e_f + e_v) = (2 + e_t)e_f e_v$$

Napisz program `polyhedron`, który wypisuje na standardowe wyjście wszystkie możliwe kombinacje całkowitej liczby ścian, $f_t = 2e_t/e_f$, oraz liczby krawędzi jednej ściany, e_f. Program załącza tylko plik nagłówkowy `stdio.h`.

Wykonanie

```
Out: 4 3
Out: 6 4
Out: 8 3
Out: 12 5
Out: 20 3
```

3 EOF: 5 marca 2019
Koniec pliku, przekierowanie

3.1 Przykłady laboratoryjne z działu EOF

3.1.1 Means: Średnia arytmetyczna i geometryczna

Średnie arytmetyczna oraz geometryczna nieujemnych liczb rzeczywistych x_1, x_2, ..., x_n, dane są wzorami:

$$A = \frac{x_1 + x_2 + ... + x_n}{n} \qquad G = \sqrt[n]{x_1 \times x_2 \times ... \times x_n}$$

Napisz program `means`, który czyta ze standardowego wejścia liczby rzeczywiste do napotkania końca pliku i wypisuje na standardowe wyjście ich średnią arytmetyczną oraz geometryczną.

Przykładowe wykonanie

```
 In: 2.5 0.3 1.7 0.25 3.75 [EOF]
Out: 1.7 1.03633
```

Rozwiązanie

```
#include <math.h>
#include <stdio.h>

int main() {
    double sum = 0., product = 1.;
    int number = 0;
    for (double value; scanf("%lg", &value) == 1; ++number) {
        sum += value;
        product *= value; }
    printf("%lg %lg\n", sum / number, pow(product, 1. / number));
    return 0; }
```

3.2 Zadania laboratoryjne z działu EOF na 26 marca

3.2.1 XOR: Różnica symetryczna

Napisz program `xor`, który czyta ze standardowego wejścia liczby zero lub jeden do napotkania końca pliku i wypisuje na standardowe wyjście ich różnicę symetryczną `xor`. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 0 1 1 0 1
Out: 1
```

Uwaga Różnica symetryczna wielu wartości logicznych jest prawdą jeśli liczba wartości prawdziwych jest nieparzysta, albo fałszem w przeciwnym razie. Spróbuj napisać program bez użycia instrukcji wyboru, instrukcji warunkowej, ani operatora warunkowego.

Rozwiązanie

```
#include <stdio.h>

int main() {
    int result = 0;
    for (int value; scanf("%i", &value) == 1;) {
        result = (result != value); }
    printf("%i\n", result);
    return 0; }
```

3.3 Zadania domowe z działu EOF

3.3.1 Deserter: Brakująca liczba

Spośród liczb naturalnych od 1 do n włącznie usuwamy losowo jedną, a resztę zapisujemy w przypadkowej kolejności. Napisz program `deserter`, który czyta zapisane liczby ze standardowego wejścia do napotkania końca pliku i wypisuje na standardowe wyjście tę brakującą. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 2 5 1 4 [EOF]
Out: 3
```

3.3.2 Statistics: Średnia i błąd

Średnia arytmetyczna x liczb rzeczywistych x_1, ..., x_n oraz jej błąd σ dane są wzorami

$$x = \frac{1}{n}\sum_{k=1}^{n} x_k \qquad \sigma = \sqrt{\frac{1}{n(n-1)}\sum_{k=1}^{n}(x_k - x)^2}$$

Napisz program `statistics`, który czyta ze standardowego wejścia liczby rzeczywiste do napotkania końca pliku i wypisuje na standardowe wyjście ich średnią arytmetyczną oraz jej błąd. Program załącza tylko pliki nagłówkowe `math.h` i `stdio.h`.

Przykładowe wykonanie

```
 In: 1 2 3 4 5 6 7 8 9 10 [EOF]
Out: 5.5 0.957427
```

3.3.3 Minimum: Najmniejsza z wielu liczb - 1 bitcoin

Napisz program `minimum`, który czyta ze standardowego wejścia liczby rzeczywiste do napotkania końca pliku i wypisuje na standardowe wyjście najmniejszą z nich. Jeżeli nie wprowadzono żadnej liczby, program nic nie wypisuje. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 23.5 7.16 2 -1.3 -7 0.13 -1.3 28 -7 23.5 [EOF]
Out: -7
```

3.3.4 Pass: Zagadnienie mijania

Zapisujemy w jednej linii losowo zera i jedynki oddzielając je spacjami. Wyobraźmy sobie, że w pewnej chwili wszystkie zera przesuwają się na początek linii, a wszystkie jedynki na koniec. Napisz program `pass`, który czyta ze standardowego wejścia zapisane liczby do napotkania końca pliku i wypisuje na standardowe wyjście, ile razy zera miną się z jedynkami. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 0 1 1 0 1 0 1 [EOF]
Out: 5
```

3.3.5 EKG: Pulsometr

Plik tekstowy `ekg.txt` zawiera sygnał EKG próbkowany z częstotliwością 128Hz i przyjmujący wartości mniej więcej od 0 do 6000mV. Napisz program `ekg`, który wczytuje ten plik ze standardowego wejścia i wypisuje na standardowe wyjście średnią liczbę uderzeń serca na minutę. Program załącza tylko plik nagłówkowy `stdio.h`.

Uwaga Obejrzyj sygnał w dowolnym programie do sporządzania wykresów. Widać tam wyraźne piki o wysokości około 6000mV. Ich położenia można znaleźć sprawdzając, kiedy sygnał przekracza progową wartość równą przykładowo 3000mV. Dzieje się tak gdy z dwóch kolejnych wartości sygnału pierwsza jest mniejsza, a druga większa od 3000mV. Analizując sygnał wystarczy więc pamiętać dwie ostatnio wczytane wartości. Do wyznaczenia tętna wystarczy znaleźć położenia pierwszego i ostatniego piku oraz liczbę pików między nimi.

3.3.6 Neighbors: Najbliźsi sąsiedzi

Napisz program `neighbors`, który czyta ze standardowego wejścia liczby rzeczywiste do napotkania końca pliku i wypisuje na standardowe wyjście parę tych kolejnych liczb, które się najmniej różnią. Program załącza tylko pliki nagłówkowe `math.h` i `stdio.h`.

Przykładowe wykonanie

```
 In: 0.3 1.5 8.9 8.7 1.6 0.2 [EOF]
Out: 8.9 8.7
```

3.3.7 Polygonal: Długość łamanej - 1 bitcoin

Napisz program `polygonal`, który do napotkania końca pliku czyta ze standardowego wejścia pary liczb rzeczywistych określające kartezjańskie współrzędne punktów na płaszczyznie i wypisuje na standardowe wyjście długość łamanej otwartej łączącej te punkty, od pierwszego do ostatniego. Jeżeli podano tylko jedną parę liczb, program wypisuje zero. Jeżeli nie podano żadnej pary, program nic nie wypisuje. Program załącza tylko pliki nagłówkowe `math.h` i `stdio.h`.

Przykładowe wykonanie

```
 In: -0.3 7.5
 In: 9.5 -3.7
 In: 5.0 0.4
 In: [EOF]
Out: 20.9699
```

4 Funkcje: 12 marca 2019
Funkcje, liczby pseudolosowe

4.1 Przykłady laboratoryjne z działu Funkcje

4.1.1 Ellipse: Pole elipsy

Przybliżone pole elipsy o równaniu

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

można obliczyć następująco. Rozważamy opisany na tej elipsie prostokąt o wierzchołkach w punktach (a, b), $(a, -b)$, $(-a, -b)$, $(-a, b)$. Losujemy wewnątrz tego prostokąta dużo punktów tak by wypełniały go równomiernie. Stosunek liczby punktów wewnątrz elipsy, N_e, do liczby punktów w całym prostokącie, N_r, jest w przybliżeniu równy stosunkowi pola elipsy, A_e, do pola prostokąta, A_r,

$$\frac{N_e}{N_r} \approx \frac{A_e}{A_r}$$

Ponieważ pole prostokąta $A_r = 4ab$, więc dostajemy stąd

$$A_e \approx 4abN_e/N_r$$

Napisz funkcję `uniform`, która przyjmuje dwie liczby rzeczywiste i zwraca pseudolosową liczbę rzeczywistą z wyznaczonego nimi przedziału. Korzystając z tej funkcji napisz funkcję `ellipse`, która przyjmuje długości półosi elipsy oraz liczbę punktów do wylosowania i zwraca przybliżone pole elipsy obliczone opisaną metodą. Korzystając z tej funkcji napisz program `ellipse`, który wczytuje ze standardowego wejścia długości półosi elipsy oraz liczbę punktów do wylosowania i wypisuje na standardowe wyjście pole elipsy obliczone opisaną metodą oraz dokładnie ze wzoru na pole elipsy.

Przykładowe wykonanie

```
 In: 2 5 1000
Out: 31.08 31.4159
```

Rozwiązanie

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

double uniform(double x1, double x2) {
    return x1 + (x2 - x1) * rand() / RAND_MAX; }

double ellipse(double a, double b, int n) {
    int k = 0;
    for (int i = 0; i < n; ++i) {
        double x = uniform(-a, a), y = uniform(-b, b);
        if (x * x / a / a + y * y / b / b < 1.) {
            ++k; }}
    return 4. * a * b * k / n; }

int main() {
    srand(time(NULL));
    double a, b;
    int n;
    scanf("%lg%lg%i", &a, &b, &n);
    printf("%lg %lg\n", ellipse(a, b, n), 3.1415926 * a * b);
    return 0; }
```

4.2 Zadania laboratoryjne z działu Funkcje

4.2.1 Prime: Czy liczba jest pierwsza

Napisz funkcję `prime`, która przyjmuje nieujemną liczbę całkowitą i zwraca jeden jeśli jest ona pierwsza albo zero w przeciwnym razie. Jeden nie jest liczbą pierwszą. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%s\n", prime(97) ? "true" : "false");
    return 0; }
```

Wykonanie

```
Out: true
```

Rozwiązanie

```
#include <stdio.h>

int prime(int number) {
    int divisor = 2;
    while (divisor < number && number % divisor) {
        ++divisor; }
    return divisor == number; }

int main() {
    printf("%s\n", prime(97) ? "true" : "false");
    return 0; }
```

4.3 Zadania domowe z działu Funkcje na 2 kwietnia

4.3.1 Sign: Znak liczby

Napisz funkcję `sign`, która przyjmuje liczbę całkowitą i zwraca -1, 0, lub 1 jeżeli liczba ta jest odpowiednio ujemna, równa zeru, lub dodatnia. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%i\n", sign(-15));
    return 0; }
```

Wykonanie

```
Out: -1
```

Uwaga Spróbuj napisać funkcję bez użycia instrukcji wyboru, instrukcji warunkowej, ani operatora warunkowego.

4.3.2 Geometric: Ciąg geometryczny

n-ty wyraz ciągu geometrycznego o wyrazie początkowym a_0 i ilorazie q jest równy $a_n = a_0 q^n$. Napisz funkcję `geometric`, która przyjmuje a_0, q oraz n i zwraca a_n. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `math.h`.

Przykładowy program

```
int main() {
    printf("%lg\n", geometric(8., -0.5, 3));
    return 0; }
```

Wykonanie

```
Out: -1
```

4.3.3 Round: Zaokrąglanie z zadaną dokładnością

Zdefiniowana w nagłówku `math.h` funkcja `round` zaokrągla podaną liczbę rzeczywistą do najbliższej liczby całkowitej, na przykład `round(3.14159)` daje w wyniku 3. Korzystając z tej funkcji napisz własną funkcję `round`, która przyjmuje liczbę rzeczywistą oraz liczbę cyfr po przecinku i zwraca podaną liczbę zaokrągloną do żądanej liczby cyfr po przecinku. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `math.h`.

Przykładowy program

```
int main() {
    printf("%lg\n", round(3.14159, 3));
    return 0; }
```

Przykładowe wykonanie

```
Out: 3.142
```

4.3.4 Implication: Implikacja

Implikacja $p \Rightarrow q$ wartości logicznych p i q jest określona tabelą:

p	q	$p \Rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

Napisz funkcję `implication`, która przyjmuje p oraz q i zwraca $p \Rightarrow q$. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%i\n", implication(1, 0));
    return 0; }
```

Wykonanie

```
Out: 0
```

Uwaga Spróbuj napisać funkcję bez użycia instrukcji wyboru, instrukcji warunkowej, ani operatora warunkowego.

4.3.5 Lesser: Mniejsza z dwóch liczb - 1 bitcoin

Napisz funkcję `lesser`, która przyjmuje dwie liczby rzeczywiste i zwraca mniejszą z nich. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%lg\n", lesser(3.12, 2.13));
    return 0; }
```

Wykonanie

```
Out: 2.13
```

Uwaga Spróbuj napisać funkcję bez użycia instrukcji warunkowej.

4.3.6 Bifactorial: Podwójna silnia

Podwójna silnia nieujemnej liczby całkowitej n, oznaczana jako $n!!$, to iloczyn wszystkich dodatnich liczb całkowitych mniejszych lub równych n, o takiej samej parzystości jak n, przy czym $0!! = 1$. Napisz funkcję `bifactorial`, która przyjmuje nieujemną liczbę całkowitą i zwraca jej podwójną silnię. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%i\n", bifactorial(6));
    return 0; }
```

Wykonanie

```
Out: 48
```

4.3.7 Euclid: Algorytm Euklidesa

Największy wspólny dzielnik dwóch dodatnich liczb całkowitych można znaleźć następującą metodą przypisywaną Euklidesowi. Pierwszą liczbę zastępujemy resztą z dzielenia przez drugą i zamieniamy liczby miejscami. Czynność tę powtarzamy aż pierwsza liczba stanie się równa zeru. Wtedy druga jest poszukiwanym największym wspólnym dzielnikiem wyjściowych liczb. Napisz funkcję `euclid`, która przyjmuje dwie dodatnie liczby całkowite i zwraca ich największy wspólny dzielnik znaleziony metodą Euklidesa. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%i\n", euclid(102, 663));
    return 0; }
```

Wykonanie

```
Out: 51
```

4.3.8 Digits: Cyfry dziesiętne

Napisz funkcję `digits`, która przyjmuje nieujemną liczbę całkowitą i zwraca liczbę cyfr w jej zapisie dziesiętnym. Przyjmij, że liczba zero ma zero cyfr. Napisz funkcję `digit`, która przyjmuje nieujemną liczbę całkowitą oraz numer cyfry w jej zapsie dziesiętnym i zwraca tę cyfrę. Cyfry numerujemy od zera dla cyfry jedności. Funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Funkcje nie korzystają z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%i %i\n", digits(2018), digit(2018, 3));
    return 0; }
```

Wykonanie

```
Out: 4 2
```

4.3.9 Power: Potęga całkowita

Napisz funkcję `power`, która przyjmuje niezerową liczbę rzeczywistą x oraz dowolną liczbę całkowitą n i zwraca n-tą potęgę liczby x. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%lg\n", power(-0.5, -3));
    return 0; }
```

Wykonanie

```
Out: -8
```

Uwaga Spróbuj napisać funkcję bez użycia instrukcji warunkowej.

4.3.10 Coin: Rzut oszukaną monetą - 1 bitcoin

Napisz funkcję `coin` symulującą rzut oszukaną monetą. Funkcja przyjmuje prawdopodobieństwo wyrzucenia orła i zwraca jeden jeśli wypadł orzeł albo zero jeśli reszka. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    srand(time(NULL));
    for (int counter = 0; counter < 10; ++counter) {
        printf("%s ", coin(0.2) ? "heads" : "tails"); }
    printf("\n");
    return 0; }
```

Przykładowe wykonanie

```
Out: heads heads tails tails heads tails tails tails tails tails
```

4.3.11 Root: Pierwiastek kwadratowy

Pierwiastek kwadratowy z liczby rzeczywistej x można obliczyć następująco. Jeżeli $x < 1$ to pierwiastek leży między 0 a 1, zaś w przeciwnym razie między 1 a x. Bierzemy środek r odpowiedniego z tych przedziałów. Jeżeli $x < r^2$, to poszukiwany pierwiastek leży w lewej połowie przedziału, zaś w przeciwnym razie leży w prawej połowie. Do dalszych rozważań bierzemy więc odpowiednią połowę, dzielimy ją na pół i tak dalej. Dzięki temu w każdym kroku dwukrotnie zawężamy przedział, w którym leży pierwiastek. Ze względu na skończoną dokładność obliczeń środek któregoś kolejnego przedziału okaże się numerycznie równy jednemu z jego krańców. Oznacza to, że znaleźliśmy wynik z dokładnością maszynową. Napisz funkcję `root`, która przyjmuje nieujemną liczbę rzeczywistą i zwraca jej pierwiastek obliczony opisaną metodą. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%lg %lg\n", root(7.), sqrt(7));
    return 0; }
```

Wykonanie

```
Out: 2.64575 2.64575
```

4.3.12 Pitagoras: Trójki pitagorejskie

Trójką pitagorejską nazywamy każde trzy dodatnie liczby całkowite a, b, c, takie że $a^2+b^2 = c^2$. Trójkę nazywamy pierwotną, jeżeli liczby a, b, c są względnie pierwsze. Napisz program `pitagoras`, który wczytuje ze standardowego wejścia liczbę dodatnią liczbę całkowitą d i wypisuje na standardowe wyjście wszystkie pierwotne trójki pitagorejskie, dla których $c < d$. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 30
Out: 3 4 5
Out: 5 12 13
Out: 8 15 17
Out: 7 24 25
Out: 20 21 29
```

4.3.13 RNG: Generator liczb pseudolosowych

Napisz bezargumentową funkcję `rng`, która zwraca pseudolosową liczbę całkowitą. Funkcja oblicza kolejną liczbę x_{next} na podstawie poprzedniej liczby x_{previous} ze wzoru

$$x_{\text{next}} = (33 * x_{\text{previous}} + 1) \bmod 1024$$

Jest to prosta wersja liniowego generatora kongruentnego. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta ze zmiennych globalnych ani z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    for (int counter = 0; counter < 10; ++counter) {
        printf("%i ", rng()); }
    printf("\n");
    return 0; }
```

Wykonanie

```
Out: 1 34 99 196 325 486 679 904 137 426
```

5 Tablice: 19 marca 2019
Tablice, argumenty wywołania programu

5.1 Przykłady laboratoryjne z działu Tablice

5.1.1 Bubble Sort: Sortowanie bąbelkowe

Bąbelkowe sortowanie tablicy przebiega następująco. Porównujemy pierwszy element z drugim i jeżeli są w niewłaściwej kolejności, to zamieniamy. Następnie porównujemy drugi element z trzecim i tak dalej, do końca tablicy. Jeżeli w takim pojedynczym przebiegu musieliśmy wykonać choćby jedną zamianę, to powtarzamy wszystko od początku. W przeciwnym razie tablica jest już posortowana. Napisz funkcję `bubble_sort`, która przyjmuje tablicę liczb całkowitych oraz jej długość i sortuje tę tablicę bąbelkowo w kolejności niemalejącej. Korzystając z tej funkcji napisz program `bubble_sort`, który wczytuje ze standardowego wejścia 10 liczb całkowitych i wypisuje je na standardowe wyjście w kolejności niemalejącej.

Przykładowe wykonanie

```
 In: 5 7 3 6 3 5 1 9 3 1
Out: 1 1 3 3 3 5 5 6 7 9
```

Rozwiązanie

```
#include <stdio.h>

void bubble_sort(int table[], int size) {
    int unordered = size;
    while (unordered) {
        unordered = 0;
        for (int index = 0; index + 1 < size; ++index) {
            if (table[index + 1] < table[index]) {
                int element = table[index];
                table[index] = table[index + 1];
                table[index + 1] = element;
                unordered = 1; }}}}

int main() {
    int table[10];
    for (int index = 0; index < 10; ++index) {
        scanf("%i", &table[index]); }
    bubble_sort(table, 10);
    for (int index = 0; index < 10; ++index) {
        printf("%i ", table[index]); }
    printf("\n");
    return 0; }
```

5.1.2 Beaufort: Skala Beauforta

Skala Beauforta określona jest następującą tabelą:

Beaufort	km/h
0	0 - 0.5
1	0.5 - 6.5
2	6.5 - 11.5
3	11.5 - 19.5
4	19.5 - 29.5
5	29.5 - 39.5
6	39.5 - 50.5
7	50.5 - 62.5
8	62.5 - 75.5
9	75.5 - 87.5
10	87.5 - 102.5
11	102.5 - 117.5
12	117.5 - ∞

Napisz funkcję `beaufort`, która przyjmuje prędkość wiatru w kilometrach na godzinę i zwraca odpowiadającą jej siłę wiatru w skali Beauforta. Korzystając z tej funkcji napisz program `beaufort`, który przyjmuje jako argument wywołania prędkość wiatru w kilometrach na godzinę i wypisuje na standardowe wyjście siłę wiatru w skali Beauforta.

Przykładowe wykonanie

```
  Linux: ./beaufort 29.5
Windows: beaufort.exe 29.5
Out: 5
```

Rozwiązanie

```
#include <stdio.h>
#include <stdlib.h>

int beaufort(double speed) {
    static const double thresholds[] =
        {0.5, 6.5, 11.5, 19.5, 29.5, 39.5, 50.5, 62.5, 75.5, 87.5, 102.5, 117.5};
    int degree;
    for (degree = 0; degree < 12 && thresholds[degree] <= speed; ++degree);
    return degree; }

int main(int argc, char *argv[]) {
    printf("%i\n", beaufort(atof(argv[1])));
    return 0; }
```

5.2 Zadania domowe z działu Tablice na 9 kwietnia

5.2.1 Days: Długości miesięcy - 1 bitcoin

Napisz program `days`, który przyjmuje jako argumenty wywołania rok oraz numer miesiąca i wypisuje na standardowe wyjście liczbę dni w tym miesiącu z uwzględnieniem lat przestępnych. Program załącza tylko pliki nagłówkowe `stdlib.h` i `stdio.h`.

Przykładowe wykonanie

```
  Linux: ./days 2000 2
Windows: days.exe 2000 2
Out: 29
```

5.2.2 Nominals: Odliczanie kwoty w nominałach

W pewnym państwie emitowane są nominały 1, 2, 5, 10, 20, 50, 100, 200. Napisz program `nominals`, który przyjmuje jako argument wywołania kwotę bez groszy i wypisuje na standardowe wyjście dające ją w sumie nominały, od największego do najmniejszego. Program odlicza zadaną kwotę w możliwie najmniejszej liczbie emitowanych nominałów dysponując nieograniczoną liczbą monet lub banknotów każdego nominału. Program załącza tylko pliki nagłówkowe `stdlib.h` i `stdio.h`.

Przykładowe wykonanie

```
  Linux: ./nominals 739
Windows: nominals.exe 739
Out: 200 200 200 100 20 10 5 2 2
```

5.2.3 Find: Wyszukiwanie elementu

Napisz funkcję `find`, która przyjmuje stałą tablicę liczb całkowitych, jej długość, oraz pojedynczą wartość całkowitą, i zwraca indeks pierwszego wystąpienia tej wartości w tablicy albo długość tablicy jeżeli poszukiwana wartość w niej nie występuje. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table[] = {3, 7, -1, 12, -5, 7, 10};
    printf("%i\n", find(table, 7, 7));
    return 0; }
```

Wykonanie

```
Out: 1
```

5.2.4 Count: Zliczanie elementów

Napisz funkcję `count`, która przyjmuje stałą tablicę liczb całkowitych, jej długość, oraz pojedynczą wartość całkowitą i zwraca liczbę wystąpień tej wartości w zadanej tablicy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table[] = {3, 7, -1, 12, -5, 7, 10};
    printf("%i\n", count(table, 10, 7));
    return 0; }
```

Wykonanie

```
Out: 2
```

5.2.5 Minimum: Element najmniejszy

Napisz funkcję `minimum`, która przyjmuje stałą tablicę liczb całkowitych oraz jej długość i zwraca indeks najmniejszego elementu tej tablicy. Jeżeli takich elementów jest kilka, funkcja zwraca indeks pierwszego z nich. Jeżeli taki element nie istnieje, funkcja zwraca długość tablicy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table[] = {9, -7, 5, 1, 12, 3, -7};
    printf("%i\n", minimum(table, 7));
    return 0; }
```

Wykonanie

```
Out: 1
```

5.2.6 Binary Search: Wyszukiwanie binarne

Chcąc znaleźć daną wartość w tablicy posortowanej niemalejąco można zastosować wyszukiwanie binarne. Rozważamy element leżący w połowie tablicy. Jeżeli jest on większy od poszukiwanej wartości, to na pewno leży ona w lewej połowie tablicy, więc do niej ograniczamy dalsze poszukiwania. W przeciwnym razie prowadzimy je tylko w prawej połowie. Wybraną połowę znów dzielimy na pół i tak dalej. Napisz funkcję `binary_search`, która przyjmuje posortowaną niemalejąco stałą tablicę liczb rzeczywistych, jej długość, oraz pojedynczą wartość rzeczywistą i zwraca indeks pierwszego elementu tablicy większego od podanej wartości. Jeżeli taki element nie istnieje, funkcja zwraca długość tablicy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const double table[] = {-4.3, -1.2, 0.1, 8.2, 11.8};
    printf("%i\n", binary_search(table, 5, 7.6));
    return 0; }
```

Wykonanie

```
Out: 3
```

5.2.7 Matrix: Macierze

Napisz zestaw funkcji działających na tablicach liczb rzeczywistych 2×2 traktowanych jako macierze. Zaimplementuj:

- Funkcję `scan`, która przyjmuje macierz i wczytuje jej elementy ze standardowego wejścia.
- Funkcję `print`, która przyjmuje stałą macierz i wypisuje jej elementy na standardowe wyjście.
- Funkcję `determinant`, która przyjmuje stałą macierz i zwraca jej wyznacznik.
- Funkcję `invert`, która przyjmuje dwie macierze, z czego druga jest stała, i wpisuje do pierwszej macierzową odwrotność drugiej.

Funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Program ten wczytuje ze standardowego wejścia macierz i wypisuje na standardowe wyjście jej wyznacznik oraz odwrotność. Funkcje korzystają tylko z pliku nagłówkowego `stdio.h`.

Przykładowy program

```
int main() {
    double matrix[2][2], inverse[2][2];
    scan(matrix);
    printf("%lg\n", determinant(matrix));
    invert(inverse, matrix);
    print(inverse);
    return 0; }
```

Przykładowe wykonanie

```
 In: 1 2
 In: 3 4

Out: -2

Out: -2 1
Out: 1.5 -0.5
```

5.2.8 Accumulate: Akumulacja

Napisz funkcję `accumulate`, która przyjmuje tablicę liczb rzeczywistych oraz jej długość i do każdego elementu dodaje sumę wszystkich go poprzedzających. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    double table[] = {3.1, 2.7, -0.5, 0.1, 4.3};
    accumulate(table, 5);
    for (int index = 0; index < 5;) {
        printf("%lg", table[index++]); }
    printf("\n");
    return 0; }
```

Wykonanie

```
Out: 3.1 5.8 5.3 5.4 9.7
```

5.2.9 Reverse: Odwracanie kolejności elementów

Napisz funkcję `reverse`, która przyjmuje tablicę liczb całkowitych oraz jej długość i odwraca kolejność elementów tej tablicy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table[] = {7, -1, 12, 3, 10, 7, -5};
    reverse(table, 7);
    for (int index = 0; index < 7;) {
        printf("%i ", table[index++]); }
    printf("\n");
    return 0; }
```

Wykonanie

```
Out: -5 7 10 3 12 -1 7
```

5.2.10 Shuffle: Tasowanie elementów

Napisz funkcję `shuffle`, która przyjmuje tablicę liczb całkowitych oraz jej długość i losowo przestawia elementy tej tablicy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    srand(time(NULL));
    int table[] = {7, -1, 12, 3, 10, 7, -5};
    shuffle(table, 7);
    for (int index = 0; index < 7;) {
        printf("%i ", table[index++]); }
    printf("\n");
    return 0; }
```

Przykładowe wykonanie

```
Out: 3 7 -5 -1 12 10 7
```

5.2.11 Selection Sort: Sortowanie przez wybór - 1 bitcoin

Sortowanie tablicy w kolejności niemalejącej metodą wybierania przebiega następująco. Znajdujemy w tablicy pierwszy od lewej element najmniejszy. Jeśli nie jest on pierwszym elementem tablicy, to go z nim zamieniamy. Następnie powtarzamy te czynności dla tablicy bez pierwszego elementu i tak dalej. Napisz funkcję `selection_sort`, która przyjmuje tablicę liczb całkowitych oraz jej długość i sortuje tę tablicę niemalejąco metodą wybierania. Oprócz tego funkcja wypisuje na standardowe wyjście w kolejnych liniach elementy tablicy po każdej dokonanej zamianie. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `stdio.h`.

Przykładowy program

```
int main() {
    int table[] = {3, 7, -1, 12, -5, 7, 10};
    selection_sort(table, 7);
    return 0; }
```

Wykonanie

```
Out: -5 7 -1 12 3 7 10
Out: -5 -1 7 12 3 7 10
Out: -5 -1 3 12 7 7 10
Out: -5 -1 3 7 12 7 10
Out: -5 -1 3 7 7 12 10
Out: -5 -1 3 7 7 10 12
```

6 Pliki: 26 marca 2019
Znaki, pliki

6.1 Przykłady laboratoryjne z działu Pliki

6.1.1 Capital: Zamiana małych liter na wielkie

Napisz program `capital`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego zamieniając małe litery na wielkie.

Przykładowy plik wejściowy input.txt

```
Ala ma 15 lat!
Czy Ala ma 17 lat?
```

Przykładowe wywołanie

```
  Linux: ./capital input.txt output.txt
Windows: capital.exe input.txt output.txt
```

Przykładowy plik wyjściowy output.txt

```
ALA MA 15 LAT!
CZY ALA MA 17 LAT?
```

Rozwiązanie

```
#include <stdio.h>

char upper(char character) {
    return 'a' <= character && character <= 'z' ? 'A' + character - 'a': character; }

int main(int argc, char *argv[]) {
    FILE *input = fopen(argv[1], "r");
    FILE *output = fopen(argv[2], "w");
    char character;
    while (fscanf(input, "%c", &character) == 1) {
        fprintf(output, "%c", upper(character)); }
    fclose(input);
    fclose(output);
    return 0; }
```

6.2 Zadania laboratoryjne z działu Pliki

6.2.1 Is: Rodzaje znaków

Napisz program `is`, który wczytuje ze standardowego wejścia jeden znak bez opuszczania znaków białych i wypisuje na standardowe wyjście `digit`, `upper`, `lower` albo `other` jeśli jest on odpowiednio cyfrą, wielką literą, małą literą albo innym znakiem. Program załącza tylko pliki nagłówkowe `ctype.h` i `stdio.h`.

Przykładowe wykonanie

```
 In: a
Out: lower
```

Rozwiązanie

```
#include <ctype.h>
#include <stdio.h>

int main() {
    char character = fgetc(stdin);
    printf("%s\n", isdigit(character) ? "digit" :
                   isupper(character) ? "upper" :
                   islower(character) ? "lower" : "other");
    return 0; }
```

6.3 Zadania domowe z działu Pliki na 16 kwietnia

6.3.1 Char: Znak o zadanym kodzie

Napisz program `char`, który wczytuje ze standardoweego wejścia kod znaku i wypisuje ten znak na standardowe wyjście. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
 In: 82
Out: R
```

6.3.2 ASCII: Tabela kodów ASCII

Napisz program `ascii`, który wypisuje na standardowe wyjście wszystkie znaki od wykrzyknika do litery zet wraz z ich kodami. Program załącza tylko plik nagłówkowy `stdio.h`.

Wykonanie

```
Out: 33 !    43 +    53 5    63 ?    73 I    83 S    93 ]   103 g   113 q
Out: 34 "    44 ’    54 6    64 @    74 J    84 T    94 ^   104 h   114 r
Out: 35 #    45 -    55 7    65 A    75 K    85 U    95 _   105 i   115 s
Out: 36 $    46 .    56 8    66 B    76 L    86 V    96 ‘   106 j   116 t
Out: 37 %    47 /    57 9    67 C    77 M    87 W    97 a   107 k   117 u
Out: 38 &    48 0    58 :    68 D    78 N    88 X    98 b   108 l   118 v
Out: 39 ’    49 1    59 ;    69 E    79 O    89 Y    99 c   109 m   119 w
Out: 40 (    50 2    60 <    70 F    80 P    90 Z   100 d   110 n   120 x
Out: 41 )    51 3    61 =    71 G    81 Q    91 [   101 e   111 o   121 y
Out: 42 *    52 4    62 >    72 H    82 R    92 \   102 f   112 p   122 z
```

6.3.3 WC: Zliczanie znaków, wyrazów i linii

Napisz program `wc`, który przyjmuje jako argument wywołania nazwę pliku tekstowego i wypisuje na standardowe wyjście liczbę zawartych w tym pliku linii, słów, oraz znaków łącznie ze znakami białymi. Przyjmij dla uproszczenia, że każda linia kończy się znakiem `LF`. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowy plik wejściowy input.txt

```
ala ma kota[LF]
lorem ipsum dolor sit amet[LF]
litwo ojczyzno moja[LF]
[EOF]
```

Przykładowe wykonanie

```
  Linux: ./wc input.txt
Windows: ws.exe input.txt
Out: 4 11 60
```

6.3.4 Cat: Łączenie plików tekstowych

Napisz program `cat`, który przyjmuje jako argumenty wywołania nazwy dowolnej liczby plików tekstowych i wypisuje zawartości kolejnych plików na standardowe wyjście nie wstawiając między nimi żadnych dodatkowych znaków. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowy plik wejściowy input1.txt

```
to samo[EOF]
```

Przykładowy plik wejściowy input2.txt

```
lot do Londynu[EOF]
```

Przykładowe wykonanie

```
  Linux: ./cat input1.txt input2.txt
Windows: cat.exe input1.txt input2.txt
Out: to samolot do Londynu[EOF]
```

6.3.5 Numerator: Numeracja linii

Napisz program `numerator`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego dopisując na początku każdej linii jej kolejny numer liczony od jednego. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowy plik wejściowy input.txt

```
Ala ma kota.

To kot Ali.
```

Przykładowe wywołanie

```
  Linux: ./numerator input.txt output.txt
Windows: numerator.exe input.txt output.txt
```

Przykładowy plik wyjściowy output.txt

```
1 Ala ma kota.
2
3 To kot Ali.
```

6.3.6 Separator: Odstępy między wyrazami

Napisz program `separator`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego zamieniając każdą sekwencję spacji i tabulatorów na pojedynczą spację. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowy plik wejściowy input.txt

```
Ala     ma  kota.
```

Przykładowe wywołanie

```
  Linux: ./separator input.txt output.txt
Windows: separator.exe input.txt output.txt
```

Przykładowy plik wyjściowy output.txt

```
Ala ma kota.
```

6.3.7 Caesar: Szyfr Cezara - 1 bitcoin

Kodowanie wiadomości tekstowej szyfrem Cezara z przesunięciem n polega na zastąpieniu każdej litery inną, znajdującą się w alfabecie n pozycji dalej. Przesunięcie n może być dodatnie lub ujemne, przy czym przyjmujemy, że za literą z wypada litera a, zaś przed literą a wypada litera z. Małe litery są kodowane jako małe, a duże jako duże. Inne znaki nie są szyfrowane. Napisz program `caesar`, który przyjmuje jako argumenty wywołania przesunięcie oraz nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego kodując ją szyfrem Cezara z zadanym przesunięciem. Program załącza tylko pliki nagłówkowe `stdio.h` i `stdlib.h`.

Przykładowy plik wejściowy input.txt

```
Bwf Dbftbs!
Wfoj, wjej, wjdj!
```

Przykładowe wykonanie

```
  Linux: ./caesar -1 input.txt output.txt
Windows: caesar.exe -1 input.txt output.txt
```

Przykładowy plik wyjściowy output.txt

```
Ave Caesar!
Veni, vidi, vici!
```

6.3.8 LF: Znaki końca linii

W systemie Linux koniec linii oznacza się pojedynczym znakiem *line feed*, czyli `\n`, zaś w systemie Windows używa się pary *carriage return - line feed*, czyli `\r\n`. Napisz program `lf`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego zastępując pojedyncze znaki `\n` parami `\r\n` i na odwrót. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowy plik wejściowy input.txt

```
Ala ma kota.[CR][LF]
To kot Ali.[LF]
[EOF]
```

Przykładowe wykonanie

```
  Linux: ./lf input.txt output.txt
Windows: lf.exe input.txt output.txt
```

Przykładowy plik wyjściowy output.txt

```
Ala ma kota.[LF]
To kot Ali.[CR][LF]
[EOF]
```

6.3.9 Camel: Kamelizacja wyrazów

Napisz program `camel`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego zamieniając pierwszą literę każdego słowa na wielką, a następne litery na małe. Pozostałe znaki program przepisuje bez zmian. Program załącza tylko pliki nagłówkowe `ctype.h` i `stdio.h`.

Przykładowy plik wejściowy input.txt

```
AlA mA 15 LAT!
czy ala ma 17 lat?
```

Przykładowe wywołanie

```
  Linux: ./camel input.txt output.txt
Windows: camel.exe input.txt output.txt
```

Przykładowy plik wyjściowy output.txt

```
Ala Ma 15 Lat!
Czy Ala Ma 17 Lat?
```

6.3.10 Letters: Zliczanie liter - 1 bitcoin

Napisz program `letters`, który przyjmuje jako argument wywołania nazwę pliku tekstowego i wypisuje na standardowe wyjście liczby wystąpień kolejnych liter w tym pliku. Program nie rozróżnia wielkich i małych liter oraz pomija wszelkie inne znaki. Program załącza tylko pliki nagłówkowe `ctype.h` i `stdio.h`.

Przykładowy plik wejściowy input.txt

```
lorem ipsum dolor sit amet
```

Przykładowe wywołanie

```
  Linux: ./letters input.txt
Windows: letters.exe input.txt
Out: 1 0 0 1 2 0 0 0 2 0 0 2 3 0 3 1 0 2 2 2 1 0 0 0 0 0
```

7 Napisy: 2 kwietnia 2019
Napisy

7.1 Przykłady laboratoryjne z działu Napisy

7.1.1 Trailer: Usuwanie spacji z końca linii

Napisz program `trailer`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego opuszczając znaki białe znajdujące się na końcach linii. Przyjmij, że linia ma najwyżej 255 znaków łącznie ze znakiem końca wiersza.

Rozwiązanie

```
#include <stdio.h>

void trail(char line[]) {
    int end = 0;
    for (int index = 0; line[index]; ++index) {
        if (line[index] != ' ' && line[index] != '\t' && line[index] != '\n') {
            end = 1 + index; }}
    line[end] = '\0'; }

int main(int argc, char *argv[]) {
    FILE *input = fopen(argv[1], "r");
    FILE *output = fopen(argv[2], "w");
    char line[256];
    while (fgets(line, 256, input)) {
        trail(line);
        fprintf(output, "%s\n", line); }
    fclose(output);
    fclose(input);
    return 0; }
```

7.2 Zadania laboratoryjne z działu Napisy

7.2.1 Length: Długość napisu

Napisz funkcję `length`, która przyjmuje stały napis i zwraca jego długość, czyli liczbę znaków poprzedzających bajt zerowy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%i\n", length("ala ma kota"));
    return 0; }
```

Wykonanie

```
Out: 11
```

Rozwiązanie

```
int length(const char string[]) {
    int size = 0;
    while (string[size]) {
        ++size; }
    return size; }

#include <stdio.h>

int main() {
    printf("%i\n", length("ala ma kota"));
    return 0; }
```

7.3 Zadania domowe z działu Napisy na 23 kwietnia

7.3.1 Integer: Konwersja łańcucha do liczby całkowitej

Napisz funkcję `integer`, która przyjmuje stały napis z dziesiętnym zapisem nieujemnej liczby całkowitej i zwraca tę liczbę. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%i\n", integer("2018"));
    return 0; }
```

Wykonanie

```
Out: 2018
```

7.3.2 Compare: Porównywanie napisów - 1 bitcoin

Napisz funkcję `compare`, która przyjmuje dwa stałe napisy i zwraca jeden jeśli pierwszy wypada w kolejności alfabetycznej przed drugim albo zero w przeciwnym razie. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%s\n", compare("eisenhower", "einstein") ? "true" : "false");
    return 0; }
```

Wykonanie

```
Out: false
```

7.3.3 Search: Wyszukiwanie napisów w tekście

Napisz funkcję `search`, która przyjmuje dwa stałe napisy i zwraca indeks pierwszego wystąpienia drugiego napisu w pierwszym. Jeżeli drugi napis w pierwszym nie występuje, funkcja zwraca długość pierwszego napisu. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    printf("%i\n", search("to niesamowity samolot", "samo"));
    return 0; }
```

Wykonanie

```
Out: 6
```

7.3.4 Erase: Usuwanie wycinka napisu

Napisz funkcję `erase`, która przyjmuje napis, indeks pierwszego znaku oraz długość pewnego wycinka tego napisu, i usuwa z napisu ten wycinek. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    char string[] = "this is not the best solution";
    erase(string, 8, 3);
    printf("%s\n", string);
    return 0; }
```

Wykonanie

```
Out: this is the best solution
```

7.3.5 Palindrom: Wykrywanie palindromów

Palindrom to tekst, który czytany po literze od końca jest taki sam, jak czytany od początku. Nie liczy się przy tym wielkość liter ani żadne znaki poza literami. Napisz program `palindrom`, który przyjmuje jako argument wywołania dowolny napis i wypisuje na standardowe wyjście `true` jeśli jest on palindromem albo `false` w przeciwnym razie. Program załącza tylko pliki nagłówkowe `ctype.h` i `stdio.h`.

Przykładowe wykonanie

```
  Linux: ./palindrom "Ile Roman ladny dyndal na moreli?"
Windows: palindrom.exe "Ile Roman ladny dyndal na moreli?"
Out: true
```

7.3.6 Grep: Linie zawierające podane słowo - 1 bitcoin

Napisz program `grep`, który przyjmuje jako argument wywołania napis oraz nazwę pliku tekstowego i wypisuje na standardowe wyjście wszystkie linie tego pliku zawierające podany napis. Przyjmij, że każda linia ma najwyżej 255 znaków łącznie ze znakiem końca wiersza. Program załącza tylko plik nagłówkowy `stdio.h`

Przykładowy plik wejściowy input.txt

```
raz dwa trzy
raz trzy
sto dwanaście
DWA
```

Przykładowe wywołanie

```
  Linux: ./grep dwa input.txt
Windows: grep.exe dwa input.txt
```

Przykładowe standardowe wyjście

```
raz dwa trzy
sto dwanaście
```

7.3.7 Blanker: Usuwanie pustych linii

Napisz program `blanker`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego pomijając linie puste oraz zawierające tylko znaki białe. Przyjmij, że każda linia ma najwyżej 255 znaków łącznie ze znakiem końca wiersza. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowy plik wejściowy input.txt

```
lorem ipsum

dolor sit


amet
consectetur adipisci elit
```

Przykładowe wywołanie

```
  Linux: ./blanker input.txt output.txt
Windows: blanker.exe input.txt output.txt
```

Przykładowy plik wyjściowy output.txt

```
lorem ipsum
dolor sit
amet
consectetur adipisci elit
```

8 Sprawdzian: 9 kwietnia 2019

9 Wskaźniki: 16 kwietnia 2019
Adresy pojedynczych zmiennych

9.1 Przykłady laboratoryjne z działu Wskaźniki

9.1.1 Cartesian: Współrzędne biegunowe i kartezjańskie

Napisz funkcję `cartesian`, która przyjmuje współrzędne biegunowe punktu na płaszczyznie, r, φ, z kątem w radianach, oraz adresy dwóch zmiennych, i wpisuje do nich współrzędne kartezjańskie tego samego punktu, x, y. Korzystając z tej funkcji napisz program `cartesian`, który wczytuje ze standardowego wejścia współrzędne biegunowe z kątem w stopniach i wypisuje na standardowe wyjście odpowiadające im współrzędne kartezjańskie.

Przykładowe wykonanie

```
 In: 1 60
Out: 0.5 0.866025
```

Rozwiązanie

```
#include <math.h>
#include <stdio.h>

#define PI 3.14159265358979323846

void cartesian(double *x, double *y, double r, double phi) {
    *x = r * cos(phi);
    *y = r * sin(phi); }

int main() {
    double x, y, r, phi;
    scanf("%lg%lg", &r, &phi);
    cartesian(&x, &y, r, PI * phi / 180);
    printf("%lg %lg\n", x, y);
    return 0; }
```

9.2 Zadania laboratoryjne z działu Wskaźniki

9.2.1 Clock: Położenie wskazówek zegara

Napisz funkcję `clock`, która przyjmuje całkowite liczby godzin i minut, na przykład 13 i 23 dla godziny trzynastej dwadzieścia trzy, oraz adresy dwóch zmiennych rzeczywistych i wpisuje do nich odpowiadające podanej godzinie kąty wychylenia wskazówek zegara wyrażone w stopniach i liczone zgodnie z ruchem wskazówek zegara od położenia pionowo w górę. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    double small, large;
    clock(&small, &large, 13, 23);
    printf("%lg %lg\n", small, large);
    return 0; }
```

Wykonanie

```
Out: 41.5 138
```

Rozwiązanie

```
void clock(double *small, double *large, int hour, int minute) {
    int minutes = 60 * hour + minute;
    *small = minutes % 720 / 2.;
    *large = minutes % 60 * 6.; }

#include <stdio.h>

int main() {
    double small, large;
    clock(&small, &large, 13, 23);
    printf("%lg %lg\n", small, large);
    return 0; }
```

9.3 Zadania domowe z działu Wskaźniki na 7 maja

9.3.1 DMS: Stopnie, minuty, sekundy

W geografii kąty często wyraża się podając całkowite liczby stopni, minut i sekund. Napisz funkcję `dms`, która przyjmuje kąt w stopniach jako liczbę rzeczywistą oraz adresy trzech zmiennych całkowitych i wpisuje do nich całkowite liczby stopni, minut oraz sekund. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int degrees, minutes, seconds;
    dms(&degrees, &minutes, &seconds, 123.37);
    printf("%i %i %i\n", degrees, minutes, seconds);
    return 0; }
```

Wykonanie

```
Out: 123 22 12
```

9.3.2 Fraction: Część całkowita i ułamkowa

Napisz funkcję `fraction`, która przyjmuje nieujemną liczbę rzeczywistą oraz adresy dwóch zmiennych rzeczywistych i wpisuje do nich część całkowitą oraz ułamkową podanej liczby. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `math.h`.

Przykładowy program

```
int main() {
    double integral, fractional;
    fraction(&integral, &fractional, 3.141593);
    printf("%lg %lg\n", integral, fractional);
    return 0; }
```

Wykonanie

```
Out: 3 0.141593
```

9.3.3 Quadratic: Równanie kwadratowe - 1 bitcoin

Napisz funkcję `quadratic` rozwiązującą równanie kwadratowe $ax^2+bx+c=0$. Funkcja przyjmuje wartości parametrów a, b, c oraz adresy dwóch zmiennych rzeczywistych i zwraca wyróżnik równania. Jeżeli równaie posiada rozwiązania, funkcja wpisuje je pod wskazane adresy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `math.h`.

Przykładowy program

```
int main() {
    double delta, x1, x2;
    delta = quadratic(&x1, &x2, 1.2, 3.7, -4);
    printf("%lg\n", delta);
    printf("%lg %lg\n", x1, x2);
    return 0; }
```

Wykonanie

```
Out: 32.89
Out: -3.93124 0.847908
```

9.3.4 Exchange: Zamiana wartości jednej zmiennej

Napisz funkcję `exchange`, która przyjmuje adres zmiennej rzeczywistej oraz liczbę rzeczywistą, wpisuje liczbę do zmiennej i zwraca poprzednią wartość zmiennej. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    double a = 2.71828, b = exchange(&a, 3.14159);
    printf("%lg %lg\n", a, b);
    return 0; }
```

Wykonanie

```
Out: 3.14159 2.71828
```

9.3.5 Swap: Zamiana wartości dwóch zmiennych

Napisz funkcję `swap`, która przyjmuje adresy dwóch zmiennych rzeczywistych i zamienia ich wartości. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    double a = -3.2, b = 17.1;
    swap(&a, &b);
    printf("%lg %lg\n", a, b);
    return 0; }
```

Wykonanie

```
Out: 17.1 -3.2
```

9.3.6 Choose: Wybór zmiennej

Napisz funkcję `choose`, która przyjmuje wartość całkowitą oraz adresy dwóch zmiennych rzeczywistych i zwraca adres pierwszej albo drugiej z nich jeżeli przekazana wartość całkowita jest odpowiednio niezerowa albo zerowa. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    double a = 2.3, b = 3.2;
    choose(a, b, a > b) = 10.9;
    printf("%lg %lg\n", a, b);
    return 0; }
```

Wykonanie

```
Out: 2.3 10.9
```

9.3.7 Lesser: Zmienna o mniejszej wartości

Napisz funkcję `lesser`, która przyjmuje adresy dwóch zmiennych całkowitych i zwraca adres tej z nich, która ma mniejszą wartość. Jeśli zmienne mają jednakowe wartości, funkcja zwraca wskaźnik pusty. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `stddef.h`.

Przykładowy program

```
int main() {
    int a = 11, b = 5;
    *lesser(&a, &b) = -7;
    printf("%i %i\n", a, b);
    return 0; }
```

Wykonanie

```
Out: 11 -7
```

9.3.8 RNG: Generator liczb pseudolosowych - 1 bitcoin

Napisz funkcję `rng` generującą pseudolosowe liczby całkowite. Funkcja oblicza kolejną liczbę x_{next} na podstawie poprzedniej x_{previous} według wzoru

$$x_{\text{next}} = (33 * x_{\text{previous}} + 1) \bmod 1024$$

Funkcja nie przyjmuje żadnych argumentów i zwraca adres statycznej zmiennej zadeklarowanej wewnątrz funkcji. Każde wywołanie funkcji traktuje wartość tej zmiennej jako x_{previous} i nadaje jej wartość x_{next}. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    *rng() = 7;
    for (int counter = 0; counter < 10; ++counter) {
        printf("%i ", *rng()); }
    printf("\n"); }
```

Wykonanie

```
Out: 232 489 778 75 428 813 206 655 112 625
```

10 Arytmetyka: 7 maja 2019
Arytmetyka wskaźników

10.1 Przykłady laboratoryjne z działu Arytmetyka

10.1.1 Bubble Sort: Sortowanie bąbelkowe

Napisz funkcję `bubble_sort`, która przyjmuje adres początku i adres końca wycinka tablicy liczb całkowitych i sortuje ten wycinek bąbelkowo w kolejności niemalejącej. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej.

Przykładowy program

```
int main() {
    int table[10] = {20, -1, 13, 5, -1, 7, 5, 2, -5, 9};
    bubble_sort(table + 3, table + 8);
    for (int *pointer = table; pointer < table + 10;) {
        printf("%i ", *pointer++); }
    printf("\n");
    return 0; }
```

Wykonanie

```
Out: 20 -1 13 -1 2 5 5 7 -5 9
```

Rozwiązanie

```
void swap(int *pointer1, int *pointer2) {
    int element = *pointer1;
    *pointer1 = *pointer2;
    *pointer2 = element; }

void bubble_sort(int *begin, int *end) {
    int unordered = begin < end;
    while (unordered) {
        unordered = 0;
        for (int *pointer = begin; pointer + 1 < end; ++pointer) {
            if (pointer[1] < pointer[0]) {
                swap(pointer, pointer + 1);
                unordered = 1; }}}}

#include <stdio.h>

int main() {
    int table[10] = {20, -1, 13, 5, -1, 7, 5, 2, -5, 9};
    bubble_sort(table + 3, table + 8);
    for (int *pointer = table; pointer < table + 10; ++pointer) {
        printf("%i ", *pointer); }
    printf("\n");
    return 0; }
```

10.2 Zadania laboratoryjne z działu Arytmetyka

10.2.1 Count: Zliczanie elementów

Napisz funkcję `count`, która przyjmuje adres początku oraz adres końca wycinka tablicy stałych całkowitych oraz pojedynczą wartość całkowitą i zwraca liczbę wystąpień tej wartości w zadanym wycinku. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów tablicy i nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table[] = {3, 7, -1, 12, -5, -1, 10};
    printf("%i\n", count(table + 2, table + 7, -1));
    return 0; }
```

Wykonanie

```
Out: 2
```

Rozwiązanie

```
int count(const int *begin, const int *end, int element) {
    int count = 0;
    while (begin < end) {
        if (*begin++ == element) {
            ++count; }}
    return count; }

#include <stdio.h>

int main() {
    const int table[] = {3, 7, -1, 12, -5, -1, 10};
    printf("%i\n", count(table + 2, table + 7, -1));
    return 0; }
```

10.3 Zadania domowe z działu Arytmetyka na 4 czerwca

10.3.1 Reverse: Odwracanie kolejności elementów

Napisz funkcję `reverse`, która przyjmuje adres początku oraz adres końca wycinka tablicy liczb całkowitych i odwraca kolejność elementów tego wycinka. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów tablicy i nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table[] = {7, -1, 12, 3, 10, 5, -5};
    reverse(table, table + 6);
    for (int *pointer = table; pointer < table + 7;) {
        printf("%i ", *pointer++); }
    printf("\n");
    return 0; }
```

Przykładowe wykonanie

```
Out: 5 10 3 12 -1 7 -5
```

10.3.2 Accumulate: Akumulacja - 1 bitcoin

Napisz funkcję `accumulate`, która przyjmuje adres początku i adres końca wycinka tablicy liczb rzeczywistych i do każdego elementu tego wycinka dodaje sumę wszystkich go poprzedzających. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów tablicy i nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    double table[] = {3.1, 2.7, -0.5, 0.1, 4.3};
    accumulate(table, table + 3);
    for (double *pointer = table; pointer < table + 5;) {
        printf("%lg ", *pointer++); }
    printf("\n");
    return 0; }
```

Wykonanie

```
Out: 3.1 5.8 5.3 0.1 4.3
```

10.3.3 Selection Sort: Sortowanie przez wybór

Sortowanie tablicy w kolejności niemalejącej przez wybór przebiega następująco. Znajdujemy w tablicy pierwszy element najmniejszy i zamieniamy go miejscami z pierwszym elementem tablicy. Następnie powtarzamy te czynności dla tablicy bez pierwszego elementu i tak dalej. Napisz funkcję `selection_sort`, która przyjmuje adres początku i adres końca wycinka tablicy liczb całkowitych i sortuje ten wycinek niemalejąco przez wybór. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów tablicy i nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int table[] = {3, 15, -2, 35, 0, -53, 20};
    selection_sort(table, table + 7);
    for (int *pointer = table; pointer < table + 7;) {
        printf("%i ", *pointer++); }
    printf("\n");
    return 0; }
```

Wykonanie

```
Out: -53 -2 0 3 15 20 35
```

10.3.4 Find: Wyszukiwanie elementu - 1 bitcoin

Napisz funkcję `find`, która przyjmuje adres początku i adres końca wycinka tablicy stałych całkowitych oraz pojedynczą wartość całkowitą i zwraca adres pierwszego wystąpienia tej wartości w wycinku albo adres końca wycinka jeżeli ta wartość w nim nie występuje. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów tablicy i nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table[] = {1, 7, -5, 2, 8, -5, 3};
    printf("%li\n", find(table + 3, table + 7, -5) - table);
    return 0; }
```

Wykonanie

```
Out: 5
```

10.3.5 Minimum: Element najmniejszy

Napisz funkcję `minimum`, która przyjmuje adres początku oraz końca wycinka tablicy liczb całkowitych i zwrca adres najmniejszego elementu tego wycinka. Jeżeli takich elementów jest kilka, funkcja zwraca adres pierwszego z nich. Jeżeli taki element nie istnieje, funkcja zwraca adres końca wycinka. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie używa indeksów tablicy i nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const int table[] = {-15, 7, 5, 1, 12, -3, 8};
    printf("%li\n", minimum(table + 1, table + 7) - table);
    return 0; }
```

Wykonanie

```
Out: 5
```

10.3.6 Endian: Little endian i big endian

Pamięć komputera to ogromna tablica bajtów, z których każdy ma swój kolejny indeks zwany też adresem. Liczby całkowite są zapisywane w tej pamięci w dwójkowym układzie pozycyjnym. Osiem pierwszych cyfr zapisu dwójkowego licząc od prawej tworzy najmniej znaczący bajt liczby i tak dalej aż do najbardziej znaczącego bajtu. Bajty te mogą być zapisane w pamięci tak, by najmniej znaczący znajdował się pod najmniejszym adresem zaś najbardziej znaczący pod największym albo odwrotnie. Te dwa sposoby nazywamy odpowiednio `little endian` oraz `big endian`. Różne procesory korzystają z różnych zapisów. Napisz program `endian`, który sprawdza doświadczalnie, czy komputer używa zapisu `little endian` czy `big endian`, i wypisuje na standardowe wyjście odpowiednio `little` lub `big`. Program załącza tylko plik nagłówkowy `stdio.h`.

Przykładowe wykonanie

```
Out: little
```

Uwaga Aby doświadczalnie sprawdzić, jakiego zapisu używa dany komputer, tworzymy dwubajtową zmienną całkowitą bez znaku i wpisujemy do niej liczbę złożoną z dwóch różniących się bajtów. Następnie odczytujemy bajt o mniejszym adresie, równym adresowi całej zmiennej, i sprawdzamy, czy jest on równy mniej, czy bardziej znaczącemu bajtowi wpisanej liczby.

11 Alokacja: 14 maja 2019 Dynamiczna alokacja pamięci

11.1 Przykłady laboratoryjne z działu Alokacja

11.1.1 Numbers: Wczytywanie liczb

Napisz funkcję `numbers`, która przyjmuje adres zmiennej całkowitej oraz adres pliku, czyta z tego pliku liczby rzeczywiste do napotkania końca pliku i umieszcza je w dynamicznie utworzonej tablicy, pod przekazany adres wpisuje liczbę wczytanych liczb i zwraca adres wynikowej tablicy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej.

Przykładowy program

```
int main(int argc, char *argv[]) {
    int size;
    FILE *input = fopen(argv[1], "r");
    double *table = numbers(&size, input);
    fclose(input);
    for (int index = 0; index < size; ++index) {
        printf("%lg ", table[index]); }
    printf("\n");
    free(table);
    return 0; }
```

Przykładowy plik wejściowy input.txt

```
3.14 -5 128.3
```

Przykładowe wykonanie

```
  Linux: ./numbers input.txt
Windows: numbers.exe input.txt
Out: 3.14 -5 128.3
```

Rozwiązanie

```
#include <stdio.h>
#include <stdlib.h>

double *numbers(int *size, FILE *input) {
    *size = 0;
    int capacity = 0;
    double *table = NULL;
    for (double element; fscanf(input, "%lg", &element) == 1; ++*size) {
        if (*size == capacity) {
            capacity = (capacity ? 2 * capacity : 1);
            table = realloc(table, capacity * sizeof(double)); }
        table[*size] = element; }
    return table; }

int main(int argc, char *argv[]) {
    int size;
    FILE *input = fopen(argv[1], "r");
    double *table = numbers(&size, input);
    fclose(input);
    for (double *pointer = table; pointer < table + size; ++pointer) {
        printf("%lg ", *pointer); }
    printf("\n");
    free(table);
    return 0; }
```

11.2 Zadania laboratoryjne z działu Alokacja

11.2.1 Password: Losowe hasło

Napisz funkcję `password`, która przyjmuje długość hasła, tworzy tej długości hasło złożone z losowych cyfr i małych liter, umieszcza je w dynamicznie utworzonym napisie, i zwraca adres tego napisu. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main(int argc, char *argv[]) {
    srand(time(NULL));
    char *string = password(7);
    printf("%s\n", string);
    free(string);
    return 0; }
```

Przykładowe wykonanie

```
Out: 5zy4hsu
```

Rozwiązanie

```
#include <stdlib.h>

char *password(int size) {
    char *string = malloc(size + 1);
    for (int index = 0; index < size; ++index) {
        int character = rand() % 36;
        string[index] = character < 10 ? '0' + character : 'a' + character - 10; }
    string[size] = '\0';
    return string; }

#include <stdio.h>
#include <time.h>

int main(int argc, char *argv[]) {
    srand(time(NULL));
    char *string = password(7);
    printf("%s\n", string);
    free(string);
    return 0; }
```

11.3 Zadania domowe z działu Alokacja na 11 czerwca

11.3.1 Eratostenes: Sito Eratostenesa - 1 bitcoin

Wszystkie liczby pierwsze mniejsze od dodatniej liczby całkowitej n można wyznaczyć następującą metodą zwaną sitem Eratostenesa. Spośród liczb większych od 1 i mniejszych od n wykreślamy wszystkie wielokrotności 2 poczynając od 4, następnie wszystkie wielokrotności 3 poczynając od 6 i tak dalej. Pozostałe na koniec niewykreślone liczby to wszystkie liczby pierwsze mniejsze od n. Napisz program `eratostenes`, który przyjmuje jako argument wywołania dodatnią liczbę całkowitą i wypisuje na standardowe wyjście wszystkie mniejsze od niej liczby pierwsze w kolejności rosnącej. Program załącza tylko pliki nagłówkowe `stdio.h` i `stdlib.h`.

Przykładowe wykonanie

```
  Linux: ./eratostenes 100
Windows: eratostenes.exe 100
Out: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Uwaga Korzystając z następujących obserwacji można przyspieszyć działanie programu i zmniejszyć zużycie pamięci:

1. Wielokrotności każdej liczby złożonej zostały już wykreślone wcześniej jako wielokrotności jej dzielników. Można więc pominąć wykreślanie wielokrotności liczb już wykreślonych.
2. Wielokrotności liczby k od drugiej do $(k-1)$-szej zostały już wykreślone wcześniej jako wielokrotności liczb 2,..., $(k-1)$. Wykreślanie wielokrotności każdej liczby można więc zacząć od jej kwadratu.
3. Jeżeli kwadrat danej liczby jest większy lub równy n, to kwadraty wszystkich kolejnych będą większe od n. Oznacza to, że wszystkie liczby złożone mniejsze od n zostały już wykreślone i całą procedurę można zakończyć.
4. Liczby parzyste większe od dwóch nie są pierwsze. Można je zatem całkowicie pominąć i nawet nie pamiętać w tablicy.
5. Do przechowania informacji o wykreśleniu liczby wystarczy jeden bit.

11.3.2 Combination: Losowa kombinacja

Napisz funkcję `combination`, która przyjmuje stałą tablicę nieujemnych liczb całkowitych oraz jej długość, i tworzy dynamicznie tablicę zawierającą w losowej kolejności tyle zer, jedynek, i tak dalej, jakie są wartości kolejnych komórek tablicy przekazanej jako argument. Funkcja zwraca adres utworzonej tablicy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    srand(time(NULL));
    const int counts[] = {3, 0, 5, 2};
    int *table = combination(counts, 4);
    for (int index = 0; index < 10;) {
        printf("%i ", table[index++]); }
    printf("\n");
    free(table);
    return 0; }
```

Przykładowe wykonanie

```
Out: 2 3 2 0 2 3 2 0 0 2
```

11.3.3 Permutations: Wyznaczanie wszystkich permutacji

Napisz program `permutations`, który przyjmuje jako argument wywołania nieujemną liczbę całkowitą i wypisuje na standardowe wyjście wszystkie permutacje wszystkich mniejszych od niej nieujemnych liczb całkowitych. Program załącza tylko pliki nagłówkowe `stdio.h` i `stdlib.h`.

Przykładowe wykonanie

```
  Latex: ./permutations 3
Windows: permutations.exe 3
Out: 0 1 2
Out: 0 2 1
Out: 1 0 2
Out: 1 2 0
Out: 2 0 1
Out: 2 1 0
```

11.3.4 Substring: Wycinek napisu

Napisz funkcję `substring`, która przyjmuje stały napis, indeks pierwszego znaku oraz długość pewnego wycinka tego napisu, tworzy dynamicznie kopię tego wycinka i zwraca jej adres. Funkcja alokuje dla kopii najmniejszą możliwą ilość pamięci. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    char *string = substring("a long time ago in a galaxy far far away", 7, 8);
    printf("%s\n", string);
    free(string);
    return 0; }
```

Wykonanie

```
Out: time ago
```

11.3.5 Catenate: Łączenie napisów

Napisz funkcję `catenate`, która przyjmuje dwa stałe napisy dowolnej długości, tworzy trzeci będący ich połączeniem alokując dla niego jak najmniej pamięci i zwraca jego adres. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    char *string = catenate("to samo", "lot do Londynu");
    printf("%s\n", string);
    free(string);
    return 0; }
```

Wykonanie

```
Out: to samolot do Londynu
```

11.3.6 Variation: Losowa wariacja

Napisz funkcję `variation`, która przyjmuje nieujemne liczby całkowite $k \leq n$, tworzy dynamicznie k - elementową tablicę niepowtarzających się losowych liczb całkowitych z przedziału od 0 włącznie do n wyłącznie, i zwraca adres tej tablicy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    srand(time(NULL));
    int *table = variation(10, 100);
    for (int index = 0; index < 10;) {
        printf("%i ", table[index++]); }
    printf("\n");
    free(table);
    return 0; }
```

Przykładowe wykonanie

```
Out: 78 15 23 50 49 87 2 35 98 69
```

11.3.7 Relay: Przesiadka

Kierowca planuje podróż z miasta początkowego przez kilka pośrednich do miasta końcowego. Trasa jest na tyle długa, że wymaga po drodze jednego noclegu. Kierowca chce tak wybrać miasto na nocleg, aby kilometraże pierwszego i drugiego dnia podróży jak najmniej się różniły. Napisz program `relay`, który czyta ze standardowego wejścia odległości między kolejnymi miastami do napotkania końca pliku i wypisuje na standardowe wyjście numer miasta, w którym wypada nocleg, przy czym miasto początkowe ma numer zero. Program załącza tylko pliki nagłówkowe `math.h`, `stdio.h` i `stdlib.h`.

Przykładowe wykonanie

```
 In: 15.2 23.1 2.5 7.3 11 5.3
Out: 2
```

11.3.8 Insert: Wstawianie napisu do napisu - 1 bitcoin

Napisz funkcję `insert`, która przyjmuje napis, indeks znaku w tym napisie, oraz stały napis i wstawia drugi napis do pierwszego począwszy od podanego indeksu, realokując w miarę potrzeby pamięć. Funkcja zwraca adres wynikowego napisu. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    char *string = calloc(1, 1);
    string = insert(string, 0, "diary");
    string = insert(string, 2, "ction");
    printf("%s\n", string);
    free(string);
    return 0; }
```

Wykonanie

```
Out: dictionary
```

11.3.9 Text: Wczytywanie tekstu

Napisz funkcję `text`, która przyjmuje adres pliku, czyta z niego znaki bez opuszczania znaków białych aż do napotkania końca pliku, umieszcza je w dynamicznie utworzonym napisie, i zwraca jego adres. Jeżeli nie wczytano ani jednego znaku, funkcja nie alokuje żadnej pamięci i zwraca wskaźnik pusty. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z plików nagłówkowych `stdio.h` i `stdlib.h`.

Przykładowy program

```
int main() {
    char *string = text(stdin);
    printf("%s\n", string);
    free(string);
    return 0; }
```

Wykonanie

```
 In: Ala ma kota.
 In: To kot Ali.
Out: Ala ma kota.
Out: To kot Ali.
```

11.3.10 Line: Wczytywanie jednej linii

Napisz funkcję `line`, która przyjmuje adres pliku, czyta z niego jedną linię, umieszcza ją w dynamicznie utworzonym napisie bez znaku końca linii, i zwraca adres tego napisu. Jeżeli wczytana linia zawiera tylko znak końca linii, funkcja tworzy napis pusty. Jeżeli nie wczytano ani jednego znaku, funkcja nie alokuje żadnej pamięci i zwraca wskaźnik pusty. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z plików nagłówkowych `stdio.h` i `stdlib.h`.

Przykładowy program

```
int main() {
    for (char *string; string = line(stdin); free(string)) {
        printf("%s\n", string); }
    return 0; }
```

Wykonanie

```
 In: Ala ma kota.
Out: Ala ma kota.
 In: To kot Ali.
Out: To kot Ali.
```

11.3.11 Word: Wczytywanie jednego słowa

Napisz funkcję `word`, która przyjmuje adres pliku, czyta z niego jedno słowo opuszczając poprzedzające je znaki białe, umieszcza to słowo w dynamicznie utworzonym napisie, i zwraca adres tego napisu. Jeżeli nie wczytano ani jednego znaku niebiałego, funkcja nie alokuje żadnej pamięci i zwraca wskaźnik pusty. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z plików nagłówkowych `stdio.h` i `stdlib.h`.

Przykładowy program

```
int main() {
    for (char *string; string = word(stdin); free(string)) {
        printf("%s ", string); }
    printf("\n");
    return 0; }
```

Wykonanie

```
 In:     Ala  ma kota.
Out: Ala ma kota.
```

11.3.12 Lines: Wczytywanie wszystkich linii

Napisz funkcję `lines`, która przyjmuje adres zmiennej całkowitej oraz adres pliku, czyta z tego pliku wszystkie linie, każdą umieszcza w dynamicznie utworzonym napisie, adresy kolejnych napisów umieszcza w dynamicznie utworzonej tablicy wskaźników, liczbę wczytanych linii wpisuje pod adres przekazany jako argument, i zwraca adres utworzonej tablicy wskaźników. Jeżeli nie wczytano ani jednej linii, funkcja nie alokuje żadnej pamięci i zwraca wskaźnik pusty. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z plików nagłówkowych `stdio.h` i `stdlib.h`.

Przykładowy program

```
int main() {
    int size;
    char **strings = lines(&size, stdin);
    for (int index = 0; index < size;) {
        printf("%s\n", strings[index++]); }
    for (int index = 0; index < size;) {
        free(strings[index++]); }
    free(strings);
    return 0; }
```

Wykonanie

```
 In: Ala ma kota.
 In: To kot Ali.
Out: Ala ma kota.
Out: To kot Ali.
```

11.3.13 Words: Wczytywanie wszystkich słów

Napisz funkcję `words`, która przyjmuje adres zmiennej całkowitej oraz adres pliku, czyta z tego pliku wszystkie słowa, każde umieszcza w dynamicznie utworzonym napisie, adresy kolejnych napisów umieszcza w dynamicznie utworzonej tablicy wskaźników, liczbę wczytanych słów wpisuje pod adres przekazany jako argument, i zwraca adres utworzonej tablicy wskaźników. Jeżeli nie wczytano ani jednego słowa, funkcja nie alokuje żadnej pamięci i zwraca wskaźnik pusty. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z plików nagłówkowych `stdio.h` i `stdlib.h`.

Przykładowy program

```
int main() {
    int size;
    char **strings = words(&size, stdin);
    for (int index = 0; index < size;) {
        printf("%s ", strings[index++]); }
    printf("\n");
    for (int index = 0; index < size;) {
        free(strings[index++]); }
    free(strings);
    return 0; }
```

Wykonanie

```
 In: Ala ma kota.
 In: To kot Ali.
Out: Ala ma kota. To kot Ali.
```

12 Struktury: 21 maja 2019
Struktury

12.1 Przykłady laboratoryjne z działu Struktury

12.1.1 Quiz: Test z tabliczki mnożenia

Napisz program `quiz`, który wypisuje na standardowe wyjście dziesięć losowych pytań z tabliczki mnożenia liczb od 1 do 10 i po każdym pytaniu wczytuje ze standardowego wejścia odpowiedź. Każde pytanie zadaje aż do uzyskania poprawnej odpowiedzi. Pytania nie mogą się powtarzać, przy czym pytania różniące się jedynie kolejnością mnożonych liczb uważamy za jednakowe. Ponadto pytania powinny być zadawane w kolejności rosnących wyników mnożenia.

Przykładowe wykonanie

```
Out: 1 7 In: 7
Out: 9 1 In: 5
Out: 9 1 In: 9
Out: 4 3 In: 12
Out: 6 2 In: 12
Out: 6 3 In: 18
Out: 4 5 In: 20
Out: 3 7 In: 21
Out: 5 6 In: 30
Out: 4 9 In: 36
Out: 6 7 In: 42
```

Rozwiązanie na następnej stronie

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

struct Question {
    int a, b; };

struct Question draw() {
    struct Question question = {1 + rand() % 10, 1 + rand() % 10};
    return question; }

void print(const struct Question *question) {
    printf("%i %i ", question->a, question->b); }

int solution(const struct Question *question) {
    return question->a * question->b; }

int equal(const struct Question *question1, const struct Question *question2) {
    return question1->a == question2->a && question1->b == question2->b ||
           question1->a == question2->b && question1->b == question2->a; }

void fill(struct Question questions[], int size) {
    for (int index2 = 0; index2 < size; ++index2) {
        int index1;
        do {
            questions[index2] = draw();
            for (index1 = 0;
                 index1 < index2 && !equal(&questions[index1], &questions[index2]);
                 ++index1); }
        while (index1 < index2); }}

void sort(struct Question questions[], int size) {
    int unordered = size;
    while (unordered) {
        unordered = 0;
        for (int index = 0; index + 1 < size; ++index) {
            if (solution(&questions[index + 1]) < solution(&questions[index])) {
                struct Question question = questions[index];
                questions[index] = questions[index + 1];
                questions[index + 1] = question;
                unordered = 1; }}}}

void ask(const struct Question questions[], int size) {
    for (int index = 0; index < size; ++index) {
        int answer;
        do {
            print(&questions[index]);
            scanf("%i", &answer); }
        while (answer != solution(&questions[index])); }}

int main() {
    srand(time(NULL));
    struct Question questions[10];
    fill(questions, 10);
    sort(questions, 10);
    ask(questions, 10); }
```

12.2 Zadania domowe z działu Struktury na 18 czerwca

12.2.1 Div: Dzielenie z resztą

Korzystając ze struktury `div_t` oraz funkcji `div` z pliku nagłówkowego `stdlib.h` napisz program `div`, który wczytuje ze standardowego wejścia dwie liczby całkowite i wypisuje na standardowe wyjście iloraz oraz resztę z dzielenia pierwszej przez drugą. Program załącza tylko pliki nagłówkowe `stdio.h` i `stdlib.h`.

Przykładowe wykonanie

```
 In: 23 5
Out: 4 3
```

12.2.2 Yearday: Kolejny dzień roku - 1 bitcoin

Korzystając ze struktury `tm` oraz funkcji z pliku nagłówkowego `time.h` napisz program `yearday`, który wczytuje ze standardowego wejścia datę w formacie `yyyy mm dd` i wypisuje na standardowe wyjście numer odpowiadającego jej dnia w roku licząc od jednego dla pierwszego stycznia. Program załącza tylko pliki nagłówkowe `stdio.h` i `time.h`.

Przykładowe wykonanie

```
 In: 2018 04 19
Out: 109
```

12.2.3 Date: Bieżąca data

Korzystając ze struktury `tm` oraz funkcji z pliku nagłówkowego `time.h` napisz program `date`, który wypisuje na standardowe wyjście aktualną datę zapisaną jak poniżej. Program załącza tylko pliki nagłówkowe `stdio.h` i `time.h`.

Przykładowe wykonanie

```
Out: friday 20 april 2018
```

12.2.4 Interval: Dni między datami

Korzystając ze struktury `tm` oraz funkcji z pliku nagłówkowego `time.h` napisz program `interval`, który wczytuje ze standardowego wejścia dwie daty w formacie `yyyy mm dd` i wypisuje na standardowe wyjście liczbę dni upływających od pierwszej do drugiej. Program załącza tylko pliki nagłówkowe `stdio.h` i `time.h`.

Przykładowe wykonanie

```
 In: 2018 04 30
 In: 2018 05 02
Out: 2
```

12.2.5 Polygonal: Długość łamanej

Napisz strukturę `Point` przechowującą kartezjańskie współrzędne punktu na płaszczyznie. Napisz funkcję `polygonal`, która przyjmuje stałą tablicę takich punktów oraz jej rozmiar i zwraca długość łamanej otwartej łączącej te punkty, od pierwszego do ostatniego. Jeżeli tablica zawiera mniej niż dwa punkty, funkcja zwraca zero. Struktura i funkcja powinny być przystosowane do użycia w przykładowym programie poniżej. Korzystają one tylko z pliku nagłówkowego `math.h`.

Przykładowy program

```
int main() {
    const struct Point points[] = {{-0.3, 7.5}, {9.5, -3.7}, {5., 0.4}};
    printf("%lg\n", polygonal(points, 3));
    return 0; }
```

Wykonanie

```
Out: 20.9699
```

12.2.6 Center: Środek odcinka

Napisz strukturę `Point` przechowującą kartezjańskie współrzędne punktu na płaszczyznie. Napisz funkcję `center`, która przyjmuje adresy dwóch stałych punktów i zwraca środek łączącego je odcinka. Struktura i funkcja powinny być przystosowane do użycia w przykładowym programie poniżej. Nie korzystają one z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    const struct Point point1 = {-2.5, 3.7};
    const struct Point point2 = {3.4, 10.9};
    struct Point point = center(&point1, &point2);
    printf("%lg %lg\n", point.x, point.y);
    return 0; }
```

Wykonanie

```
Out: 0.45 7.3
```

12.2.7 Distance: Długość odcinka

Napisz strukturę `Point` przechowującą kartezjańskie współrzędne punktu na płaszczyznie. Napisz funkcję `distance`, która przyjmuje adresy dwóch stałych punktów i zwraca ich odległość. Struktura i funkcja powinny być przystosowane do użycia w przykładowym programie poniżej. Korzystają one tylko z pliku nagłówkowego `math.h`.

Przykładowy program

```
int main() {
    const struct Point point1 = {-2.5, 3.7};
    const struct Point point2 = {3.4, 10.9};
    printf("%lg\n", distance(&point1, &point2));
    return 0; }
```

12.2.8 Polygon: Wielokąt foremny - 1 bitcoin

Napisz strukturę `Polygon` przechowującą liczbę kątów oraz długość boku wielokąta foremnego. Zaimplementuj:

1. Funkcję `area`, która przyjmuje adres stałego wielokąta i zwraca jego pole.
2. Funkcję `scaled`, która przyjmuje adres stałego wielokąta oraz skalę jednokładności i zwraca wielokąt przekształcony przez tę jednokładność.
3. Funkcję `scale`, która przyjmuje adres wielokąta oraz skalę jednokładności i przekształca podany wielokąt przez tę jednokładność.

Struktura i funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Korzystają one tylko z pliku nagłówkowego `math.h`.

Przykładowy program

```
int main() {
    const struct Polygon polygon1 = {7, 10.};
    printf("%lg\n", area(&polygon1));
    struct Polygon polygon2 = scaled(&polygon1, 2.);
    printf("%lg\n", area(&polygon2));
    scale(&polygon2, 2.);
    printf("%lg\n", area(&polygon2));
    return 0; }
```

Wykonanie

```
Out: 363.391
Out: 1453.56
Out: 5814.26
```

12.2.9 Triangle: Trójkąt

Zadeklaruj strukturę `Point` przechowującą kartezjańskie współrzędne punktu na płaszczyznie oraz strukturę `Triangle` reprezentującą trójkąt o wierzchołkach w danych punktach. Zaimplementuj:

1. Funkcję `area`, która przyjmuje adres stałego trójkąta i zwraca jego pole. Pole A trójkąta o bokach a, b, c, dane jest wzorem Herona:

$$A = \sqrt{s(s-a)(s-b)(s-c)} \quad \text{gdzie} \quad s = \frac{a+b+c}{2},$$

2. Funkcję `center`, która przyjmuje adres stałego trójkąta i zwraca jego środek masy. Środek masy to punkt, którego współrzędne są średnimi arytmetycznymi odpowiednich współrzędnych wszystkich wierzchołków.

Struktury i funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Korzystają one tylko z pliku nagłówkowego `math.h`.

Przykładowy program

```
int main() {
    const struct Triangle triangle = {{-3, 2}, {5, 4}, {1, -6}};
    printf("%lg\n", area(&triangle));
    struct Point point = center(&triangle);
    printf("%lg %lg\n", point.x, point.y);
    return 0; }
```

Wykonanie

```
Out: 36
Out: 1 0
```

12.2.10 Abonent: Książka telefoniczna

Każda linia pliku `abonent.txt` zawiera nazwisko, imię, oraz numer telefonu jednego abonenta. Numery telefonów są dziewięciocyfrowe, a nazwiska i imiona są jednoczłonowe, pisane małymi literami i mają najwyżej po trzydzieści jeden znaków. Linie pliku nie są w żaden sposób uporządkowane, zaś ich liczba może być dowolna i nie jest z góry znana. Napisz program `abonent`, który przyjmuje jako argument wywołania ciąg małych liter i wypisuje na standardowe wyjście dane wszystkich abonentów, których nazwisko zawiera ten ciąg. Program sortuje wydruk w kolejności alfabetycznej nazwisk, a następnie imion. Program załącza tylko pliki nagłówkowe `stdio.h`, `stdlib.h` i `string.h`.

Przykładowy plik abonent.txt

```
tarkowski stanislaw 036107800
poniatowski jozef 847109428
sobieski jan 007654321
sniezka krolewna 047957321
kleks ambrozy 123456789
poniatowski stanislaw 758840401
```

Przykładowe wykonanie

```
  Linux: ./abonent ni
Windows: abonent.exe ni
Out: poniatowski stanislaw 758840401
Out: poniatowski jozef 847109428
Out: sniezka krolewna 047957321
```

13 Bloki: 28 maja 2019
Blokowe struktury danych

13.1 Przykłady laboratoryjne z działu Bloki

13.1.1 Matrix: Tablica dwuwymiarowa o dowolnych rozmiarach

Napisz strukturę `Matrix` reprezentującą dwuwymiarową tablicę liczb rzeczywistych o wierszach i kolumnach indeksowanych od zera. Zaimplementuj:

- Funkcję `alloc`, która przyjmuje rozmiary tablicy, alokuje pamięć dla danych oraz samej struktury, i zwraca adres struktury.
- Funkcję `dealloc`, która przyjmuje adres struktury i zwalnia pamięć danych oraz samej struktury.
- Funkcję `get`, która przyjmuje adres struktury, indeks wiersza oraz indeks kolumny, i zwraca wartość odpowiadającego im elementu.
- Funkcję `set`, która przyjmuje adres struktury, indeks wiersza oraz indeks kolumny, nową wartość odpowiadającego im elementu, i nadaje mu tę wartość.
- Funkcję `print`, która przyjmuje adres struktury i drukuje wszystkie elementy tablicy na standardowe wyjście.
- Funkcję `scan`, która przyjmuje adres struktry i wczytuje wszystkie elementy tablicy ze standardowego wejścia.

Struktura i funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Korzystają one tylko z plików nagłówkowych `stdio.h` i `stdlib.h`.

Przykładowy program

```
int main() {
    Matrix *matrix = alloc(2, 3);
    scan(matrix);
    print(matrix);
    dealloc(matrix);
    return 0; }
```

Przykładowe wykonanie

```
 In: 1 2 3
 In: 4 5 6
Out: 1 2 3
Out: 4 5 6
```

Rozwiązanie na następnej stronie

```
#include <stdio.h>
#include <stdlib.h>

struct Matrix {
    int rows, columns;
    double *elements; };

double get(struct Matrix *matrix, int row, int column) {
    return matrix->elements[row * matrix->columns + column]; }

void set(struct Matrix *matrix, int row, int column, double element) {
    matrix->elements[row * matrix->columns + column] = element; }

void print(struct Matrix *matrix) {
    for (int row = 0; row < matrix->rows; ++row) {
        for (int column = 0; column < matrix->columns; ++column) {
            printf("%lg ", get(matrix, row, column)); }
        printf("\n"); }}

void scan(struct Matrix *matrix) {
    for (int row = 0; row < matrix->rows; ++row) {
        for (int column = 0; column < matrix->columns; ++column) {
            double element;
            scanf("%lg", &element);
            set(matrix, row, column, element); }}}

struct Matrix *alloc(int rows, int columns) {
    struct Matrix *matrix = malloc(sizeof(struct Matrix));
    matrix->rows = rows;
    matrix->columns = columns;
    matrix->elements = malloc(matrix->rows * matrix->columns * sizeof(double));
    return matrix; }

void dealloc(struct Matrix *matrix) {
    free(matrix->elements);
    free(matrix); }

int main() {
    int rows, columns;
    scanf("%i%i", &rows, &columns);
    struct Matrix *matrix = alloc(rows, columns);
    scan(matrix);
    printf("\n");
    print(matrix);
    dealloc(matrix);
    return 0; }
```

13.2 Zadania domowe z działu Bloki na nigdy

13.2.1 Histogram: Histogram

Napisz strukturę `Histogram` reprezentującą jednowymiarowy histogram o binach jednakowej szerokości, indeksowanych od zera. Zadeklaruj w niej składową `size` przechowującą liczbę binów. Ponadto zaimplementuj:

- Funkcję `alloc`, która przyjmuje krańce przedziału histogramowania oraz liczbę binów, alokuje pamięć dla danych oraz samej struktury, zeruje wszystkie biny, i zwraca adres struktury.
- Funkcję `dealloc`, która przyjmuje adres struktury i zwalnia pamięć danych oraz samej struktury.
- Funkcję `get`, która przyjmuje adres stałej struktury oraz indeks binu i zwraca liczbę zliczeń w tym binie.
- Funkcję `insert`, która przyjmuje adres struktury oraz liczbę rzeczywistą i zwiększa o jeden liczbę zliczeń w binie, w którym wypada ta liczba. Jeżeli liczba leży poza przedziałem histogramowania, histogram pozostaje bez zmian.

Struktura i funkcje powinny być przystosowana do użycia w przykładowym programie poniżej. Korzystają one jedynie z nagłówka `stdlib.h`.

Przykładowy program

```
int main() {
    Histogram *histogram = alloc(0., 1., 2);
    insert(histogram, 0.17);
    insert(histogram, 0.75);
    insert(histogram, 0.33);
    for (int index = 0; index < histogram->size;) {
        printf("%i ", get(histogram, index++)); }
    printf("\n");
    dealloc(histogram);
    return 0; }
```

Wykonanie

```
Out: 2 1
```

13.2.2 Buffer: Bufor kołowy

Bufor kołowy o ustalonej pojemności n działa następująco. Nowoutworzony bufor jest pusty. Dołożenie elementu zwiększa jego rozmiar o jeden. Kiedy w buforze jest już n elementów, dołożenie kolejnego zamazuje pierwszy i tak dalej. Nie zwiększa to już rozmiaru bufora. Można odczytać lub zmienić wartość elementu o podanym indeksie, przy czym indeks zero odpowiada najdawniej dołożonemu spośród jeszcze niezamazanych elementów. Napisz strukturę `Buffer` reprezentującą kołowy bufor liczb rzeczywistych. Zadeklaruj w niej składową `size` przechowującą rozmiar bufora. Ponadto zaimplementuj:

- Funkcję `alloc`, która przyjmuje pojemność bufora, alokuje pamięć dla danych oraz samej struktury, zeruje rozmiar bufora, i zwraca adres struktury.
- Funkcję `dealloc`, która przyjmuje adres struktury i zwalnia pamięć danych oraz samej struktury.
- Funkcję `get`, która przyjmuje adres struktury oraz indeks elementu i zwraca jego wartość.
- Funkcję `set`, która przyjmuje adres struktury, indeks elementu, oraz jego nową wartość i nadaje mu tę wartość.
- Funkcję `push`, która przyjmuje adres struktury oraz liczbę rzeczywistą i dokłada ją do bufora.

Struktura i funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Korzystają one tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    Buffer *buffer = alloc(2);
    push(buffer, -7.2);
    push(buffer, 6.3);
    set(buffer, 1, 5.1);
    push(buffer, 0.5);
    for (int index = 0; index < buffer->size;) {
        printf("%lg ", get(buffer, index++)); }
    printf("\n");
    dealloc(buffer);
    return 0; }
```

Wykonanie

```
Out: 5.1 0.5
```

13.2.3 Boolean: Tablica wartości logicznych

Napisz strukturę `Boolean` reprezentującą jednowymiarową tablicę wartości zero lub jeden, indeksowaną od zera. Tablica przechowuje te wartości w pojedynczych bitach. W strukturze zadeklaruj składową `size` przechowującą rozmiar tablicy. Ponadto zaimplementuj:

- Funkcję `alloc`, która przyjmuje rozmiar tablicy, alokuje pamięć dla danych oraz samej struktury, i zwraca adres struktury.
- Funkcję `dealloc`, która przyjmuje adres struktury i zwalnia pamięć danych oraz samej struktury.
- Funkcję `get`, która przyjmuje adres struktury oraz indeks elementu i zwraca jego wartość.
- Funkcję `set`, która przyjmuje adres struktury, indeks elementu, oraz jego nową wartość i nadaje mu tę wartość.

Struktura i funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Korzystają one tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    struct Boolean *boolean = alloc(5);
    for (int index = 0; index < boolean->size; ++index) {
        set(boolean, index, index % 2); }
    for (int index = 0; index < boolean->size;) {
        printf("%i ", get(boolean, index++)); }
    dealloc(boolean);
    return 0; }
```

Przykładowe wykonanie

```
Out: 0 1 0 1 0
```

13.2.4 Graph: Graf nieskierowany

Napisz strukturę `Graph` reprezentującą graf nieskierowany o ustalonej liczbie węzłów, indeksowanych od zera. Zaimplementuj:

- Funkcję `alloc`, która przyjmuje liczbę węzłów, alokuje pamięć dla danych oraz samej struktury, rozłącza wszystkie węzły, i zwraca adres struktury.

- Funkcję `dealloc`, która przyjmuje adres struktury i zwalnia pamięć danych oraz samej struktury.
- Funkcję `linked`, która przyjmuje adres struktury oraz indeksy dwóch węzłów i zwraca jeden, jeśli są one połączone, albo zero w przeciwnym razie.
- Metodę `link`, która przyjmuje adres struktury, indeksy dwóch węzłów oraz liczbę jeden lub zero i odpowiednio łączy lub rozłącza te węzły.

Struktura i funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Korzystają one tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    struct Graph *graph = alloc(4);
    link(graph, 0, 2, 1);
    printf("%i %i\n", linked(graph, 0, 1), linked(graph, 0, 2));
    dealloc(graph);
    return 0; }
```

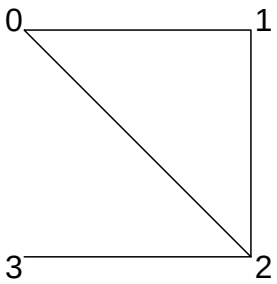
Wykonanie

```
Out: 0 1
```

13.2.5 Wandergraph: Wędrówka po grafie

Korzystając ze struktury `Graph` z poprzedniego zadania napisz program `wandergraph` pozwalający użytkownikowi chodzić po grafie przedstawionym na rysunku. Po uruchomieniu program wypisuje na standardowe wyjście indeks początkowego węzła, równy 0, i wczytuje ze standardowego wejścia indeks węzła, do którego użytkownik chce przejść. Zależnie od tego, czy jest on połączony z obecnym, program przenosi użytkownika lub pozostawia w dotychczasowym miejscu. Potem znowu wyświetla indeks aktualnego węzła i tak dalej. Program załącza tylko pliki nagłówkowe `stdio.h` i `stdlib.h`.

Graf



Początek przykładowego wykonania

```
Out: 0  In: 3
Out: 0  In: 1
Out: 1
```

13.2.6 Vector: Pojemnik typu wektor

Pojemnik typu wektor to rodzaj tablicy powiększającej się w miarę dokładania nowych elementów. Napisz strukturę `Vector` reprezentującą wektor liczb rzeczywistych. Zadeklaruj w niej składową `size` przechowującą rozmiar wektora. Ponadto zaimplementuj:

- Bezargumentową funkcję `alloc`, która alokuje pamięć dla struktury, zeruje rozmiar wektora, i zwraca adres struktury.

- Funkcję `dealloc`, która przyjmuje adres struktury i zwalnia pamięć danych oraz samej struktury.
- Funkcję `get`, która przyjmuje adres struktury oraz indeks elementu i zwraca jego wartość.
- Funkcję `set`, która przyjmuje adres struktury, indeks elementu, oraz jego nową wartość, i nadaje mu tę wartość.
- Funkcję `push`, która przyjmuje adres struktury oraz liczbę rzeczywistą i dokłada tę liczbę na koniec wektora zwiększając jego rozmiar o jeden.

Struktura oraz funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Korzystają one tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    struct Vector *vector = alloc();
    push(vector, 1.2);
    push(vector, 2.3);
    set(vector, 0, 3.4);
    for (int index = 0; index < vector->size;) {
        printf("%lg ", get(vector, index++)); }
    printf("\n");
    dealloc(vector);
    return 0; }
```

Wykonanie

```
Out: 3.4 2.3
```

14 Wiązania: 4 czerwca 2019
Wiązane struktury danych

14.1 Przykłady laboratoryjne z działu Wiązania

14.1.1 Stack: Stos

Stos to struktura danych podobna do stosu kartek. Umożliwia kładzenie elementów na górę oraz ich zdejmowanie z góry. Napisz strukturę `Stack` reprezentującą stos liczb rzeczywistych. Zaimplementuj:

- Bezargumentową funkcję `alloc`, która alokuje pamięć dla struktury, tworzy pusty stos, i zwraca adres struktury.
- Funkcję `empty`, która przyjmuje adres struktury i zwraca jeden jeśli stos jest pusty albo zero w przeciwnym razie.
- Funkcję `push`, która przyjmuje adres struktury oraz liczbę rzeczywistą i odkłada tę liczbę na stos.
- Funkcję `pop`, która przyjmuje adres struktury, zdejmuje ze stosu element i zwraca jego wartość.
- Funkcję `clear`, która przyjmuje adres struktury i usuwa ze stosu wszystkie elementy.
- Funkcję `dealloc`, która przyjmuje adres struktury i zwalnia pamięć danych oraz samej struktury.

Struktura i funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Program ten czyta ze standardowego wejścia liczby rzeczywiste do napotkania końca pliku i wypisuje je na standardowe wyjście w odwrotnej kolejności.

Przykładowy program

```
int main() {
    struct Stack *stack = alloc();
    for (double value; scanf("%lg", &value) == 1;) {
        push(stack, value); }
    while (!empty(stack)) {
        printf("%lg ", pop(stack)); }
    printf("\n");
    dealloc(stack);
    return 0; }
```

Wykonanie

```
Out: 2.3 1.2
```

Rozwiązanie na następnej stronie

```
#include <stdio.h>
#include <stdlib.h>

struct Stack {
    struct Node {
        double value;
        struct Node *lower; };
    struct Node *top; };

struct Stack *alloc() {
    struct Stack *stack = malloc(sizeof(struct Stack));
    stack->top = NULL;
    return stack; }

int empty(struct Stack *stack) {
    return stack->top == NULL; }

void push(struct Stack *stack, double value) {
    struct Node *node = malloc(sizeof(struct Node));
    node->value = value;
    node->lower = stack->top;
    stack->top = node; }

double pop(struct Stack *stack) {
    struct Node *node = stack->top;
    double value = node->value;
    stack->top = node->lower;
    free(node);
    return value; }

void clear(struct Stack *stack) {
    while (stack->top) {
        struct Node *node = stack->top;
        stack->top = node->lower;
        free(node); }}

void dealloc(struct Stack *stack) {
    while (stack->top) {
        struct Node *node = stack->top;
        stack->top = node->lower;
        free(node); }
    free(stack); }

int main() {
    struct Stack *stack = alloc();
    for (double value; scanf("%lg", &value) == 1;) {
        push(stack, value); }
    while (!empty(stack)) {
        printf("%lg ", pop(stack)); }
    printf("\n");
    dealloc(stack);
    return 0; }
```

14.2 Zadania domowe z działu Wiązania na nigdy

14.2.1 Register: Kasa sklepowa

Napisz program `register` obliczający łączną należność za towary zakupione w sklepie. Po uruchomieniu program wypisuje na standardowe wyjście początkową należność równą zero, po czym wczytuje ze standardowego wejścia cenę pierwszego towaru. Następnie wypisuje aktualną należność równą tej cenie, wczytuje cenę następnego towaru i tak dalej. Jeżeli zamiast ceny wczytane zostaje polecenie `undo`, program odejmuje od aktualnej należności ostatnio dodaną cenę. Kolejne polecenie `undo` odejmuje wcześniej dodaną cenę i tak dalej. Program kończy działanie po napotkaniu końca pliku lub polecenia `exit`. Program załącza tylko pliki nagłówkowe `stdio.h`, `stdlib.h` i `string.h`.

Przykładowe wykonanie

```
Out: 0  In: 3
Out: 3  In: 5
Out: 8  In: undo
Out: 3  In: 10
Out: 13 In: exit
```

14.2.2 Ariadna: Nić Ariadny

Idąc przez miasto zapisujemy na przemian w kolejnych liniach jaką ulicą idziemy oraz w którą stronę skręcamy. Nazwy ulic mają najwyżej 63 znaki. Napisz program `ariadna`, który przyjmuje jako argument wywołania nazwę pliku tekstowego z takim zapisem i wypisuje na standardowe wyjście analogiczny zapis drogi powrotnej. Program załącza tylko pliki nagłówkowe `stdio.h` i `stdlib.h`.

Przykładowy plik wejściowy input.txt

```
Newelska
prawo
Księcia Janusza
lewo
Dywizjonu 303
```

Przykładowe wykonanie

```
  Linux: ./ariadna input.txt
Windows: ariadna.exe input.txt
Out: Dywizjonu 303
Out: prawo
Out: Księcia Janusza
Out: lewo
Out: Newelska
```

14.2.3 Brackets: Dopasowanie nawiasów

Napisz program `brackets`, który przyjmuje jako argument wywołania nazwę pliku z kodem `C` i wypisuje na standardowe wyjście `true` jeśli w kodzie tym nawiasy otwierające i zamykające są do siebie poprawnie dopasowane albo `false` w przeciwnym razie. Program uwzględnia nawiasy okrągłe, kwadratowe, oraz klamrowe i sprawdza dopasowanie nawiasów każdego rodzaju do siebie oraz do pozostałych rodzajów. Badany kod nie zawiera nawiasów w komentarzach ani w stałych tekstowych. Program załącza tylko pliki nagłówkowe `stdio.h` i `stdlib.h`.

Przykładowy plik wejściowy input.txt

```
int main() {
    printf("Hello World!\n"); }
    return 0; }
```

Przykładowe wykonanie

```
  Linux: ./brackets input.txt
Windows: brackets.exe input.txt
Out: false
```

14.2.4 System: Liczbowe układy pozycyjne

Napisz program `system`, który przyjmuje jako argument wywołania podstawę liczbowego układu pozycyjnego, zapis liczby przy tej podstawie, oraz nową podstawę, i wypisuje na standardowe wyjście zapis podanej liczby przy nowej podstawie. Program załącza tylko pliki nagłówkowe `stdio.h` i `stdlib.h`.

Przykładowe wykonanie

```
  Linux: ./system 16 1A 2
Windows: system.exe 16 1A 2
Out: 11010
```

14.2.5 Postfix: Notacja postfiksowa

Notacja postfiksowa, zwana też Odwrotną Notacją Polską, to sposób zapisu wyrażeń arytmetycznych umożliwiający łatwe obliczanie ich wartości przez komputer. W notacji tej symbol działania zapisuje się po jego argumentach. Przykładowo, wyrażenie $3/5$ ma postać `3 5 /`, wyrażenie $3/5+7$ ma postać `3 5 / 7 +`, zaś wyrażenie $7 + 3/5$ ma postać `7 3 5 / +`. Algorytm obliczania wartości wyrażeń postfiksowych jest następujący:

- Dla każdego kolejnego składnika wyrażenia:
  - Jeżeli składnik jest liczbą:
    - Odłóż ją na stos.
  - Jeżeli składnik jest symbolem działania:
    - Zdejmij ze stosu dwie liczby, wykonaj na nich to działanie, a jego wynik odłóż na stos.
- Zdejmij ze stosu wartość wyrażenia.

Napisz program `postfix`, który do napotkania końca pliku czyta ze standardowego wejścia składniki wyrażenia postfiksowego i wypisuje na standardowe wyjście jego wartość. Zapis liczb może zawierać dowolnie dużo znaków. Składniki wyrażenia, czyli liczby i operatory, są oddzielone znakami białymi. Zaimplementuj dodawanie, odejmowanie, mnożenie i dzielenie. Program załącza tylko pliki nagłówkowe `stdio.h`, `stdlib.h` i `string.h`.

Przykładowe wykonanie

```
 In: 7 3 5 / +
Out: 7.6
```

14.2.6 Infix: Notacja infiksowa

W analogii do notacji postfiksowej, zwykła notacja nazywa się też infiksową, gdyż symbole działań znajdują się między liczbami. Aby obliczyć wartość wyrażenia infiksowego, najprościej przetłumaczyć je najpierw na postfiksowe. Algorytm tłumaczenia wyrażeń zawierających liczby, operatory dwuargumentowe oraz nawiasy, jest następujący:

- Dla każdego elementu wyrażenia infixowego:
  - Jeżli element jest liczbą:
    - Dopisz go do wyrażenia postfixowego.
  - Jeżeli element jest operatorem:

        * Dopóki stos jest niepusty oraz element na stosie jest operatorem o priorytecie wyższym niż element:
            · Zdejmij operator ze stosu i dodaj go do wyrażenia postfixowego.
        * Połóż element na stosie.
    - Jeżeli element jest lewym nawiasem:
        * Odłóż go na stos.
    - Jeżeli element jest prawym nawiasem:
        * Dopóki element na stosie nie jest lewym nawiasem:
            · Zdejmij go ze stosu i dopisz do wyrażenia postfixowego.
        * Zdejmij lewy nawias ze stosu.
- Dopóki stos nie jest pusty:
    - Zdejmuj elementy ze stosu i dopisuj je do wyrażenia postfixowego.

Napisz program `infix`, który do napotkania końca pliku czyta ze standardowego wejścia składniki wyrażenia infiksowego i wypisuje na standardowe wyjście odpowiadające mu wyrażenie postfiksowe. Zapis liczb może zawierać dowolnie dużo znaków. Składniki wyrażenia infiksowego, czyli liczby, operatory i nawiasy, są oddzielone znakami białymi. Składniki wyrażenia postfiksowego program oddziela spacjami. Zaimplementuj dodawanie, odejmowanie, mnożenie, dzielenie i nawiasy. Program załącza tylko pliki nagłówkowe `stdio.h`, `stdlib.h` i `string.h`.

Przykładowe wykonanie

```
 In: 7 * ( 2 + 3 )
Out: 7 2 3 + *
```

14.2.7 Queue: Kolejka jednokierunkowa

Kolejka jednokierunkowa to struktura danych podobna do kolejki w sklepie. Umożliwia dodawanie elementów na koniec kolejki oraz ich wyjmowanie z początku. Napisz strukturę `Queue` reprezentującą kolejkę liczb rzeczywistych. Zadeklaruj w niej składową `front` pamiętającą adres początku kolejki. Ponadto zaimplementuj:

- Bezargumentową funkcję `alloc`, która alokuje pamięć dla struktury, tworzy pustą kolejkę, i zwraca adres struktury.
- Funkcję `dealloc`, która przyjmuje adres struktury i zwalnia pamięć danych oraz samej struktury.
- Funkcję `push`, która przyjmuje adres struktury oraz liczbę rzeczywistą i dodaje tę liczbę na koniec kolejki.
- Funkcję `pop`, która przyjmuje adres struktury, usuwa element z początku kolejki, i zwraca jego wartość.

Struktura i funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Korzystają one tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    struct Queue *queue = alloc();
    push(queue, 1.2);
    push(queue, 2.3);
    while (queue->front) {
        printf("%lg ", pop(queue)); }
    printf("\n");
    dealloc(queue);
    return 0;  }
```

Wykonanie

```
Out: 1.2 2.3
```

14.2.8 Set: Zbiór jako drzewo wyszukiwań binarnych

Napisz strukturę `Set` reprezentującą zbiór niepowtarzających się liczb całkowitych. Zaimplementuj go jako najprostsze drzewo wyszukiwań binarnych, bez żadnych mechanizmów balansowania. Napisz:

- Bezargumentową funkcję `alloc`, która alokuje pamięć dla struktury, tworzy pusty zbiór, i zwraca adres struktury.
- Funkcję `dealloc`, która przyjmuje adres stałej struktury i zwalnia pamięć danych oraz samej struktury.
- Funkcję `count`, która przyjmuje adres stałej struktury oraz wartość całkowitą i zwraca liczbę jej wystąpień w zbiorze.
- Funkcję `insert`, która przyjmuje adres struktury oraz liczbę całkowitą i dodaje ją do zbioru.
- Funkcję `remove`, która przyjmuje adres struktury oraz liczbę całkowitą i usuwa ją ze zbioru.

Struktura i funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Korzystają one tylko z pliku nagłówkowego `stdlib.h`.

Przykładowy program

```
int main() {
    struct Set *set = alloc();
    insert(set, 2);
    insert(set, 0);
    remove(set, 4);
    remove(set, 0);
    for (int key = 0; key < 5;) {
        printf("%i ", count(set, key++)); }
    printf("\n");
    dealloc(set);
    return 0; }
```

Wykonanie

```
Out: 0 0 1 0 0
```

15 Kolokwium: 11 czerwca 2019