Politechnika Wrocławska

Programowanie Systemowe

LABORATORIUM

Temat: **Zmienne automatyczne**

Hanna Kieszek – 283797

09.11.2025

W poprzedniej wersji tego sprawozdania została zauważona niespójność nazw folderu SRCDIR, w paru miejscach zamiast tego nazywa się on SRC_DIR. W tej wersji jest to ujednolicone.

1. Cel ćwiczenia

Celem ćwiczenia było zapoznanie się z działaniem zmiennych automatycznych i funkcji wildcard oraz patsubst.

2. Wykorzystane narzędzia

1. Debian
2. Terminal

3. Przebieg laboratorium

3.1. Stworzenie pliku testowego

Na samym początku zajęć został stworzony plik testowy Makefile2 w którym zostało przedstawione działanie poszczególnych zmiennych automatycznych. Zmienna automatyczna ma wartość która jest zależna od reguły w której się znajduje. Pierwsza zmienna $@ pozwala odnieść się do nazwy reguły. Druga zmienna $< odnosi się do pierwszej zależności znajdującej się po prawej stronie dwukropka. Natomiast zmienna automatyczna $^ odnosi się jednocześnie do wszystkich zależności po dwukropku.

Plik testowy wyglądał w następujący sposób:

```
all:  plik1 plik2 plik3 plik4
	echo $@
	echo $<
	echo $^
plik1:
	echo $<
plik2:

plik3:

plik4:
```

Output jaki został uzyskany w terminalu po wywołaniu make to:

```
echo

echo all
all
echo plik1
plik1
echo plik1 plik2 plik3 plik4
plik1 plik2 plik3 plik4
```

3.2. Modyfikacja pliku makefile z poprzednich laboratoriów

Do dalszej części laboratoriów został wykorzystany plik Makefile z poprzednich zajęć. Zostały w nim zamienione wybrane zmienne (te gdzie było to możliwe) na zmienne automatyczne tak aby program działał dokładnie w ten sam sposób co przed tą modyfikacją. Finalnie plik Makefile wyglądał w następujący sposób:

```
OUTDIR=build
PLIK=plik.c
SCRDIR=src
CC= gcc
TARGET=hello

all: $(TARGET)

clean:
	rm -rf $(OUTDIR)

$(TARGET):   $(OUTDIR) plik.o
	$(CC) $</plik.o -o $</$@

plik.o:   $(OUTDIR)
	$(CC) -c $(SRCDIR)/$(PLIK) -o $</$@

$(OUTDIR):
	mkdir $@
```

3.3. Stworzenie plików testowych i poznanie nowych funkcji makefile

Na potrzeby kolejnej części zadania zostały stworzone pliki main.c, hello.h a nazwa pliku HelloWorld w języku c została zamieniona z plik.c na hello.c. Wszystkie te pliki zostały zapisane w katalogu src.

W pliku testowym Makefile2 została usunięta jego treść aby użyć go do przetestowania nowych funkcji makefile. Pierwszą z nich jest funkcja wildcard, która pozwala na wyszukiwanie plików, na przykład w jakimś folderze, na podstawie danego wzorca. Druga funkcja to patsubst, która zamienia fragmenty nazw plików według podanego w niej wzorca. Poniżej zostały przedstawione schematy tych funkcji:

```
$(wildcard wzorzec)
$(patsubst wzorzec, zamiana, lista)
```

Do pliku Makefile2 została dodana zmienna SRCDIR - folder źródłowy, OUTDIR - folder docelowy, SRC - zmienna zawierająca wildcard, która reprezentowała wszystkie pliki .c z folderu

źródłowego oraz OBJ - zawierającą funkcję patsubst, która zamieniała rozszerzenie .c na .o plików ze zmiennej SRC, zachowując ich nazwę. Makefile2 ma postać:

```
SRCDIR=src
OUTDIR=build
SRC=$(wildcard $(SRCDIR)/*.c)
OBJ=$(patsubst $(SRCDIR)/%.c, $(OUTDIR)/%.o, $(SRC))

all:
	echo $(SRC)
	echo$(OBJ)
```

Znaki * oraz % zastępują dowolną wartość na miejscach w których się znajdują. Dzięki temu mogą zostać wyszukane wszystkie pliki z rozszerzeniem .c lub .o.

Po uruchomieniu make, w terminalu zostanie wyświetlone:

```
echo hello.c main.c
hello.c main.c
echo hello.o main.o
hello.o main.o
```

3.4. Zastosowanie wildcard i patsubst w pliku Makefile

Plik makefile został zmodyfikowany tak, aby po wywołaniu make kompilował on wszystkie pliki c w folderze src a następnie kompilował i linkował pliki obiektowe które powstały.

Ze zmiennych została usunięta zmienna PLIK natomiast zostały dodane SRC oraz OBJ które odpowiadają funkcją wildcard oraz patsubst. Zmodyfikowane zmienne zostały podane poniżej:

```
OUTDIR=build
SRCDIR=src
CC= gcc
TARGET=$(OUTDIR)/hello
SRC=$(wildcard $(SRCDIR)/*.c)
OBJ=$(patsubst $(SRCDIR)/%.c, $(OUTDIR)/%.o, $(SRC))
```

Reguła target oraz clean nie zostały zmienione:

```
all: $(TARGET)

clean:
	rm -rf $(OUTDIR)
```

Zmieniona reguła $(TARGET) przyjmuje tylko jedną zależność od zmiennej OBJ, która zawiera wszystkie pliki .o z katalogu build odpowiadające plikom c z katalogu src. Komenda zawarta w regule target odpowiada za skompilowanie i linkowanie plików obiektowych ze zmiennej OBJ. $(CC) - używany kompilator, $^ - wszystkie pliki z OBJ (gdyby zamiast $^ było $< zostałby użyty tylko pierwszy plik ze zmiennej OBJ a nie wszystkie), -o - opcja zapisywania, $@ - zmienna $(TARGET).

```
$(TARGET): $(OBJ)

	$(CC) $^ -o $@
```

Reguła plik.o została uogólniona dla wszystkich plików .o nadając jej nazwę $(OUTDIR)/%.o która oznacza wszystkie pliki obiektowe z folderu build (% oznacza dowolny ciąg znaków). Ma ona zależność do wszystkich plików c z folderu src, opisany konstrukcją $(SRCDIR)/%.. Oprócz tego na końcu tej linijki została dodana konstrukcja | $(OUTDIR) czyli zależność porządkową. Oznacza to, że zanim program wykona tę regułę sprawdzi czy build istnieje.

Ta reguła jest regułą wzorcową która opisuje jak ma być tworzony plik obiektowy z wykorzystaniem odpowiadającego mu pliku w języku c. Zawiera ona komendę kompilacji (-c oznacza tylko kompilowanie), która odnosi się do $< czyli pierwszego elementu po dwukropku oraz do $@ czyli elementu przed dwukropkiem. Plik src/%.c jest kompilowany do build/%.o gdzie procent oznacza dowolny ciąg znaków i jest taki sam dla obydwu plików. Ponieważ ta reguła iteruje po plikach c z folderu src, to wartość procenta będzie zmieniała się na nazwę kolejnych plików. Finalna postać reguły $(OUTDIR)/%.o znajduje się poniżej:

```
$(OUTDIR)/%.o: $(SRCDIR)/%.c    | $(OUTDIR)

	$(CC) -c $< -o $@
```

Reguła $(OUTDIR) tworzy katalog build.

```
$(OUTDIR):
	mkdir $@
```

Zaletą Makefile’a który został napisany w ten sposób jest to, że jeżeli jakiś plik c ulegnie zmianie i make wykryje, że ma on nowszy czas modyfikacji niż jego obiektowy odpowiednik to skompiluje jeszcze raz tylko plik który się zmienił a nie wszystkie od nowa. Finalna postać Makefile’a znajduje się poniżej:

```
OUTDIR=build
SRCDIR=src
CC= gcc
TARGET=$(OUTDIR)/hello
SRC=$(wildcard $(SRCDIR)/*.c)
OBJ=$(patsubst $(SRCDIR)/%.c, $(OUTDIR)/%.o, $(SRC))
```

```
all: $(TARGET)

clean:
	rm -rf $(OUTDIR)

$(TARGET): $(OBJ)

	$(CC) $^ -o $@

$(OUTDIR)/%.o: $(SRCDIR)/%.c    | $(OUTDIR)

	$(CC) -c $< -o $@

$(OUTDIR):
	mkdir $@
```