Systemy Operacyjne 1
Laboratorium 1
„Wprowadzenie do edytora VIM, kompilatora GCC, programu `make`, podręcznika `man` oraz powłoki systemu `Linux`"
(jeden tydzień)

dr inż. Arkadiusz Chrobot

3 października 2020

Wstęp

Niniejsza instrukcja zawiera opisu sposobu użytkowania oprogramowania narzędziowego, które będzie używane na pozostałych zajęciach laboratoryjny. Część 1 zawiera opis wybranych poleceń powłoki systemowej (interpretera poleceń). Część 2 zawiera opis działania edytora tekstowego VIM. Część 3 to krótki opis podręcznika systemowego, a część 4 zawiera objaśnienia dotyczące sposobu użycia kompilatora `gcc` i programu `make`. Część 5 zawiera informację na temat debuggera dostępnego w systemie Linux. Instrukcja kończy się listą zadań do wykonania.

1. Polecenia powłoki systemu Linux

System Linux oferuje użytkownikom dwa rodzaje interpreterów poleceń: graficzne i tekstowe. Graficzne interpretery są obecnie chętniej używane ze względu na krótki czas potrzebny do nauki ich obsługi, jednakże tekstowe interpretery w niektórych zastosowaniach są bardziej elastyczne i pozwalają na bardziej efektywną pracę. W systemie Linux, który jest kompatybilny z systemem Unix, dostępnych jest wiele interpreterów tekstowych, które są nazywane powłokami (ang. *shell*). Sposób ich działania i składnia większości poleceń są na ogół podobne. Najpopularniejszą powłoką jest `bash`. W tabeli 1 wymieniono polecenia, których opanowanie pozwala na obsługę systemu w zakresie wymaganym na laboratorium. Ich składnia jest w większości przypadków taka sama dla wszystkich powłok. Opcje poleceń, takie jak nazwy plików, są zapisane kursywą. Uwaga: w systemie Linux wielkość liter ma znaczenie.

Polecenie	**Opis**
cd *nazwa*	Zmiana bieżącego katalogu na wskazany. W szczególności *nazwa* może być nazwą katalogu nadrzędnego, czyli dwiema kropkami (..).
pwd	Wyświetlenie nazwy bieżącego katalogu.
ls	Wyświetlenie zawartości bieżącego katalogu.
mkdir *nazwa*	Utworzenie katalogu o podanej nazwie.
rmdir *nazwa*	Usunięcie katalogu o podanej nazwie, jeśli jest pusty.
rm *nazwa*	Usunięcie pliku o podanej nazwie.
rm -r *nazwa*	Usunięcie katalogu wraz z zawartością (rekurencyjnie).
cat *nazwa*	Wyświetlenie zawartości pliku tekstowego na ekran. Może też posłużyć, do połączenia kilku plików tekstowych w całość.
more *nazwa*	Wyświetlenie pliku tekstowego z podziałem na strony.
less *nazwa*	Wyświetlenie pliku tekstowego z podziałem na strony i możliwością nawigowania między stronami.
cp *nazwa gdzie*	Skopiowanie pliku o podanej nazwie do wskazanego katalogu.
mv *nazwa gdzie*	Przemieszczenie pliku do wskazanego katalogu lub zmiana jego nazwy.
chmod *prawa nazwa*	Zmiana praw dostępu do pliku.
ps aux	Wyświetlenie listy uruchomionych procesów wraz z informacją o ich stanie i identyfikatorze.
kill -9 *pid*	Przerwanie działania procesu o podanym *pid* (identyfikatorze).
pkill -9 *nazwa*	Przerwanie działania procesów, których o podanej nazwie (lub takich, których nazwa zawiera nazwę zawartą w poleceniu).
man *nazwa polecenia*	Wyświetla strony systemowego podręcznika.

Tabela 1: Polecenia powłoki

Więcej informacji o poszczególnych poleceniach a nawet o funkcjach (podprogramach) języka C można uzyskać za pomocą ostatniego polecenia wymienionego w tabeli. W szczególności polecenie `man man` wyświetli stronę podręcznika o poleceniu `man`. Inne informacje na temat posługiwania się podręcznikiem systemowym umieszczone są w rozdziale 3.

Z interpretera tekstowego możemy korzystać również w trybie graficznym, wystarczy uruchomić program nazywany terminalem (skrót znajduje się w menu „Programy” lub na pasku w postaci ikonki z czarnym ekranem).

2. Edytor VIM

Edytor VIM jest klonem edytora VI, jednego z pierwszych pełnoekranowych edytorów tekstu jakie pojawiły się dla systemu Unix. Mimo, że początkowo jego obsługa może wydawać się bardzo skomplikowana, to ten edytor jest bardzo ceniony w środowisku programistów, ze względu na liczne ułatwienia i dodatki. Jego obsługa nie wymaga używania myszy, choć nie jest to także zabronione. VIM działa zarówno w środowisku tekstowym jak i graficznym. Dostępny jest dla wielu systemów operacyjnych.

Edycję nowego lub już istniejącego pliku z kodem w języku C można rozpocząć wydając w powłoce polecenie `vim nazwa.c` lub `vi nazwa.c`. Praca z edytorem VIM odbywa się w kilku trybach. W tej instrukcji zostaną opisane dwa podstawowe oraz dwa dodatkowe tryby, które w pełni wystarczają do posługiwania się tym edytorem.

2.1. Tryb poleceń

Jest to domyślny tryb pracy edytora VIM, dostępny zaraz po jego uruchomieniu. W tym trybie dostępne są dwa rodzaje poleceń. Pierwsze uruchamiane są za pomocą odpowiednich kombinacji klawiszy, drugie dostępne są za pomocą linii poleceń edytora. Poniżej opisano najważniejsze z nich.
Kombinacje klawiszy:

h przemieszczenie kursora w lewo, przydatne, jeśli klawiatura nie dysponuje klawiszami kursora;

l przemieszczenie kursora w prawo, przydatne, jeśli klawiatura nie dysponuje klawiszami kursora;

j przemieszczenie kursora w dół, przydatne, jeśli klawiatura nie dysponuje klawiszami kursora;

k przemieszczenie kursora w górę, przydatne, jeśli klawiatura nie dysponuje klawiszami kursora;

u cofnięcie ostatniej operacji (ang. undo);

^ ustawienie kursora na pierwszym znaku w wierszu;

$ ustawienie kursora na ostatnim znaku w wierszu;

***** odnalezienie i zaznaczenie wszystkich wystąpień w tekście wyrażenia, które znajduje się pod kursorem;

. powtórzenie ostatniego polecenia, przydaje się szczególnie przy złożonych poleceniach;

yy zapamiętanie jednego wiersza tekstu (ang. copy), jeśli chcemy zapamiętać więcej wierszy wtedy wybieramy kombinację y*n*y lub *n*yy, gdzie *n* jest liczbą wierszy;

dd usunięcie pojedynczego wiersza tekstu, jeśli chcemy usunąć więcej wierszy postępujemy tak jak przy poleceniu yy;

p wklejenie (ang. paste) zapamiętanych lub usuniętych wierszy w miejsce, gdzie znajduje się kursor, jeśli chcemy tę operację wykonać kilka razy to przed wydaniem polecenia piszemy liczbę jego powtórzeń;

x usunięcie znaku znajdującego się „pod" kursorem, możemy tę operację powtórzyć tak jak operację wklejania,

dw usunięcie słowa, można tę operację powtórzyć tak jak yy lub dd;

yw skopiowanie słowa, można tę operację powtórzyć tak jak yy lub dd;

cw usunięcie słowa i przejście do trybu wstawiania, można tę operację powtórzyć tak jak yy lub dd;

i PRZEJŚCIE DO TRYBU WSTAWIANIA;

a przesunięcie kursora o jeden znak w prawo i przejście do trybu wstawiania;

o przesunięcie kursora o jeden wiersz w dół i przejście do trybu wstawiania;

r następny wprowadzony znak zastąpi ten, który znajduje się pod kursorem;

R przejście do trybu zastępowania;

J złączenie dwóch sąsiednich wierszy;

gg ustawienie kursora na początku tekstu (pierwszy wiersz);

G ustawienie kursora na końcu tekstu (ostatni wiersz);

=G ustawienie wcięć w kodzie źródłowym programu od miejsca, gdzie znajduje się kursor;

== ustawienie wcięcia bieżącego wiersza w kodzie źródłowym;

Linia poleceń:

:q zakończenie pracy edytora;

:q! zakończenie pracy edytora bez zapisywania edytowanego pliku;

:wq zapisanie pliku i zakończenie pracy edytora;

:x zapisanie zmian w pliku i zakończenie pracy edytora;

:w zapisanie pliku, po spacji można opcjonalnie podać nazwę pod jaką plik ma być zapisany;

:set tw=*n* ustalenie maksymalnej liczby znaków w wierszu; *n* jest liczbą;

/*fraza* wyszukanie wystąpienia *frazy* w tekście;

:s/*fraza1*/*fraza2* zastąpienie *frazy1 frazą2* jeśli kursor znajduje się w miejscu wystąpienia tej pierwszej,

:s/*fraza1*/*fraza2*/*g* jak wyżej, ale w całym wierszu;

:%s/*fraza1*/*fraza2* jak wyżej, ale zastępowane jest pierwsze wystąpienia *frazy1* we wszystkich wierszach i jest to proces automatyczny (nie trzeba „ręcznie” wyszukiwać frazy);

:%s/*fraza1*/*fraza2*/*g* jak wyżej, ale zastępowane jest każde wystąpienie *frazy1* w pliku;

:s powtórzenie ostatniego zastąpienia;

:syn on włączenie kolorowania składni kodu źródłowego (często jest ono domyślnie włączone);

:syn off wyłączenie kolorowania składni kodu źródłowego;

:set cindent włącza automatyczne formatowanie wcięć w kodzie źródłowym dla języka C (domyślnie często włączone);

:set smartindent włączenie automatycznego formatowania wcięć w kodzie źródłowym po przejściu kursorem do nowego wiersza (domyślnie często włączone);

:set autoindent włącza automatyczne ustawianie wcięcia w odniesieniu do poprzedniego wiersza (domyślnie często włączone);

:set wrap włącza zawijanie wierszy;

set nowrap wyłącza zawijanie wierszy;

:set number włącza numerowanie wierszy;

:set nonumber wyłącza numerowanie wierszy;

:set relativenumber włącza numerowanie wierszy względem bieżącego położenia kursora, pomocne przy kopiowaniu lub usuwaniu wielu wierszy kodu;

:set norelativenumber wyłącza numerowanie wierszy względem bieżącego położenia kursora;

:set ruler włącza „linijkę”, czyli wyświetlanie współrzędnych kursora po prawej stronie dolnego wiersza ekranu;

:*n* przejście do wiersza o numerze *n*;

:set paste włącza wierne kopiowanie fragmentów tekstu, kiedy do kopiowania używana jest mysz, działa po przejściu w tryb wstawiania lub edycji;

:set nopaste wyłącza wierne kopiowanie fragmentów tekstu, kiedy do kopiowania używana jest mysz;

:redo odwrotność cofnięcia;

:colo *nazwa* pozwala ustawić schemat kolorów edytora, *nazwa* oznacza nazwę schematu;

:tabnew tworzy nową zakładkę w edytorze, opcjonalnie może być podana nazwa pliku, którego zawartość ma być w zakładce wyświetlona;

:tabprev przejście do poprzedniej zakładki;

:tabnext przejście do następnej zakładki;

:e *nazwa* otwarcie pliku o podanej nazwie;

:sp podział poziomy ekranu na dwa równe okna, jeśli po poleceniu występuje nazwa pliku, to będzie on wyświetlany w jednym z tych okien;

:vsp jak wyżej, ale podział jest pionowy;

Ctrl+w przełączanie między oknami;

:! *polecenie* wykonanie polecenia powłoki systemowej, w ten sposób można np. kompilować i uruchamiać programy nie opuszczając edytora;

Działanie niektórych z poleceń `:set` można odwrócić stawiając na ich końcu znak wykrzyknika `(!)`. Wiele z nich można także wykonać automatycznie po uruchomieniu edytora lub można przypisać je określonym klawiszom (najczęściej klawiszom funkcyjnym). Taki efekt uzyskuje się edytując zawartość pliku `.vimrc`, który tworzony jest w katalogu głównym użytkownika (w Linuksie wszystkie pliki, których nazwa zaczyna się od kropki są ukryte). Przykład takiego pliku zawiera listing 1.

```
set nocp                              "wyłącz tryb kompatybilności z VI
set nowrap                            "wyłącz zawijanie wierszy
set autoindent shiftwidth=8           "ustaw głębokość wcięć na 8 znaków
                                      "i włącz automatyczne formatowanie
set cindent                           "włącz formatowanie kodu źródłowego C
set showcmd                           "pokaż komendy wpisywane w trybie edycji
set ruler                             "włącz wyświetlanie współrzędnych kursora
set tw=80                             "ustaw długość wiersza na 80 znaków
set showmatch                         "włącz parowanie nawiasów
syn on                                "włącz kolorowanie składni
color evening                         "włącz schemat kolorów o nazwie evening

"odwzoruj polecenia na klawisze:
map <F2> :set wrap!<cr>                 "F2 - włącz/wyłącz zawijanie wierszy
map <F3> :set number!<cr>               "F3 - włącz/wyłącz numeracje wierszy
map <F4> :set relativenumber!<cr>       "F4 - włącz/wyłącz numerację wierszy
                                        "względem bieżącego położenia kursora
map <F5> :set paste!<cr>                "F5 - włącz/wyłącz wierne kopiowanie tekstu
```

Listing 1: Przykładowa zawartość pliku `.vimrc`

2.2. Tryby wstawiania i zastępowania

Są to tryby w których wprowadzamy tekst do pliku. Różnią się tym, że w trybie wstawiania nowe znaki są dostawiane do istniejących, a w trybie zastępowania zastępują istniejące. Przydatne kombinację klawiszy to: `Ctrl+p` - uzupełnienie tekstu wzorcem, który występuje już wcześniej, `Ctrl+n` - uzupełnienie tekstu wzorcem, który występuje już później w tekście. W przypadku języka C te kombinacje mogą podpowiadać nazwy funkcji. Kombinacja klawiszy `Shift+k` powoduje wyświetlenie, o ile istnieje, strony podręcznika dla funkcji, której nazwa jest umieszczona pod kursorem. Powrót do trybu poleceń następuje przez naciśnięcie klawisza `Esc` (ang. escape).

2.3. Tryb wizualny

Ten tryb jest rzadko używany, ale czasem jest przydatny, jeśli trzeba przeprowadzić działania na kolumnach tekstu, a nie wierszach. Z trybu poleceń można przełączyć się do trybu wizualnego naciskając kombinację klawiszy `Ctrl+v`. Następnie można zaznaczyć kolumnę tekstu i po naciśnięciu klawisza `I` (duże i) wprowadzić znaki. Po powrocie do trybu poleceń (klawisz `Esc`) te znaki zostaną skopiowane do każdego wiersza znajdującego się w kolumnie. Ten tryb umożliwia np. komentowanie kilku wierszy kodu w jednym kroku.

3. Podręcznik `man`

Podręcznik `man` zawiera opisy poleceń powłoki, oraz funkcji języka C. Składnia wywołania jest następująca: `man nazwa_polecenia`. W przypadku, kiedy istnieje zarówno funkcja, jak i polecenie o takiej samej nazwie, to należy podać jeszcze stronę podręcznika, np.: `man 3 printf`. Aby uzyskać informację na temat wszystkich stron dotyczących danego zagadnienia należy zastosować opcję `-k`, np.: `man -k printf`. Jeśli chcemy, aby podręcznik `man` wyświetlił po kolei wszystkie strony dotyczące danego zagadnienia, to możemy użyć opcji `-a`, np.: `man -a printf`. Pracę z daną stroną kończymy wciskając klawisz `q`. Jeśli przeglądamy tylko jedną stronę podręcznika, to wciśnięcie tego klawisza spowoduje wyłączenie programu `man`. Jeśli chcemy znaleźć na stronie interesujące informacje, to możemy je wyszukać stosując te same polecenia, co w edytorze VIM: `/` oraz `?`.

4. Kompilator `gcc` i narzędzie `make`

Kompilator `gcc` jest tak naprawdę zbiorem kompilatorów stworzonym przez organizację GNU. Polecenie `gcc` wywołane z poziomu powłoki dla pliku z rozszerzeniem `.c` spowoduje uruchomienie kompilatora GCC dla języka `c`. Sposób takiego wywołania jest następujący:

```
gcc -Wall -o nazwa nazwa.c
```

Opcja `-Wall` oznacza, że kompilator będzie generował wszystkie ostrzeżenia. Ostrzeżenie jest informacją, że dany fragment kodu źródłowego może powodować problemy podczas działania programu, np. może być dwuznaczny i nie zadziałać tak, jak programista by sobie tego życzył. Ostrzeżenia nie powodują przerwania kompilacji, ale utworzony kod wynikowy może zawierać defekty powodujące błędy w czasie wykonania programu. Należy więc dążyć do całkowitego wyeliminowania ostrzeżeń poprzez poprawienie wskazanych miejsc w kodzie źródłowym. Jeśli kompilator znajdzie błąd w kodzie źródłowym, wówczas przerywa kompilację i nie powstaje kod wynikowy. Opcja `-o` powoduje nadanie plikowi wykonywalnemu określonej nazwy. Warto zwrócić uwagę, że w Linuksie (i ogólnie w Uniksach) plik wykonywalny nie ma rozszerzenia (nie jest ono obowiązkowe). Nie jest także wymagane, aby nazwa pliku z kodem źródłowym, za wyjątkiem rozszerzenia, pokrywała się z nazwą pliku pliku wykonywalnego, jednakże taka konwencja upraszcza rozpoznanie zależności między tymi plikami. Program uruchamiamy z linii poleceń następująco: `./nazwa`.

W przypadku bardzo rozbudowanych programów, składających się z wielu plików z kodem źródłowym, ręczne uruchamianie kompilatora byłoby bardzo czasochłonne. Programu `make` pozwala na zaoszczędzenie czasu. Wykonuje on kompilację programu lub programów na podstawie pliku konfiguracyjnego o nazwie

`makefile`[1], w którym programista zapisuje reguły kompilacji. Dzięki temu można ją wielokrotnie powtarzać wydając jedynie polecenie `make` w powłoce systemowej. Plik `makefile` ma określoną składnię, którą przedstawimy za pomocą przykładów. Załóżmy dla uproszczenia, że wszystkie pliki źródłowe, które chcemy skompilować znajdują się w tym samym katalogu, co plik `makefile`. Jeśli chcemy skompilować tylko jeden plik o nazwie `program.c`, to wystarczy, że zapiszemy w pliku `Makefile` taką regułę, jak w listingu 2.

```
1 rule:
2         gcc -Wall -o program program.c
```

Listing 2: Jedna z najprostszych postaci pliku `makefile`

Teraz, po wydaniu polecenia `make` wywołany zostanie kompilator według reguły `rule`. Proszę zwrócić uwagę, że polecenie wywołania kompilatora zostało zapisane w kolejnym wierszu po nazwie reguły i z wcięciem o szerokości jednego tabulatora. Jest to wymóg składni pliku `makefile`

Spróbujmy teraz zmienić nazwę reguły z `rule` na `program`, tak jak w listingu 3.

```
1 program:
2         gcc -Wall -o program program.c
```

Listing 3: Zmodyfikowana postać pliku `makefile`

Teraz, jeśli wydamy polecenie `make`, to otrzymamy komunikat „make: 'program' jest aktualne", co oznacza, że program `make` wykrył, iż istnieje taki plik i stwierdził, że nie jest potrzebna ponowna kompilacja. Jeśli chcemy ją jednak wykonać, to musimy wcześniej skasować plik `program`.

Zauważmy, że wykonanie kompilacji jest możliwe wtedy i tylko wtedy, gdy plik `program.c` istnieje. Możemy o tym poinformować program `make` zmieniając regułę kompilacji w pliku `makefile` tak, jak na listingu 4.

```
1 program: program.c
2         gcc -Wall -o program program.c
```

Listing 4: Dodanie zależności do reguły w pliku `makefile`

Dzięki dodaniu po dwukropku za nazwą reguły, w tym samym wierszu, w którym się ona znajduje, nazwy pliku z kodem źródłowym, powiadamiamy program `make`, że ten plik jest konieczny do wykonania wspomnianej reguły.

Często zdarza się, że w wyniku kompilacji powstaje mnóstwo plików, które później już nie są potrzebne. Ich ręczne usuwanie może być równie uciążliwe co ręczna kompilacja. Czynność tę możemy zautomatyzować dodając odpowiednią regułę w pliku `makefile`. Załóżmy, że chcemy mieć regułę, która będzie kasowała plik wykonywalny `program`. Możemy ją zapisać tak, jak to zostało pokazane w listingu 5.

[1] Plik ten może się również nazywać `Makefile`. Jeśli jednak program `make` napotka dwa pliki o takiej samej nazwie, ale jedna będzie się zaczynała wielką literą, a druga małą, to wybierze ten zaczynający się małą literą.

```
program: program.c
	gcc -Wall -o program program.c

clean:
	rm -f program
```

Listing 5: Reguła kasująca plik `program`

Zwróćmy uwagę, na pusty wiersz pomiędzy regułami. Informuję on program `make`, że oto kończy się jedna reguła i dalej może wystąpić inna. W przypadku listingu 5 jest to reguła `clean`. Po wydaniu polecenia `make` zostanie wykonana pierwsza napotkana w pliku `makefile` reguła, czyli `program`. Jeśli chcielibyśmy wykonać regułę `clean`, to jej nazwę musimy umieścić po nazwie polecenia, czyli `make clean`. Załóżmy, że chcielibyśmy, aby plik `program` był usuwany przed każdą kompilacją. Możemy to zrobić tak, jak na listingu 6.

```
rule: program.c
	rm -f program
	gcc -Wall -o program program.c

clean:
	rm -f program
```

Listing 6: Kasowanie pliku `program` przed kompilacją - wersja 1

Teraz reguła `rule` obejmuje dwa polecenia: skasowania pliku i kompilacji. Zauważmy jednak, że dwa razy powtarzamy ten sam zapis. Możemy tego uniknąć dzięki temu, że składania reguł pliku `makefile` przewiduje dodanie zależności nie tylko od istnienia plików, ale również od wykonania innych reguł. Ilustruje to listing 7.

```
rule: program.c clean
	gcc -Wall -o program program.c

clean:
	rm -f program
```

Listing 7: Kasowanie pliku `program` przed kompilacją - wersja 2

Czasem stajemy przed problemem równoczesnego skompilowania kilku programów. Aby zapisać taki wymóg w pliku `makefile` musimy utworzyć jako pierwszą pustą regułę, której wykonanie będzie zależało od reguł kompilujących wszystkie wymagane pliki, np. w taki sposób, jak to pokazano na listingu 8.

```
all: rule next_rule

rule: program.c
	gcc -Wall -o program program.c

next_rule: program2.c
	gcc -Wall -o program2 program2.c

clean:
	rm -f program
	rm -f program2
```

Listing 8: Podwójna kompilacja

Po wydaniu polecenia `make` zostanie wykonana reguła `all`. Proszę zwrócić uwagę, że jest ona pusta, tzn. ma tylko zależności od innych reguł, ale sama nie zawiera żadnych poleceń, od razu kończy się pustym wierszem. Możemy wykonywać także pozostałe reguły umieszczając w wierszu poleceń ich nazwy po nazwie polecenia `make`.

Składania pliku `makefile` przewiduje także dodawanie komentarzy. Komentarz to tekst rozpoczynający się znakiem `#`, i kończący znakiem końca wiersza. Dodatkowo w pliku `makefile` można używać także stałych. Zazwyczaj ich nazwy piszemy wielkimi literami. Przypisujemy im wartości za pomocą znaku `=`, a odwołujemy do nich za pomocą znaku dolara i nawiasów okrągłych. Przykładowo wartość stałej o nazwie `VAR` możemy uzyskać następująco `$(VAR)`. Użycie stałych ilustruje listing 9.

```
SRC = program.c
DST = program

rule:
	gcc -Wall -o $(DST) $(SRC)
```

Listing 9: Użycie stałych w pliku `makefile`

Na zakończenie warto wspomnieć o opcji `-f` polecania `make`. Pozwala ona nadać plikowi `makefile` dowolną nazwę i wskazać go jako ten, który zawiera odpowiednie reguły dla tego polecenia, np.: `make -f rules`.

5. Debugger gdb

Do usuwania defektów w programie może być pomocny debugger `gdb`. Aby się nim posłużyć musimy kompilując program dodać flagę `-g` do wywołania kompilatora `gcc`, a następnie wywołać polecenie: `gdb` *nazwa*, gdzie *nazwa* oznacza nazwę pliku wykonywalnego. Debugger `gdb` umożliwia wykonanie programu w trybie krokowym, a ponadto umożliwia interakcję z użytkownikiem. Tabela 2 zawiera listę najważniejszych poleceń tego programu.

Polecenie	Opis
`break`	Ustawia pułapkę w wybranym wierszu programu np. `break 6`.
`clear`	Usuwa pułapkę z określonego wiersza np. `clear 6`.
`delete`	Usuwa wszystkie pułapki.
`print`	Wypisuje wartość zmiennej, np. `print liczba`.
`step`	Wykonuje jeden wiersz programu.
`next`	Jak wyżej, ale nie wchodzi do funkcji.
`run`	Uruchamia program, musi być wykonane przed `step` i `next`. Należy także pamiętać o ustawieniu pułapki w odpowiednim wierszu.
`continue`	Kontynuuje wykonanie programu, po tym jak został on przerwany po napotkaniu pułapki.
`quit`	Zakończenie działania debuggera.

Tabela 2: Najważniejsze polecenia programu `gdb`

6. Zadania

1. Uruchom program vimtutor i zapoznaj się z działaniem edytora VIM.
2. Napisz program typu „hello world” w języku C i skompiluj go. Usuń wszelkie ostrzeżenia pojawiające się podczas kompilacji.
3. Napisz własny plik makefile, w którym użyjesz stałych i rozbudowanych reguł, a następnie przetestuj działanie polecenia `make`.
4. Napisz program zawierający prostą funkcję i pętlę `for`, w której ta funkcja będzie wywoływana. Skompiluj go z opcją `-g` i prześledź jego wykonanie za pomocą `gdb` (skorzystaj z pomocy w tym programie, wydając polecenie `help`).