Politechnika Wrocławska

Programy pomocnicze: diff, make, systemy git, bazaar, debugger. Zarządzanie wersjami

wer. 10 z drobnymi modyfikacjami!

Wojciech Myszka

Katedra Mechaniki, Inżynierii Materiałowej i Biomedycznej

2023-06-19 08:01:22 +0200

HR EXCELLENCE IN RESEARCH

Co jest potrzebne programiście?

1. Umiejętność logicznego myślenia.

Co jest potrzebne programiście?

1. Umiejętność logicznego myślenia.
2. Ukończone kursy dokształcające.

Co jest potrzebne programiście?

1. Umiejętność logicznego myślenia.
2. Ukończone kursy dokształcające.
3. Motywacja do pracy.

Co jest potrzebne programiście?

1. Umiejętność logicznego myślenia.
2. Ukończone kursy dokształcające.
3. Motywacja do pracy.
4. Komputer.

Co jest potrzebne programiście?

1. Umiejętność logicznego myślenia.
2. Ukończone kursy dokształcające.
3. Motywacja do pracy.
4. Komputer.
5. Zadanie.

Co jest potrzebne programiście?

1. Umiejętność logicznego myślenia.
2. Ukończone kursy dokształcające.
3. Motywacja do pracy.
4. Komputer.
5. Zadanie.
6. Kompilator (program, który kod źródłowy przetłumaczy na na kod maszynowy).

Politechnika Wrocławska

Co jest potrzebne programiście?

1. Umiejętność logicznego myślenia.
2. Ukończone kursy dokształcające.
3. Motywacja do pracy.
4. Komputer.
5. Zadanie.
6. Kompilator (program, który kod źródłowy przetłumaczy na na kod maszynowy).
7. Jakiś edytor (program pozwalający na wygodne wpisywanie kodu źródłowego).

Jak to robiono kiedyś? I

1. Czas komputera był drogi, a dostęp do niego limitowany.
2. Dostęp on-line był wyjątkiem.
3. Gdy algorytm był gotowy — rozpoczynało się jego kodowanie (na papierze).
4. Następnie przenoszono kod na nośnik maszynowy (taśma papierowa, karty perforowane, taśma magnetyczna).
5. Nośnik z kodem źródłowym (w centrum obliczeniowym) był czytany przez kompilator, który tłumaczył na postać maszynową; w razie błędów były one zaznaczane na wydruku.
6. W przypadku błędów formalnych — koder poprawiał je i zaczynał cykl od początku.
7. Gdy błędów nie było — następowało uruchomienie programu (na dostarczonych danych).

Jak to robiono kiedyś? II

8. Wyniki otrzymywał programista i sprawdzał czy są zgodne z oczekiwaniami. Jeżeli nie — rozpoczynała się żmudna analiza algorytmu. Program był uruchamiany po raz kolejny na danych testowych. Żeby ułatwić sobie pracę dodawano „wydruki kontrolne”.
9. I tak do skutku.

Praca w środowisku interaktywnym

1. Najpierw odpadły „nośniki maszynowe” (choć w okresie przejściowym znacznie taniej było wpisać kod na prymitywnym urządzeniu na kartach lub tasiemce papierowej niż blokować dostęp do deficytowego terminala komputera i linii telekomunikacyjnej.
2. Pojawiły się środowiska ułatwiające uruchamianie programów w trybie interaktywnym.
3. Pojawiły się języki konwersacyjne.

Jak jest dziś?

Ci z Państwa, którzy programują — wiedzą doskonale. Inni...

Politechnika Wrocławska

Jak jest tworzony duży program?

1. Podzielony jest na kawałki (moduły obejmujące pewne dobrze zdefiniowane „całości").
2. Grupy modułów tworzące pewne całości — grupowane są w „biblioteki".
3. Wspólne definicje grupowane są w „plikach nagłówkowych".

Politechnika Wrocławska

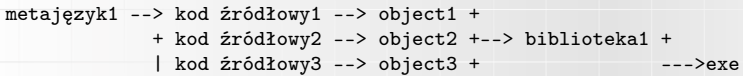
Kompilacja

Tworzenie programu jest procesem wieloetapowym.

1. Fragmenty programu mogą być pisane w jakimś **metajęzyku**, który tłumaczony jest na kod źródłowy.
2. Program w kodzie źródłowym tłumaczony jest na postać wynikową (zazwyczaj nazywa się to *object*).
3. Niektóre moduły wynikowe grupowane są w biblioteki.
4. Moduły wynikowe i biblioteki używane są do budowy programu wykonywalnego.

Politechnika Wrocławska

Schemat



Politechnika Wrocławska

Program make

Jakakolwiek zmiana w plikach z kodem metajęzyka, plikach nagłówkowych lub z kodem źródłowym wymaga przekompilowania części lub wszystkich plików.
Gdy projekt jest bardzo duży — problem jest bardzo poważny.
Wymyślono program make pozwalający ułatwiający programiście życie.
Program korzysta ze specjalnego pliku (tradycyjnie nazywa się on Makefile) który opisuje strukturę projektu.

Makefile

Plik Makefile definiuje czynności jakie należy wykonać aby z pliku jednego typu uzyskać plik wynikowy oraz zależności pomiędzy plikami.

```
metajęzyk1 --> kod źródłowy1 --> object1 +
             + kod źródłowy2 --> object2 +--> biblioteka1 +
             | kod źródłowy3 --> object3 +                ---->exe
             + kod źródłowy4 --> object4 --------------->  +
pliki nagł---+
```

W naszym przypadku exe zależy od biblioteka1 i object4. biblioteka1 zależy od object1, object2, object3.
object1 zależy od kod źródłowy1.
kod źródłowy1 zależy od metajęzyk1.
object2 zależy od kod źródłowy2
object3 zależy od kod źródłowy3
kod źródłowy2 zależy plik nagł

Politechnika Wrocławska

Przykładowy plik Makefile

albo raczej jego idea

```
exe:    biblioteka1, object4
        link -o exe object4 -L biblioteka1

biblioteka1:    object1, object2, object3
                ar -o biblioteka1 object1 object2 object3

object1:    kod źródłowy1
            kompiluj kod źródłowy1

object2:    kod źródłowy2, plik nagł
            kompiluj kod źródłowy2

object3:    kod źródłowy3, plik nagł
            kompiluj kod źródłowy3

object4:    kod źródłowy4, plik nagł
            kompiluj kod źródłowy4

kod źródłowy1:  metajęzyk1
                metakompiluj metajęzyk1 -o kod źródłowy1

all:    exe

clean:
        kasuj object* biblioteka1 exe plik źródłowy1
```

Politechnika Wrocławska

Metajęzyki

1. Metajęzyki to języki jeszcze wyższego poziomu niż języki typu C, Pascal, Fortan.
2. Pozwalają one za pomocą stosunkowo prostych zależności (i bez przejmowania się szczegółami) stworzyć kod źródłowy programu.
3. Przykład. Chcemy napisać program, który będzie rozróżniał *liczby* od *słów*.
   - Najpierw precyzyjnie musimy zdefiniować co to jest **liczba**. Liczba to jedna lub więcej cyfr z zakresu od 0 do 9. W kategoriach wyrażeń regularnych (regułowych) można to zapisać tak: [0123456789]+ lub [0-9]+ (plus oznacza tu **jedno** lub więcej powtórzeń).
   - Natomiast **słowo** to jeden lub więcej znaków, z których pierwszy jest literą i który nie zawiera odstępów i innych znaków specjalnych. Zapisujemy to tak: [a-zA-Z][a-zA-Z0-9]* (gwiazdka oznacza tu **zero** lub więcej powtórzeń.

Metajęzyki

flex

Idea programu jest następująca:

```
%%
[+-]{0,1}[0123456789]+          printf("NUMBER\n");
[a-zA-Z][a-zA-Z0-9]*      printf("WORD\n");
[ \t]+        printf("SPACE\n");
%%
```

i sprowadza się do następującego: „jak zobaczysz liczbę — wypisz **liczba**, jak zobaczysz wyraz — wypisz **wyraz**”

A sam program niewiele bardziej skomplikowany:

```
%{
#include <stdio.h>
%}

%%
[0123456789]+            printf("NUMBER\n");
[a-zA-Z][a-zA-Z0-9]*     printf("WORD\n");
[ \t]+   printf("SPACE\n");
%%
```

Kompilacja:

```
flex -o test.c test.l
~/c$ ls -l test.c
-rw-r--r-- 1 myszka myszka 41749 2008-05-26 15:50 test.c
gcc test.c -o test -ll
```

Uruchomienie

```
~/c$ ./test
ala ma 3 koty
WORD
 WORD
 NUMBER
 WORD
```

Politechnika Wrocławska

Bardziej skomplikowany przykład: kalkulator

1. Chodzi mi o pokazanie jednie pewnej idei, a nie wnikanie w głębokie szczegóły.
2. Kalkulator składa się z dwu części:
   2.1 wprowadzanie danych
   2.2 przetwarzanie danych
3. Do zbudowania analizatora danych wykorzystamy program **lex** (lub jego darmowy odpowiednik **flex**)
4. Do zbudowania części przetwarzającej dane wykorzystamy program **yacc** (lub jego darmowy odpowiednik **bison**)

Kalkulator I

Wprowadzanie danych

```
    /* calculator #1 */
%{
    #include "y.tab.h"
    #include <stdlib.h>
    void yyerror(char *);
%}

%%

[0-9]+      {
                yylval = atoi(yytext);
                return INTEGER;
            }
```

Kalkulator II

Wprowadzanie danych

```
[-+\n]      { return *yytext; }

[ \t]       ;       /* skip whitespace */

.           yyerror("Unknown character");

%%

int yywrap(void) {
    return 1;
}
```

Kalkulator I

Przetwarzanie

```
%{
    #include <stdio.h>
    int yylex(void);
    void yyerror(char *);
%}

%token INTEGER

%%

program:
        program expr '\n'         { printf("%d\n", $2); }
        |
```

Kalkulator II

Przetwarzanie

```
        ;

expr:
        INTEGER
        | expr '+' expr          { $$ = $1 + $3; }
        | expr '-' expr          { $$ = $1 - $3; }
        ;

%%

void yyerror(char *s) {
    fprintf(stderr, "%s\n", s);
}
```

Kalkulator III

Przetwarzanie

```
int main(void) {
    yyparse();
    return 0;
}
```

Politechnika Wrocławska

Czterodziałaniowy kalkulator z nawiasami I

Dane

```
%{
#include <stdlib.h>
#include <stdio.h>
#include "calc1.h"
void yyerror(char*);
extern int yylval;
%}
%%
[ \t]+      ;
[0-9]+      {yylval = atoi(yytext);
             return INTEGER;}
[-+*/]      {return *yytext;}
"("         {return *yytext;}
```

Czterodziałaniowy kalkulator z nawiasami II

Dane

```
")"         {return *yytext;}
\n          {return *yytext;}
.           {char msg[25];
             sprintf(msg,"%s␣<%s>",
                  "invalid␣character",yytext);
             yyerror(msg);}
```

Czterodziałaniowy kalkulator z nawiasami I

Przetwarzanie

```
%{
#include <stdlib.h>
#include <stdio.h>
int yylex(void);
#include "calc1.h"
%}
%token INTEGER
%%
program:
        line program
        | line

line:
```

Czterodziałaniowy kalkulator z nawiasami II

Przetwarzanie

```
        expr '\n' { printf("%d\n",$1); }
        | '\n'

expr:
        expr '+' mulex { $$ = $1 + $3; }
        | expr '-' mulex { $$ = $1 - $3; }
        | mulex { $$ = $1; }

mulex:
        mulex '*' term { $$ = $1 * $3; }
        | mulex '/' term { $$ = $1 / $3; }
        | term { $$ = $1; }
```

Czterodziałaniowy kalkulator z nawiasami III

Przetwarzanie

```
term:
      '(' expr ')' { $$ = $2; }
      | INTEGER { $$ = $1; }
%%
void yyerror(char *s)
{
 fprintf(stderr,"%s\n",s);
 return;
}
int main(void)
{
 /*yydebug=1;*/
 yyparse();
```

Czterodziałaniowy kalkulator z nawiasami IV

Przetwarzanie

```
    return 0;
}
```

Narzedzia pomocnicze

1. **diff** program pozwalający porównywać dwa pliki źródłowe
2. **patch** program pozwalający na podstawie różnic (raportowanych przez diff) wprowadzić poprawki do plików źródłowych. Pozwala to rozpowszechniać znacznie mniejsze pliki.
3. **git**, **svn**, **bazaar**, **rcs**, **cvs**,... — systemy zarządzania wersjami.
4. **debuger** — program pozwalający uruchamiać programy krok po kroku, sprawdzać zawartość zmiennych,...

Politechnika Wrocławska

configure

- Pliki dla programu make mogą być przygotowywane ręcznie.
- Gdy tworzymy wersję programu która powinna być zbudowana w (nieomal) dowolnym środowisku, może się okazać, że budowa programu wymaga spełnienia pewnych dodatkowych zależności (które nie są spełnione standardowo).
- W takim przypadku warto skorzystać z dodatkowych narzędzi pomocniczych. Jednym z nich jest **Autotools**.
- Narzędzie to składa się z szeregu skryptów, które po uruchomieniu sprawdzają w jakim środowisku budowane jest oprogramowanie, starają się zidentyfikować zależności od dodatkowych pakietów oprogramowania (bibliotek) i tworzą skrypt (program), który uruchomiony na systemie docelowych sprawdza czy wszystkie zależności są spełnione.

Politechnika Wrocławska

Zarządzanie wersjami I

1. Wszędzie tam, gdzie pracuje się na plikach tekstowych warto rozważyć użycie jednego z programów do zarządzania wersjam.
2. Ich idea polega na tym, że można zrobić „migawkę" aktualnej wersji plików.
   Migawki przechowywane są zazwyczaj w ukrytych katalogach, więc trudniej je „niechcęcy" zniszczyć.
3. W przypadku gdy coś poszło nie tak — zawsze można cofnąć się do którejś z poprzednich wersji.
4. W przypadku kodu jest to bardzo istotne:
   - mamy działającą wersję,
   - piszemy nową — kod się „rozgałęzia"
   - musimy dokonywać zmian w wersji działającej (usówanie błędów)
   - w końcu trzeba obie wersje połączyć — uwzględniając naniesone poprawki do wersji „starej" i zmiany wprowadzone w wersji „nowej"

Zarządzanie wersjami II

5. W takich sytacjach te oprogramowanie bardzo ułatwia życie.
6. Korzystam na codzień do nadzorowania zmian w „kodzie źródłowym" slajdów i innych materiałów dydaktycznych.

Politechnika Wrocławska

Repozytoria sieciowe

- Programy zarządzania wersjami (**git**, **bazaar**,...) pozwalają przechowywać informacje o zmianach w kodzie na zewnętrzych serwerach
- Jednymi z najpopularniejszych takich repozytoriów jest github; mniej popularny jest launchpad.