Zad. 1.1

Napisz program `InputOutput` odczytujący z terminala i wypisujący na konsoli kolejno wartości typu: `int`, `float` i napis. Bufor napisu ma mieć rozmiar 10. Odczyt napisu nie może powodować przepełnienia bufora. Przykładowa sesja:

```
a = 5
int value a = 5

b = 5
float value b = 5.000000

c = 0123456789
string value c = 012345678
```

- Dlaczego wypisany napis nie ma litery 9? Dlatego, że litera 9 nie zostanie zapisana w buforze, gdyż bufor o rozmiarze 10 (9 znaków + 1 na '\0') jest w pełni zapełniony pierwszymi 9 znakami.

Zad. 1.1.1

Odczytaj z terminala napis `abcdefghij` przy pomocy funkcji `fgets` z wykorzystaniem poprzedniego bufora napisu i wypisz go na konsoli.

- Dlaczego nie można pobrać drugiego napisu? Funkcja *scanf* zostawia w buforze znaki, które nie zostały przetworzone. W przypadku, gdy wprowadzimy napis naciskając klawisz *Enter*, *scanf* odczytuje dane i pozostawia '\n' w buforze. Kiedy próbujemy potem użyć funkcji *fgets*, to zamiast wczytać kolejny ciąg znaków, funkcja ta odczytuje pozostały znak '\n' z bufora, co prowadzi do sytuacji, gdzie drugi napis nie jest pobierany.
- Dlaczego na konsoli wyświetla się literka 9? Literka 9 pozostała w buforze. Kiedy funkcja *fgets* próbuje odczytać następny ciąg znaków, odczytuje właśnie tą pozostałą literkę 9, co powoduje wyświetlenie literki 9 na konsoli.
- W jaki sposób wyczyścić zawartość strumienia wejściowego? Aby wyczyścić zawartość strumienia wejściowego należy użyć funkcji *fflush*. Aby wyczyścić zawartość w buforze wejściowym, można użyć pętli, która odczyta pozostałe znaki aż do napotkania znaku nowej linii '\n' lub znaku końca strumienia 'EOF'.

Zad. 1.1.2 *

Odczytaj i wypisz na konsoli wartość zmienno-przecinkową typu `double`.

Zad. 1.1.3 *

Odczytaj i wypisz na konsoli cały napis "ala ma kota" przy pomocy funkcji `scanf`.

Zad. 1.1.4 *

Odczytaj i wypisz na konsoli cały napis "ala ma kota" przy pomocy funkcji `fgets`.

Zad. 1.2 *

Napisz program `empty` sprawdzający czy strumień wejściowy jest pusty.

Zad. 1.3

Napisz program `Fibo` wyliczający wartości ciągu Fibonacciego przy pomocy trzech funkcji.

- Podaj definicję ciągu Fibonacciego. Pierwszy i drugi wyraz ciągu Fibonacciego przyjmuje wartość 1, a każdy następny wyraz jest sumą dwóch poprzednich.

```
0   1   2   3   4   5   6    indeksy
1   1   2   3   5   8   13   wartosci
```

Zad. 1.3.1

Funkcja `fibo1` - metoda dziel i zwyciężaj.

```
f(0) = 1
f(1) = 1
f(n) = f(n-1) + f(n-2)
```

- Dokonaj analizy wywołania `fibo1(4)`. Analiza wywołania znajduje się w pliku „zadanie1_3.c”.
- Narysuj drzewo wywołań dla `fibo1(4)`. Drzewo wywołań znajduje się w pliku „zadanie1_3.c”.

Zad. 1.3.2

Funkcja `fibo2` - metoda programowania dynamicznego z ramką trójzębną.

```
r0  r1  r2
|---|---|
0   1   2   3   4   5   6    indeksy
1   1   2   3   5   8   13   wartości
    |---|---|
    r0  r1  r2
```

Przesunięcie ramki w prawo:

```
r0 = r1
r1 = r2
r2 = r1 + r0
```

- Ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość `n`-tego wyrazu ciągu Fibonacciego w funkcji `fibo2` dla `n >= 3`? Należy przesunąć ramkę w prawo n – 2 razy.
- Dokonaj analizy wywołania `fibo2(4)`. Analiza wywołania znajduje się w pliku „zadanie1_3.c”.
- Narysuj graf obliczeń dla `fibo2(4)`. Graf obliczeń znajduje się w pliku „zadanie1_3.c”.

Zad. 1.3.3 *

Funkcja `fibo3` - metoda programowania dynamicznego z ramką dwuzębną.

```
r0  r1
|---|
0   1   2   3   4   5   6    indeksy
1   1   2   3   5   8   13   wartości
|   |---|
pom r0  r1
```

Przesunięcie ramki w prawo:

```
pom = r0
r0 = r1
r1 = r0 + pom
```

- Ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość `n`-tego wyrazu ciągu Fibonacciego w funkcji `fibo3` dla `n >= 2`? Należy przesunąć ramkę w prawo n – 1 razy.
- Dokonaj analizy wywołania `fibo3(4)`. Analiza wywołania znajduje się w pliku „zadanie1_3.c”.
- Narysuj graf obliczeń dla `fibo3(4)`. Graf obliczeń znajduje się w pliku „zadanie1_3.c”.
- Która funkcja ma mniejszą złożoność obliczeniową `fibo2` czy `fibo3`? Obie funkcje mają taką samą złożoność obliczeniową, jednak funkcja `fibo3` ma mniejszą złożoność pamięciową, ponieważ używa tylko dwóch zmiennych do przechowywania wartości ciągu, podczas gdy `fibo2` używa trzech zmiennych.

Przykładowa sesja:

```
fibo1(4) = 5
fibo2(4) = 5
fibo3(4) = 5
```

Zad. 1.3.4 *

Podaj cztery inne funkcje wyliczające rekurencyjnie wartości ciągu Fibonacciego. Rekurencja z pamięcią. Rekurencja z akumulacją. Rekurencja na macierzach. Rekurencja ze strategią zmniejszania problemu.

Zad. 1.4 *

Napisz program `Sequence` wyliczający wartości ciągu $\{a_n\}$ przy pomocy trzech funkcji. Ciąg zdefiniowany jest rekurencyjnie:

```
a(0) = 1
a(1) = 4
a(n) = 2*a(n-1) + 0.5*a(n-2)
```

- Wylicz dziesięć pierwszych wyrazów ciągu $\{a_n\}$ w programie Excel.

Zad. 1.4.1 *

Funkcja `a1` - metoda dziel i zwyciężaj.

- Dokonaj analizy wywołania `a1(4)`. Analiza wywołania znajduje się w pliku „zadanie1_4.c”.
- Narysuj drzewo wywołań dla `a1(4)`. Drzewo wywołań znajduje się w pliku „zadanie1_4.c”.

Zad. 1.5 *

W pliku `liczba.txt` wyznacz funkcję `f(n)` wyliczającą liczbę cyfr dla dowolnej liczby całkowitej `n`. Wykorzystaj funkcję $\log_{10}(x)$.

Zad. 2.1

Napisz program `FiboTree` wypisujący, jak wyglądają kolejne wywołania funkcji `fibo1` razem z wartościami przez nie zwracanymi. Przykładowa sesja:

```
fibo1(4) = 5
fibo2(3) = 3
fibo3(2) = 2
fibo4(1) = 1
fibo5(0) = 1
fibo6(1) = 1
fibo7(2) = 2
fibo8(1) = 1
fibo9(0) = 1
```

- Sprawdź czy drzewo wywołań z wcześniejszego zadania zostało poprawnie narysowane. Drzewo wywołań z wcześniejszego zadania zostało poprawnie narysowane.

Zad. 2.2

Napisz program `Sequence` wyliczający wartości ciągu $\{a_n\}$ przy pomocy trzech funkcji. Ciąg zdefiniowany jest rekurencyjnie:

```
a(0) = 1
a(1) = 4
a(n) = 2*a(n-1) + 0.5*a(n-2)
```

Zad. 2.2.1 *

Funkcja `a2` - metoda programowania dynamicznego z ramką trójzębną.

- Narysuj schemat dla ramki trójzębnej analogicznie jak dla ciągu Fibonacciego. Schemat ramki trójzębnej znajduje się w pliku „zadanie2_2”.c.
- Ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość `n`-tego wyrazu ciągu $\{a_n\}$ w funkcji `a2` dla `n >= 3`? Należy przesunąć ramkę w prawo n – 2 razy.
- Pętla przesuwająca ramkę tym razem musi startować od indeksu 1.
- Dokonaj analizy wywołania `a2(4)`. Analiza wywołania znajduje się w pliku „zadanie2_2.c”.

- Narysuj graf obliczeń dla `a2(4)`. Graf obliczeń znajduje się w pliku „zadanie2_2.c”.

Zad. 2.2.2

Funkcja `a3` - metoda programowania dynamicznego z ramką dwuzębną.

- Narysuj schemat dla ramki dwuzębnej analogicznie jak dla ciągu Fibonacciego. Schemat ramki dwuzębnej znajduje się w pliku „zadanie2_2”.c.
- Ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość `n`-tego wyrazu ciągu $\{a_n\}$ w funkcji `a3` dla `n >= 2`? Należy przesunąć ramkę w prawo n – 1 razy.
- Pętla przesuwająca ramkę tym razem musi startować od indeksu 1.
- Dokonaj analizy wywołania `a3(4)`. Analiza wywołania znajduje się w pliku „zadanie2_2.c”.
- Narysuj graf obliczeń dla `a3(4)`. Graf obliczeń znajduje się w pliku „zadanie2_2.c”.

Przykładowa sesja:

```
a1(4) = 42.250000
a2(4) = 42.250000
a3(4) = 42.250000
```

Zad. 2.3 *

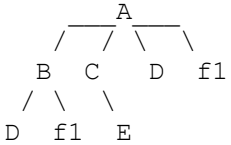
Napisz program `SequenceTree` wypisujący, jak wyglądają kolejne wywołania funkcji `a1` razem z wartościami przez nie zwracanymi. Przykładowa sesja:

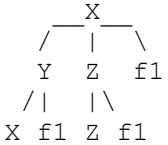
```
a1(4) = 42.250000
a2(3) = 19.000000
a3(2) = 8.500000
a4(1) = 4.000000
a5(0) = 1.000000
a6(1) = 4.000000
a7(2) = 8.500000
a8(1) = 4.000000
a9(0) = 1.000000
```

- Sprawdź czy drzewo wywołań z wcześniejszego zadania zostało poprawnie narysowane. Drzewo wywołań z wcześniejszego zadania zostało poprawnie narysowane.

Zad. 2.4 *

Zaproponuj metodę kodowanie drzewa katalogowego przy pomocy tablic trójwymiarowych w języku Java. Przetestuj ją dla następujących drzew.





Zad. 3.1

Napisz program `Fractions` wedle poniższego schematu i przetestuj jego działanie. Rozbuduj program o pozostałe działania na ułamkach, sugerowane nazwy funkcji: `sum`, `sub`, `mul`, `quo`. Przed każdą dodaną funkcją należy podać wyprowadzenie dla działania.

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

struct Fraction {
    int num, den;
};

/*
a/b + c/d = (a*d)/(b*d) + (c*b)/(d*b) = (a*d + c*b) / (b*d)
*/

struct Fraction sum(struct Fraction x, struct Fraction y) {
    int a = x.num;
    int b = x.den;

    int c = y.num;
    int d = y.den;

    struct Fraction z;

    z.num = a*d + c*b;
    z.den = b*d;

    return z;
}

void main() {
    struct Fraction x = {2, 3};
    struct Fraction y = {1, 4};

    struct Fraction z = sum(x, y);

    printf("%d/%d + %d/%d = %d/%d\n", x.num, x.den,
                                      y.num, y.den, z.num, z.den);
}
```

Zad. 3.1.1

Dodaj do programu funkcję `print` wypisującą działanie dla podanych ułamków i operatora.

```
void print(struct Fraction x, struct Fraction y, const char op);
```

Przykładowa sesja:

```
2/3 + 1/4 = 11/12
2/3 - 1/4 = 5/12
2/3 * 1/4 = 2/12
2/3 / 1/4 = 8/3
2/3 : 1/4 = 8/3
$ - nieznane dzialanie
```

Zad. 3.1.2 *

Dodaj do programu funkcję `printFraction` wypisującą ułamek wedle schematu:

```
printFraction((struct Fraction){2,0});  // NaN
printFraction((struct Fraction){0,2});  // 0
printFraction((struct Fraction){2,4});  // 1/2
printFraction((struct Fraction){-1,2}); // -1/2
printFraction((struct Fraction){1,-2}); // -1/2
printFraction((struct Fraction){4,-2}); // -2
printFraction((struct Fraction){5,-2}); // -2 1/2
```

Prototyp funkcji:

```
void printFraction(struct Fraction x);
```

Zad. 3.2

Napisz program `Strings` implementujący i testujący następujące funkcje:

Zad. 3.2.1

Funkcja `indexOf` działająca analogicznie do tej samej funkcji z klasy `String` w języku Java. W implementacji należy wykorzystać zmienne z poniższego schematu.

```
/*                 index
                   |
         0    1    2    3    4      indexes
str -> ['9']['9']['$'][' ']['\0']
       |         |
       str       ptr               pointers
*/

int indexOf(const char *str, int c);
```

Zad. 3.2.2 *

Funkcja `identity` zwracająca imię i nazwisko oddzielone spacją.

```
char *identity(const char *name, const char *surname);
```

Zad. 3.2.3 *

Funkcja `login` tworząca login użytkownika na podstawie pierwszej litery imienia i całego nazwiska. Należy pamiętać, że login na systemie Linux składa się z maksymalnie 32 znaków.

```
char *login(const char *name, const char *surname);
```

Zad. 3.3 *

Podaj słowny opis funkcji rekurencyjnej, która utworzy drzewo katalogowe dla tablicy trójwymiarowej z zad. 2.4 * z poprzednich laboratoriów. Można użyć funkcji tworzących plik i katalog oraz funkcji zmiany katalogu i przejścia do katalogu poziom wyżej.

Zad. 4.1

Napisz pełną implementację programu `Fractions` z odczytem danych z terminala i obsługą błędów.

Zad. 4.1.1

Dodaj funkcję `isNumber` sprawdzającą, czy napis `s` przechowuje liczbę całkowitą. Wykorzystaj funkcję `isdigit` z biblioteki standardowej.

```
/*
       0    1    2    3      index
s -> ['-']['3']['5']['\0']
     |    |    |    |        pointer
     s    s+1  s+2  s+3
*/

int isNumber(const char *s);
```

Zad. 4.1.2

Dodaj funkcję `trim` usuwającą z napisu `s` początkowe i końcowe znaki białe. Wykorzystaj funkcję `isspace` z biblioteki standardowej.

```
/*
       i         i'        j'        j
       0    1    2    3    4    5    6    7
s -> [' ']['a']['l']['a'][' '][' '][' ']['\0']
       k
*/

char *trim(char *s);
```

Zad. 4.1.3

Dodaj funkcję `getOperator` określającą operator z napisu `s`. Funkcja zwraca prawdę, jeśli napis jest poprawnym operatorem arytmetycznym.

```
int getOperator(char *op, const char *s);
```

Zad. 4.1.4

Dodaj funkcję `getFraction` określającą strukturę ułamkową z napisu `s`. Funkcja zwraca prawdę, jeśli napis jest ułamkiem zwykłym lub liczbą całkowitą. Napis może mieć postać:

liczba
liczba / liczba

```
/*                index
                  |
       0    1    2    3    4      indexes
s -> ['3']['7']['/']['5']['\0']
     |         |
     s         slash              pointers
*/

int getFraction(struct Fraction *x, const char *s);
```

Zad. 4.1.5

Zaimplementuj odczyt z terminala zgodnie z poniższym przykładem. Jeśli wprowadzony napis ma niepoprawny format, to jego odczyt należy powtórzyć. Przykładowa sesja:

```
a/b = 2/3
c/d = 1/4

op = +

2/3 + 1/4 = 11/12
```

Zad. 4.1.6 *

Zaimplementuj bezpieczny odczyt z terminala bez możliwości przepełnienia bufora linii.

Laboratorium 5

Zad. 5.1

Przepisz program `Files` otwierający plik do odczytu oraz implementujący i testujący następujące funkcje:

Zad. 5.1.1

Funkcja `printChars` odczytuje zawartość pliku bajt po bajcie.

```
void printChars(FILE *fp);
```

Zad. 5.1.2

Funkcja `printLines` odczytuje zawartość pliku linia po linii.

```
void printLines(FILE *fp);
```

Zad. 5.1.3 *

Funkcja `copy` kopiuje pliki analogicznie do komendy `cp`.

```
void copy(const char *addr1, const char *addr2);
```

Zad. 5.2

Napisz program `CountWords` odczytujący plik tekstowy oraz obliczający liczbę słów w tym pliku przy pomocy funkcji:

Zad. 5.2.1

Funkcja `countWords1` oblicza liczbę słów przy pomocy zmiennej stanu `inside`, gdzie wartość 0 oznacza, że zamierzamy odczytać znak poza słowem, a wartość 1 oznacza, że zamierzamy odczytać znak w słowie.

```
int countWords1(FILE *fp);
```

Zad. 5.2.2 *

Funkcja `countWords2` oblicza liczbę słów bez pomocy zmiennej stanu.

```
int countWords2(FILE *fp);
```

Zad. 5.3

Napisz program `DisplayWords` odczytujący plik tekstowy oraz wypisujący poszczególne słowa tego pliku przy pomocy funkcji `strtok` z biblioteki standardowej.

```
void printWords(FILE *fp);
```

Zad. 5.4

Napisz program `NewLine` testujący jakie znaki określają koniec linii w pliku tekstowym na systemie Windows i Linux. Porównaj działanie programu na obu systemach. Zaimplementuj funkcje:

Zad. 5.4.1

Funkcja `printHex` wypisuje plik w postaci dwucyfrowych liczb szesnastkowych.

```
void printHex(FILE *fp);
```

Zad. 5.4.2

Funkcja `printChar` wypisuje plik w postaci znaków umieszczonych w apostrofach.

```
void printChar(FILE *fp);
```

Zad. 5.4.3

Funkcja `printLinesHex` wypisuje plik linia po linii analogicznie do `printHex`.

```
void printLinesHex(FILE *fp);
```

Zad. 5.4.4

Funkcja `printLinesChar` wypisuje plik linia po linii analogicznie do `printChar`.

```
void printLinesChar(FILE *fp);
```

Przykładowa sesja:

```
1 linux.txt
2 windows.txt

Choose file: 1

linux.txt

61 6C 61 0A 6B 6F 74 0A

'a' 'l' 'a' '\n' 'k' 'o' 't' '\n'

61 6C 61 0A
6B 6F 74 0A

'a' 'l' 'a' '\n'
'k' 'o' 't' '\n'
```

Pliki do testów można pobrać ze strony:

```
http://balois.pl/file/pliki.zip
```

- Dlaczego rozmiary tych plików są różne? Różnica jest zależna od systemu operacyjnego, ponieważ każdy system inaczej oznacza koniec linii. Dla systemu „Windows” jest to ‘\r\n’, Dla systemu „Linux” jest to ‘\n’, a dla systemu „MAC OS” w starszych wersjach był to ‘\r’ a teraz jest to ‘\n’. Dlatego ten sam plik zapisany na systemie „Windows” waży więcej niż na innych systemach.

Dla systemu Linux program można uruchomić na stronie:

```
https://cocalc.com
```

- Jakie znaki określają koniec linii w systemie Linux, Windows i Mac OS? Odpowiedź na to pytanie jest już napisana wyżej.

```
https://pl.wikipedia.org/wiki/Koniec_linii
```

Zad. 5.5 *

Obsługa plików i katalogów w języku C dla systemu Windows. Referat dla dwóch osób na za dwa tygodnie.

Obsługa plików i katalogów w języku C w systemie Windows odbywa się zarówno za pomocą standardowych funkcji, takich jak ‘fopen’, ‘fread’, ‘fprintf’, jak i zaawansowanych funkcji Windows API, takich jak ‘CreateFile’, ‘ReadFile’ i ‘WriteFile’. Standardowe funkcje są przenośne między różnymi systemami, natomiast Windows API oferuje więcej opcji, np.: dostęp do atrybutów plików czy zarządzanie katalogami za pomocą funkcji ‘CreateDirectory’ i ‘RemoveDirectory’. Pliki w Windows mają specyficzne zakończenie linii ‘\r\n’, co odróżnia je od systemów Linux, które używają tylko ‘\n’. Windows API umożliwia bardziej zaawansowaną kontrolę nad operacjami na plikach, w tym zarządzanie uchwytami i atrybutami plików.

Zad. 5.6 *

Obsługa plików i katalogów w języku C dla systemu Linux. Referat dla dwóch osób na za trzy tygodnie.

Obsługa plików i katalogów w języku C w systemie Linux opiera się głównie na funkcjach z biblioteki standardowej oraz systemowych wywołaniach takich jak ‘open’, ‘read’, ‘write’, ‘close’ do pracy z plikami oraz ‘mkdir’, ‘rmdir’, ‘opendir’ do zarządzania katalogami. Standardowe funkcje, takie jak ‘fopen’, ‘fwrite’, ‘fscanf’, umożliwiają podstawową obsługę plików, ale Linux oferuje również bezpośrednie wywołania systemowe (syscalls) do zaawansowanej kontroli plików i katalogów, np.: zarządzanie uprawnieniami plików za pomocą ‘chmod’ czy ‘stat’. W systemie Linux koniec linii jest oznaczany przez ‘\n’. Można również przeglądać zawartość katalogów za pomocą funkcji ‘readdir’ oraz iterować po plikach.

Zad. 5.7 *

Obsługa plików i katalogów w języku Java. Referat dla dwóch osób na za cztery tygodnie.

W języku Java obsługa plików i katalogów opiera się głównie na klasach z pakietu ‘java.io’ oraz ‘java.nio.file’. Do podstawowych operacji służy klasa ‘File’, która pozwala tworzyć, usuwać oraz sprawdzać istnienie plików oraz katalogów. Do bardziej zaawansowanych operacji, takich jak odczyt, zapis i modyfikacja plików, używa się klas takich jak ‘FileReader’, ‘FileWriter’, ‘BufferedReader’, ‘BufferedWriter’ oraz nowoczesnych API z pakietu ‘java.nio.file’, takich jak ‘Files’ i ‘Paths’. Klasa ‘Files’ umożliwia pracę z plikami na poziomie ścieżek, tworzenie, kopiowanie, usuwanie oraz przeglądanie zawartości katalogów.

Laboratorium 6

Zad. 6.1

1. Sprawdzić czy zainstalowany jest kompilator GCC?
gcc --version
Jest zainstalowany.

2. Zainstaluj GCC jeśli nie jest zainstalowany.

Jest zainstalowany, nie muszę instalować.

3. Zaloguj się na drugim terminalu `ALT+F2`.
Drugi terminal otworzony.

4. W katalogu domowym utwórz katalog `programy`.
cd home/ur
mkdir Programy
Katalog utworzony.

5. W katalogu `programy` wykonaj `nano hello.c`.
cd home/ur/Programy
nano hello.c
Polecenie wykonane.

6. Napisz program `HelloWorld` i wyjdź z edytora.
Treść programu:

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

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

Program napisany.

Zad. 6.2

1. Na pierwszym terminalu sprawdź czy w katalogu domowym istnieje plik `.nanorc`.
cd home/ur
ls -a ~ | grep .nanorc
W katalogu domowym taki plik nie istnieje.

2. Sprawdź w jaki sposób w edytorze Nano ustawić rozmiar tabulacji na 4 i włączyć opcje tworzenia tabulacji z użyciem spacji.
W edytorze Nano, aby ustawić rozmiar tabulacji na 4 i włączyć opcje tworzenia tabulacji z użyciem spacji należy użyć opcji konfiguracyjnych, odpowiednio „set tabsize 4” i „set tabstospaces”.

3. W katalogu domowym otwórz plik `.nanorc` do edycji w edytorze Nano.
cd home/ur
nano .nanorc
Plik otworzony.

4. Dokonaj konfiguracji rozmiaru tabulacji oraz opcji tabulacji za pomocą spacji.
Treść programu:
set tabsize 4

set tabstospaces
Konfiguracja ukończona.

5. Na drugim terminalu otwórz plik `hello.c` i sprawdź, czy działają nowe ustawienia.
cd home/ur/Programy
nano hello.c
Nowe ustawienia działają.

6. Na pierwszym terminalu przejdź do katalogu `programy` i skompiluj `hello.c`.
```
gcc hello.c -o hello
```
cd home/ur/Programy
gcc hello.c -o hello
Program skompilowany.

7. Uruchom program `hello`.
cd home/ur/Programy
./hello
Program uruchomiony.

8. Dla pliku `hello.c` ustaw uprawnia tylko do odczytu. *
cd home/ur/Programy
chmod 444 hello.c
Uprawnienia ustawione.

9. Kiedy można uruchomić program `hello` przez podanie jego nazwy?
Program (jakikolwiek, nie musi być `hello`) można uruchomić, podając jego nazwę, gdy katalog, w którym znajduje się plik wykonywalny programu, jest dodany do zmiennej środowiskowej 'PATH'.

Zad. 6.3

Przy pomocy komendy `whoami` odczytaj nazwę bieżącego użytkownika.
whoami
Odpowiedź:
ur
Polecenie wykonane.

Zad. 6.4

Przy pomocy komendy `getent passwd` odczytaj wpis dla bieżącego użytkownika.
getent passwd $(whoami)
Odpowiedź:
ur:x:1000:1000:ur:/home/ur:/bin/bash
Polecenie wykonane.

Zad. 6.5

Przy pomocy komendy `getent group` odczytaj identyfikator grupy `staff`.
getent group staff
Odpowiedź:
staff:x:50:
Polecenie wykonane.

Zad. 6.6

Przy pomocy komendy `delgroup` usuń grupę `staff`.
sudo delgroup staff
Odpowiedź:
[sudo] password for ur:
info: Removing group 'staff' ...
Polecenie wykonane.

Zad. 6.7

Przy pomocy komendy `addgroup` utwórz grupę `staff` z poprzednim identyfikatorem.
sudo addgroup --gid 50 staff
Odpowiedź:
info: Adding group 'staff' (GID 50) ...
Polecenie wykonane.

- W pliku `/etc/group` znajdź wpis dla grupy `staff`.
grep staff /etc/group
Odpowiedź:
staff:x:50:
Polecenie wykonane.
- Jaki identyfikator ma grupa `staff` ? Identyfikator to 50.

`https://www.computerhope.com/unix/adduser.htm`

Zad. 6.8

Przy pomocy komendy `adduser` do grupy `staff` dodaj użytkownika `dyrektor`.
sudo adduser dyrektor
sudo adduser dyrektor staff
Odpowiedź:
info: Adding user 'dyrektor' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group 'dyrektor' (1001) ...
info: Adding new user 'dyrektor' (1001) with group 'dyrektor (1001) ...
info: Creating home directory 'home/dyrektor' ...
info: Copying files from '/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully

Changing the user information for dyrektor
Enter the new value, or press ENTER for the default
Full Name []: Pan dyrektor
Room Number []: 1
Work Phone []: 1
Home Phone []: 1
Other []: 0
Is the information correct? [Y/s]
info: Adding new user 'dyrektor' to supplemental / extra groups 'users' …
info: Adding user 'dyrektor' to group 'users' …
info: Adding user 'dyrektor' to group 'staff' …
Polecenie wykonane.

- W pliku `/etc/passwd` znajdź wpis dla użytkownika `dyrektor`.
grep dyrektor /etc/passwd
Odpowiedź:
Dyrektor:x:1001:1001:Pan dyrektor,1,1,1,0:/home/dyrektor:/bin/bash
Polecenie wykonane.
- Jaki katalog domowy ma użytkownik `dyrektor`? Katalog domowy to „home/dyrektor".
- Jaka powłoka obsługuje terminal użytkownika `dyrektor`? Powłoka obsługująca tego użytkownika to „bin/bash".

Zad. 6.9

Przy pomocy komendy `adduser` do grupy `student` dodaj użytkownika `jkowalski` z opisem `jan kowalski` oraz hasłem `haslo1`.
sudo addgroup student
sudo adduser jkowalski --ingroup student --gecos "jan kowalski"
Odpowiedź:
info: Selecting GID from range 1000 to 59999 …
info: Adding group 'student (GID 1002) …
info: Adding user 'jkowalski' …
info: Selecting UID from range 1000 to 59999 …
info: Adding new user 'jkowalski' (1002) wth group 'student (1002)' …
info: Creating home directory 'home/jkowalski' …
info: Copying files from '/etc/skel' …
New password:
Retype new password:
passwd: password updated successfully
info: Adding new user 'jkowalski' to supplemental / extra groups 'users' …
info: Adding user 'jkowalski' to group 'users' …
Polecenie wykonane.

- W pliku `/etc/passwd` przejrzyj wpis dla użytkownika `jkowalski`.
grep jkowalski /etc/passwd
Odpowiedź:
jkowalski:x:1002:1002:jan kowalski,,,:/home/jkowalski:/bin/bash

Polecenie wykonane.
- Jaki katalog domowy ma użytkownik `jkowalski`? Katalog domowy to „home/jkowalski”.
- Jaka powłoka obsługuje terminal użytkownika `jkowalski`? Powłoka obsługująca tego użytkownika to „bin/bash”.

Zad. 6.10

Przy pomocy komendy `deluser` usuń użytkownika `jkowalski`.
sudo deluser jkowalski
Odpowiedź:
info: Removing crontab …
info: Removing user ‘jkowalski’ …
Polecenie wykonane.

Zad. 6.11

Usuń katalog domowy użytkownika `jkowalski`.
sudo rm -r /home/jkowalski
Polecenie wykonane.

- Jaka opcja dla komendy `deluser` usuwa użytkownika razem z jego katalogiem domowym?
Polecenie, jakie należy użyć to „sudo deluser --remove-home jkowalski”.

Zad. 6.12

Przy pomocy komendy `openssl passwd -crypt` zaszyfruj napis `haslo1`. Powtórz komendę trzy razy. Czy szyfrowanie jest jednoznaczne?
Szyfrowanie nie jest jednoznaczne.

- Ile znaków maksymalnie może mieć hasło dla tej komendy? Hasło do tej komendy może mieć maksymalnie 8 znaków, reszta jest przycinana do tych 8 znaków.

Zad. 6.13

Przy pomocy komendy `useradd` do grupy `student` dodaj użytkownika `jkowalski` z opisem `jan kowalski` oraz hasłem `haslo1`. Parametry dla komendy należy dobrać w taki sposób, aby użytkownik został utworzony analogicznie jak w zadaniu 6.9.
sudo useradd -m -g student -c "jan kowalski" -s /bin/bash jkowalski
echo "jkowalski:haslo1" | sudo chpasswd
Polecenie wykonane.

- W pliku `/etc/passwd` przejrzyj wpis dla użytkownika `jkowalski`.
grep jkowalski /etc/passwd
Odpowiedź:
jkowalski:x:1002:1002:jan kowalski:/home/jkowalski:/bin/bash
Polecenie wykonane.
- Jaki katalog domowy ma użytkownik `jkowalski`? Katalog domowy to „home/jkowalski”.

- Jaka powłoka obsługuje terminal użytkownika `jkowalski`? Powłoka obsługująca tego użytkownika to „bin/bash”.

`https://www.computerhope.com/unix/useradd.htm`

Zad. 6.14

Przy pomocy komendy `userdel` usuń użytkownika `jkowalski` razem z jego katalogiem domowym.
sudo userdel -r jkowalski
Polecenie wykonane.

Zad. 6.15

Przepisz i przeanalizuj program `crypt-gnu.c` ilustrujący szyfrowanie haseł.
cd home/ur/Programy
gcc -o crypt-gnu crypt-gnu.c -lcrypt
./crypt-gnu
Analiza:
Program pobiera hasło i tworzy jego hash (skrócony zapis) za pomocą funkcji ‘crypt_r’ oraz soli ‘cd’. Funkcja ‘crypt_r’ zdefiniowana w nagłówku ‘crypt.h’, jest wersją ‘crypt’, która przechowuje dane szyfrowania w strukturze ‘crypt_data’. To pozwala na bezpieczne używanie funkcji w środowiskach wielowątkowych. W funkcji ‘main’ najpierw tworzona jest struktura ‘crypt_data’, której pole ‘initialized’ jest ustawiane na 0, aby przygotować je do inicjalizacji przez ‘crypt_r’. Następnie definiowane jest hasło, które zostanie zaszyfrowane. Funkcja ‘crypt_r’ przyjmuje hasło, sól oraz wskaźnik do struktury, aby przetworzyć hasło. Wynik szyfrowania jest zapisywany w zmiennej ‘encrypted’. Na końcu program wyświetla oryginalne hasło, jak i jego zaszyfrowaną wersję.

`http://balois.pl/file/crypt-gnu.c`

Zad. 6.16 *

Przepisz i przeanalizuj program `crypt-xopen.c` ilustrujący szyfrowanie haseł.
cd home/ur/Programy
gcc -o crypt-xopen crypt-xopen.c -lcrypt
./crypt- xopen
Analiza:
Program pobiera hasło i tworzy jego hash (skrócony zapis) za pomocą funkcji ‘crypt’ oraz soli ‘cd’. Funkcja ‘crypt’ zdefiniowana w nagłówku ‘unistd.h’ jest klasycznym narzędziem do szyfrowania haseł, opartym na algorytmie DES. Program używa preprocesora ‘#define _XOPEN_SOURCE’, aby umożliwić korzystanie z funkcji ‘crypt’, która jest częścią standardu X/Open. W funkcji ‘main’ zdefiniowane jest hasło, które następnie funkcja ‘crypt’ przetwarza hasło z użyciem soli ‘cd’, a wynik (zaszyfrowane hasło) jest przechowywany w zmiennej ‘encrypted’. Na koniec program wyświetla zarówno oryginalne hasło, jak i jego zaszyfrowaną wersję.

`http://balois.pl/file/crypt-xopen.c`

Poczytaj strony:

```
https://linux.die.net/man/3/crypt_r
https://manpages.ubuntu.com/manpages/bionic/pl/man3/crypt.3.html
```

Zad. 6.17

Napisz program `addusers` tworzący użytkowników na podstawie bazy w pliku `baza.txt` zawierającej:

```
jan;kowalski;haslo1
piotr;nowak;haslo2
grzegorz;adamski;haslo3
```

Użytkowników tworzymy analogicznie jak w zadaniu 6.13. Program ma działać poprawnie dla pliku `baza.txt` zapisanego zarówno w formacie Linux, jak i Windows.

cd home/ur/Programy
nano addusers.c
gcc -o addusers addusers.c
./addusers

Treść programu:

```
#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

int main() {
   FILE *file = fopen("baza.txt", "r");
   if (!file) {
      printf("\nNie mozna otworzyc pliku!\n\n");
      return 1;
   }

   printf("Plik otworzony!\n\n");
   char line[256];

   while (fgets(line, sizeof(line), file)) {
      line[strcspn(line, "\r\n")] = 0;

      char *imie = strtok(line, ";");
      char *nazwisko = strtok(NULL, ";");
      char *haslo = strtok(NULL, ";");

      if (!imie || !nazwisko || !haslo) {
         printf("Brak imienia lub nazwiska lub hasla!\n");
```

```
            continue;
        } else {
            printf("Wiersz poprawny!\n");
        }

        char username[50];
        snprintf(username, sizeof(username), "%c%s", imie[0], nazwisko);
        for (int i = 0; username[i]; i++) {
            username[i] = tolower(username[i]);
        }

        char command[256];
        snprintf(command, sizeof(command), "sudo useradd -m -c \"%s %s\" -s /bin/bash %s",
imie, nazwisko, username);

        if (system(command) != 0) {
            printf("Blad przy dodawaniu uzytkownika %s!\n", username);
            continue;
        }

        snprintf(command, sizeof(command), "echo \"%s:%s\" | sudo chpasswd", username,
haslo);

        if (system(command) != 0) {
            printf("Blad przy ustawianiu hasla dla uzytkownika %s!\n", username);
            continue;
        }

        printf("Uzytkownik %s został dodany z haslem!\n", username);
    }

    fclose(file);
    return 0;
}
```

Program napisany.

Zad. 6.18

Wejdź na poniższą stronę i obejrzyj program:

`https://dpaste.com/897JG9S9L`

Pobierz program na Linux komendą:

```
wget dpaste.com/897JG9S9L.txt -O hello.c
```

- Jakie rozszerzenie podajemy po adresie strony z programem? Rozszerzenie po adresie strony to '.txt'. Używane jest w celu pobrania pliku jako tekstowego.
- Jaka opcja służy do określenia nazwy pobieranego programu? Opcja '-O' służy do określenia nazwy pliku, pod jaką zostanie zapisany pobrany plik.

Wyświetl, skompiluj i uruchom pobrany program.
Program pod podanym linkiem nie istnieje, dlatego nie mogę go wyświetlić, skompilować i uruchomić.

Zad. 6.19

Umieść na stronie `dpaste.com` program `addusers` i pobierz go na Linux.
Link do programu: https://dpaste.com/3L8X7GCYX.
cd home/ur/Programy
wget dpaste.com/3L8X7GCYX.txt -O dpaste.c

Zad. 6.20 *

Napisz program `delusers` usuwający użytkowników podanych na liście w pliku.
cd home/ur/Programy
nano delusers.c
gcc -o delusers delusers.c
./delusers
Treść programu:

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

int main() {
   FILE *file = fopen("baza.txt", "r");
   if (!file) {
      printf("\nNie mozna otworzyc pliku!\n\n");
      return 1;
   }

   printf("Plik otworzony!\n\n");
   char line[256];

   while (fgets(line, sizeof(line), file)) {
      line[strcspn(line, "\r\n")] = 0;

      char *imie = strtok(line, ";");
      char *nazwisko = strtok(NULL, ";");
      char *haslo = strtok(NULL, ";");

      if (!imie || !nazwisko || !haslo) {
```

```
            printf("Brak imienia lub nazwiska lub hasla!\n");
            continue;
        } else {
            printf("Wiersz poprawny!\n");
        }

        char username[50];
        snprintf(username, sizeof(username), "%c%s", imie[0], nazwisko);
        for (int i = 0; username[i]; i++) {
            username[i] = tolower(username[i]);
        }

        char command[256];
        snprintf(command, sizeof(command), "sudo userdel -r %s", username);

        if (system(command) != 0) {
            printf("Blad przy usuwaniu uzytkownika %s!\n", username);
            continue;
        }

        printf("Uzytkownik %s został usuniety!\n", username);
    }

    fclose(file);
    return 0;
}
```

Program napisany.

Zad. 7.1

Napisz program `check` sprawdzający na jakim systemie został skompilowany. Wykorzystać kompilację warunkową. Przykładowa sesja:

```
Program compiled on: Windows
```

Zad. 7.2

Napisz program `arguments` wypisujący kolejno:

- Wartość zmiennej `argc`.
- Adres programu.
- Nazwę programu. *
- Przekazane parametry.

Przykładowa sesja:

```
argc = 2

addr: D:\Dev-C\arguments\arguments.exe

name: arguments.exe

params: *.c
```

Zad. 7.3

Napisz program `list` wywołujący komendę `dir` dla systemu Windows lub `ls` dla systemu Linux. Parametry podane przy uruchomieniu programu są przekazywane do obu komend.

Zad. 7.4

Napisz program `cyfry` wyliczający liczbę cyfr danej liczby całkowitej przy pomocy funkcji:

- Funkcja `cyfry1` wykorzystuje funkcję `log10`. *

- Funkcja `cyfry2` wykorzystuje funkcję `snprintf`.

- Funkcja `cyfry3` wykorzystuje dzielenie przez `10`. *

Przykładowa sesja:

```
cyfry1(-3579) = 4
cyfry2(-3579) = 4
cyfry3(-3579) = 4
```

Zad. 7.5 *

Napisz program `version` wypisujący wersję biblioteki `libc` na 3 różne sposoby dla systemu Linux i Windows.

Zad. 7.6 *

Napisz program `dodawanie` realizujący dodawanie pisemne. Interfejs programu musi wyglądać dokładnie tak samo, jak w programie `dodawanie.exe` na stronie autora. Zakładamy, że dane wejściowe mają postać:

```
Z: a = 0, 1, 2, ...
   b = 0, 1, 2, ...
```

- W jakich przypadkach program może dawać niepoprawne wyniki? Program może dawać niepoprawne wyniki w przypadku przepełnienia miejsca w tablicy danej liczby.

Przykładowa sesja:

```
a = 9237
b = 1267
```

```
 1 11
  9237
+ 1267
------
 10504
```

- Jaka liczba jest wyświetlana jako pierwsza? Jako pierwsza wyświetlana jest większa liczba.
- Ile może być maksymalnie przeniesień przy dodawaniu? Maksymalna liczba przeniesień to długość większej z dwóch liczb.
- Ile wynosi i od czego zależy szerokość słupka dodawania? Szerokość słupka dodawania wynosi długość większej z dwóch liczb plus jedno miejsce na przeniesienie i plus jedno miejsce na znak „+”. Szerokość słupka dodawania zawsze zależy od długości większej liczby.

Zad. 7.7 *

W programie `dodawanie2` dodaj do programu z poprzedniego zadania kontrolę poprawności danych wejściowych. W przypadku podania niepoprawnej liczby program ponawia prompt. Jeśli suma liczb przekracza rozmiar typu `int`, to program wypisuje stosowny komunikat.

Laboratorium 8

Zad. 8.1 *

Napisz program `regexp` ilustrujący wyrażenia regularne POSIX’a.

Zad. 8.2

Napisz program `misc` implementujący i testujący następujące funkcje:

1. Funkcja `losuj` zwraca wartość losową z przedziału `[a, b)`.

```
int losuj(int a, int b);
```

2. Funkcja `wariacje1` wylicza metodą kombinatoryczną liczbę ciągów 2-elementowych, jakie można utworzyć ze znaków określonych wyrażeniem regularnym `[a-zA-Z0-9./]`.

```
int wariacje1();
```

3. Funkcja `wariacje2` wylicza metodą brutalnej siły liczbę ciągów 2-elementowych, jakie można utworzyć z wartości określonych wyrażeniem regularnym `[a-zA-Z0-9./]`.*

```
int wariacje2();
```

4. Funkcja `set` w sposób losowy określa wartość parametru `salt` dla funkcji `crypt_r`.

```
void set(char salt[2]);
```

- Jaką ma postać i ile wartości może przyjmować parametr `salt` dla funkcji `crypt_r`? Postać `salt` to 2-znakowy string z 64 możliwych znaków dla każdego miejsca, to daje 64^2 czyli 4096 kombinacji.

`https://www.man7.org/linux/man-pages/man3/crypt_r.3.html`

- Czy dwuelementowego stringa można użyć do zainicjowania tablicy `salt[2]`? Można użyć.

5. Funkcja `errnoExample` ilustruje wypisanie błędu otwarcia pliku przy pomocy zmiennej globalnej `errno`.

```
void errnoExample();
```

- Przeczytaj rozdział NOTES ze strony:

`https://man7.org/linux/man-pages/man3/errno.3.html`

- Sprawdź na terminalu jaka nazwa symboliczna i jaki opis odpowiada błędowi nr 2. Nazwa symboliczna błędu nr 2 to „ENOENT” a opis to „No such file or directory”.

6. Funkcja `perrorExample` ilustruje wypisanie błędu otwarcia pliku przy pomocy funkcji `perror`. *

```
void perrorExample();
```

Laboratorium 9

Zad. 9.1

Przepisz program `copy` oraz przeanalizuj jego działanie.

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

#define MAX 512

int main(int argc, char* argv[]) {
    char buf[MAX];
    int desc_zrod, desc_cel;
    int lbajt;

    if (argc < 3) {
        fprintf(stderr, "Za malo argumentow. Uzyj:\n");
        fprintf(stderr, "%s <plik zrodlowy> <plik docelowy>\n", argv[0]);
        exit(1);
    }

    desc_zrod = open(argv[1], O_RDONLY);

    if (desc_zrod == -1) {
        perror("Blad otwarcia pliku zrodlowego");
```

```
        exit(1);
    }

    desc_cel = creat(argv[2], 0640);

    if (desc_cel == -1) {
        perror("Blad utworzenia pliku docelowego");
        exit(1);
    }

    while ((lbajt = read(desc_zrod, buf, MAX)) > 0) {
        if (write(desc_cel, buf, lbajt) == -1) {
            perror("Blad zapisu pliku docelowego");
            exit(1);
        }
    }

    if (lbajt == -1) {
        perror("Blad odczytu pliku zrodlowego");
        exit(1);
    }

    if (close(desc_zrod) == -1 || close(desc_cel) == -1) {
        perror("Blad zamkniecia pliku");
        exit(1);
    }

    exit(0);
}
```

- Co przechowują parametry: `argv[0], argv[1], argv[2]`? Parametry przechowują odpowiednio: `argv[0]` przechowuje nazwę wykonywalnego programu; `argv[1]` przechowuje pierwszy argument przekazany do programu, w tym przypadku jest to nazwa pliku źródłowego; `argv[2]` przechowuje drugi argument przekazany do programu, w tym przypadku jest to nazwa pliku docelowego.
- Omów funkcje: `open, create, read, write, close, exit`. Omówienie funkcji:
* Funkcja `open` służy do otwarcia pliku i zwraca desktryptor pliku, który jest używany do dalszych operacji na tym pliku. Jeśli funkcja zwraca -1, oznacza to, że wystąpił błąd podczas otwierania pliku.
* Funkcja `create` służy do tworzenia nowego pliku lub nadpisania istniejącego, jeśli już istnieje, z określonymi uprawnieniami, zwraca deskryptor pliku. Jeśli funkcja zwraca -1 oznacza to, że wystąpił błąd podczas tworzenia pliku.
* Funkcja `read` służy do odczytu danych z pliku. Przyjmuje deskryptor pliku, bufor, do którego dane mają być zapisane, i maksymalną liczbę bajtów do odczytu. Zwraca liczbę odczytanych bajtów, 0 gdy dosięgnięto koniec pliku lub -1 przy błędzie.
* Funkcja `write` służy do zapisu danych do pliku. Przyjmuje deskryptor pliku, bufor z danymi do zapisania i liczbę bajtów do zapisu. Zwraca liczbę zapisanych bajtów lub -1 w przypadku błędu.
* Funkcja `close` służy do zamknięcia deskryptora pliku, zwalniając wszelkie zasoby związane z nim. Zwraca 0 w przypadku sukcesu lub -1 w przypadku błędu.
* Funkcja `exit` służy do zakończenia programu. Argument tej funkcji określa kod wyjścia programu, przekazywany do systemu operacyjnego. Użycie exit(1) oznacza wyjście z błędem, a exit(0) oznacza zakończenie bez błędów.

Zad. 9.2

Przepisz program `size` oraz przeanalizuj jego działanie.

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

int main(int argc, char* argv[]) {
    int desc;
    long rozm;

    if (argc < 2) {
        fprintf(stderr, "Za malo argumentow. Uzyj:\n");
        fprintf(stderr, "%s <nazwa pliku>\n", argv[0]);
        exit(1);
    }

    desc = open(argv[1], O_RDONLY);

    if (desc == -1) {
        perror("Blad otwarcia pliku");
        exit(1);
    }

    rozm = lseek(desc, 0, SEEK_END);

    if (rozm == -1) {
        perror("Blad w pozycjonowaniu");
        exit(1);
    }

    printf("Rozmiar pliku %s: %ld\n", argv[1], rozm);

    if (close(desc) == -1) {
        perror("Blad zamkniecia pliku");
        exit(1);
    }

    exit(0);
}
```

- Omów funkcję `lseek`. Funkcja `lseek` jest używana do zmiany pozycji wskaźnika w otwartym pliku. W tym programie funkcja `lseek` jest używana do ustalenia rozmiaru pliku poprzez przesunięcie wskaźnika na koniec pliku, a następnie odczytanie jego pozycji, co jest równoznaczne z rozmiarem pliku. To efektywna metoda na określenie wielkości pliku bez konieczności czytania jego zawartości.

Zad. 9.3 *

Napisz program kopiujący zawartość pliku o nazwie podanej jako pierwszy parametr, do pliku którego nazwa podana jest jako drugi parametr.

Zad. 9.4 *

Napisz program zmieniający kolejność znaków w każdej linii pliku o nazwie podanej jako parametr.

Zad. 9.5 *

Napisz procedurę kopiowania ostatnich 10 znaków, słów i ostatnich 10 linii jednego pliku do innego.

Zad. 9.6 *

Napisz program do rozpoznawania czy plik o podanej nazwie jest plikiem tekstowym, plik tekstowy zawiera znaki o kodach 0-127, można w tym celu użyć funkcji `isascii`.

Zad. 9.7 *

Napisz program, który w pliku o nazwie podanej jako ostatni argument zapisze połączoną zawartość wszystkich plików, których nazwy zostały podane w linii poleceń przed ostatnim argumentem.

Zad. 9.8 *

Napisz program do porównywania plików o nazwach przekazanych jako argumenty. Wynikiem działania programu ma być komunikat, że pliki są identyczne, pliki różnią się od znaku nr <nr znaku> w linii <nr znaku linii> lub - gdy jeden z plików zawiera treść drugiego uzupełnioną o jakieś dodatkowe znaki - plik <nazwa> zawiera <liczba> znaków więcej niż zawartość pliku <nazwa>.