INFORMATIKA

CSS preprocesor SASS

MARTIN TRNEČKA

Přírodovědecká fakulta UP, Olomouc

1. Úvod

Výuka tvorby webových stránek, zejména výuka základů tvorby webového front-endu, kterou budeme v tomto textu adresovat, je nedílnou součástí výuky informatiky na základních školách, středních školách a gymnáziích. Navíc, díky tomu, že je dnes tvorba webových stránek běžnou a často žádanou dovedností, setkáme se s ní i v řadě výběrových seminářů a zájmových kroužků.

Vzhledem k běžně vyhrazené časové dotaci obvykle není možné se této problematice ve výuce věnovat tak, aby studenti mohli tvořit své vlastní reálné projekty. Výuka je mnohdy redukována pouze na představení základní syntaxe HTML, CSS a případně jazyka JavaScript, bez hlubšího pochopení.

Přestože to na první pohled nemusí být patrné, nejkomplikovanější je v tomto ohledu technologie CSS, která slouží pro vizualizaci HTML elementů. Psaní správného CSS kódu vyžaduje znalosti celé řady principů, které typicky nejsou ve výuce diskutovány, například: pravidlo kaskády, specifičnost, dědičnost, box-model, atomické CSS, CSS metodiky (OOCSS, BEM, SUIT CSS, ITCSS) a mnohé další. Takováto výuka je odtržena od reality, jelikož vytvořit komplexní a dobře fungující webovou stránku bez pochopení výše uvedeného je velmi obtížné. Například bez respektování pravidla kaskády či použití CSS metodik bude výsledný CSS kód vždy nepřehledný, obtížně udržovatelný a redundantní, nemluvě o tom, že napsat takový kód zabere paradoxně více času.

Ačkoliv by zařazení uvedených principů do výuky bylo řešení, je třeba si uvědomit, že je to ve většině případů časově velmi obtížně zvládnutelné. Tyto principy si weboví vývojáři osvojují roky a jejich správná výuka vyža-

duje nemalé zkušenosti. Pokud vyučující těmito zkušenostmi nedisponuje, může snadno u studentů vytvořit nesprávné programátorské návyky, které bude třeba později pracně odbourávat.

Nabízí se tedy otázka, co ukázat studentům, kteří se o danou problematiku zajímají a nespokojí se základy syntaxe CSS. Jednou z možností jsou *CSS preprocesory*. Ty přináší značné rozšíření základní syntaxe a umožňují studentům se nad tvorbou CSS kódu více zamyslet a vytvářet výrazně kvalitnější CSS kód i bez znalostí CSS metodik a dalších pokročilejších záležitostí.[1)] Navíc CSS preprocesory obsahují jednoduché programátorské konstrukce, které studenti využijí i při osvojování algoritmizace a programování. V následujícím textu si vysvětlíme vztah CSS a CSS preprocesorů, představíme CSS preprocesor SASS a ukážeme několik jednoduchých úkolů, které by spolu s výkladem potřebné teorie mohly být náplní několika vyučovacích hodin ať už běžné výuky nebo, a to je pravděpodobnější, výběrového semináře. V textu předpokládáme základní znalosti technologie CSS a HTML.

2. CSS preprocesory

Technologii CSS představil v roce 1994 norský výzkumník Håkon Wium Lie.[2)] O dva roky později, v roce 1996, se zrodil první standard této technologie, který byl zároveň prvním webovým standardem konsorcia W3C (World Wide Web Consortium), jenž dodnes standardizuje webové technologie. Zatímco možnosti, a stejně tak i potřeby, vizualizovat HTML elementy se od té doby velmi vyvinuly, samotná technologie CSS se příliš neměnila.[3)]

Ačkoliv stálost a jednoduchost CSS technologie měly své výhody, v době největšího rozvoje vizualizace webových stránek,[4)] se ukázaly jako zásadní omezení. Weboví vývojáři potřebovali pokročilejší konstrukce, jako například proměnné či funkce, které by jim usnadnili, psaní CSS kódu. Reakcí na toto ze strany W3C nevyslyšené volání byl příchod CSS preprocesorů.

[1)] Dodejme, že CSS preprocesory nejsou náhradou za zmíněné znalosti. Později ukážeme, že CSS preprocesory generují CSS kód. Kvalita výsledného kódu je pořád ovlivněna našimi znalostmi, ale CSS preprocesory umožňují kód lépe strukturovat, vyhnout se redundanci a vytvářet znovupoužitelné části kódu.

[2)] Håkon Wium Lie byl spolupracovníkem Tima Berners-Leeho, kterého netřeba představovat, a Roberta Cailliaua, kteří stojí za vznikem služby WWW.

[3)] Tím máme na mysli, že postupně přibývaly nové vlastnosti, ale syntaxe zápisu CSS pravidel se kromě několika drobných změn prakticky nezměnila.

[4)] Roky 2004–2014, tedy období vývoje dnes již neexistujícího HTML 5.

CSS preprocesory jsou nástroje, které umožňují převést (transpilovat) kód preprocesoru, zapsaný v jednoduchém programovacím jazyku (jazyk preprocesoru), do CSS kódu. Jazyk preprocesoru disponuje konstrukcemi pro řízení běhu programu, jako například podmínky a cykly, dále obsahuje funkce, umožňuje vytvářet znovupoužitelné části CSS kódu, používat plnohodnotnou dědičnost (nikoliv pouze dědičnost na úrovni HTML) a významně rozšiřuje syntaktické možnosti zápisu CSS pravidel.

Preprocesory získaly obrovskou popularitu mezi webovými vývojáři a postupem času se staly běžnou webovou technologií.[5)] Na toto později, poněkud neochotně, zareagovalo W3C a nastartovalo rozvoj, z jejich pohledu dokončené, technologie CSS jako takové.[6)] Postupně se v CSS objevily proměnné, funkce `calc()`, vnořování a další prvky, se kterými přišly preprocesory. Tímto rozvojem byl význam preprocesoru mírně oslaben. Přesto jsou dnes možnosti preprocesorů daleko nad možnostmi běžně podporovaného CSS. Navíc preprocesory umožňují generovat CSS kód, což pravděpodobně nikdy nebude pomocí běžného CSS možné.

3. Preprocesor SASS

CSS preprocesorů existuje hned několik, mezi ty nejznámější patří například SASS, LESS a Stylus. Dříve se mezi nimi vedla dlouhá válka o post nejlepšího, respektive nejpoužívanějšího, preprocesoru, kterou po letech bojů vyhrál preprocesor SASS (Syntactically Awesome Style Sheets), jenž je volně dostupný na stránce `sass-lang.com`. Vzhledem k tomu, že se dnes ostatní preprocesory prakticky nepoužívají, zaměříme se právě na něj.

3.1. Instalace

Instalace preprocesoru SASS je poměrně jednoduchá. Existuje celá řada nástrojů a rozšíření do různých editorů, které SASS obsahují, případně je možné jej nainstalovat pomocí nástroje `npm` následujícím příkazem.

```
npm install -g sass
```

[5)] Zajímavé je, že se ale nikdy nestaly základní technologií. Prohlížeče je nikdy nezačaly nativně podporovat.

[6)] Nutno dodat, že W3C mělo poměrně omezené možnosti. Kompletně změnit CSS by bylo vzhledem k rozšířenosti této technologie velmi komplikované. Navíc CSS již bylo v této době rozděleno do nezávislých modulů, což komplikuje jeho vývoj, jelikož neexistuje žádný modul popisující syntaxi CSS jako celek.

Rovněž je možné použít stránku sass-lang.com/playground nebo službu codepen.io. Ty jsou pro výukové účely nejvhodnější.

Pro úplnost dodejme, že existuje několik implementací SASS: Ruby Sass, LibSass a Dart Sass. První je již nevyvíjená, druhá je stále funkční, ale je považována za zastaralou. V nových projektech by měla být používána Dart Sass, kterou budeme v tomto textu uvažovat.

3.2. Základy SCSS

Preprocesor SASS umožňuje zapisovat kód preprocesoru pomocí dvou syntaxí: starší SASS, která se dnes již prakticky nepoužívá a novější SCSS (Sassy CSS). Stejné jméno syntaxe preprocesoru a preprocesoru často působí nedorozumění. Přestože je stále možné používat SASS syntaxi, pokud dnes někdo řekne, že používá SASS má na mysli SCSS syntaxi.

Syntaxe SCSS je plně kompatibilní se syntaxí CSS, ale navíc rozšiřuje stávající konstrukty CSS (například přidává nová at-pravidla) a přináší zcela nové konstrukty do CSS (například zanořování). Samotný jazyk CSS preprocesoru je poměrně jednoduchý a omezený programovací jazyk.[7] Obsahuje běžné operátory, umožňuje práci s řetězci a čísly. V následující části si jej stručně představíme.

Proměnné

Jedním z hlavních přínosů CSS preprocesorů bylo zavedení proměnných, které v CSS znatelně chyběly. V SCSS se proměnné se zapisují s prefixem `$`. Ukázka následuje.

SCSS kód

```scss
$color-main: #58508d;
$color-contrast: #ffa600;

.box {
  background-color: $color-main;
  color: $color-contrast;
}

.box--inverse {
  background-color: $color-contrast;
  color: $color-main;
}
```

[7] Nenajdeme zde nic, co bychom neviděli v jiných programovacích jazycích.

Vygenerovaný CSS kód

```
.box {
  background-color: #58508d;
  color: #ffa600;
}

.box--inverse {
  background-color: #ffa600;
  color: #58508d;
}
```

Ve výše uvedeném SCSS kódu jsme deklarovali globální proměnou. Proměnné definované v deklaračním bloku (deklarační blok je v CSS vymezen složenými závorkami) jsou lokální v daném deklaračním bloku.

Později se v CSS objevila nativní podpora proměnných. Z programátorského hlediska jsou ale tyto proměnné poněkud odlišené od proměnných v běžných programovacích jazycích. CSS proměnné jsou deklarativní. Změna jejich hodnoty změní i předchozí použití dané proměnné. Proměnné v SCSS jsou imperativní, tedy změna jejich hodnoty neovlivní jejich předchozí použití. Dalším zásadním rozdílem je, že preprocesor při transpilaci proměnné nahrazuje jejich hodnotami. V případě nativních CSS proměnných, jsou tyto proměnné uloženy ve výstupním CSS.[8)]

Komentáře

V SCSS je možné používat kromě běžných CSS komentářů (zapisovaných pomocí `/* */`) i komentáře uvozené symboly `//`. Běžné CSS komentáře budou ve výsledném CSS kódu, zatímco komentáře začínající symboly `//` budou odstraněny.[9)]

SCSS kód

```
// komentář pro preprocesor
/* komentář pro CSS */
```

Vygenerovaný CSS kód

```
/* komentář pro CSS */
```

[8)] Prohlížeč je tedy musí zpracovat. V případě použití preprocesoru má prohlížeč méně práce, což má za následek zvýšení rychlosti.

[9)] Jedná se o komentáře zdrojového kódu preprocesoru, ve výsledném CSS nemají význam.

Interpolace

Výsledek vyhodnocení SCSS kódu může být snadno předán do CSS pomocí `#{}`. Toho se využívá zejména v případě, kdy chceme CSS vlastnost určit pomocí proměnné tak, jak je ukázáno níže.

SCSS kód

```
$where: top;

.box {
  #{$where}: 1em;
}
```

Vygenerovaný CSS kód

```
.box {
  top: 1em;
}
```

Nesting

Jedním ze zásadních vylepšení CSS syntaxe, které přinesly CSS preprocesory, je *nesting* (vnořování).Ten umožňuje zanořovat CSS pravidla. Zanoření umožňuje zkrátit zápis CSS selektorů a zpřehlednit kód. Následující kód generuje (klasické) třídy pro navigační menu.

SCSS kód

```
.nav {
  height: 3em;

  ul {
    margin: 0;
    padding: 0;

    li {
      list-style-type: none;

      a {
        text-decoration: none;
      }
    }
  }
}
```

Vygenerovaný CSS kód

```
.nav {
  height: 3em;
}

.nav ul {
  margin: 0;
  padding: 0;
}

.nav ul li {
  list-style-type: none;
}

.nav ul li a {
  text-decoration: none;
}
```

Pomocí *rodičovského selektoru*, který se zapisuje symbolem `&`, je možné se odkázat na rodiče v hierarchii, která je vytvořena zanořením. Například.

SCSS kód

```
a {
  text-decoration: underline;

  &:hover {
    text-decoration: none;
  }
}
```

Vygenerovaný CSS kód

```
a {
  text-decoration: underline;
}

a:hover {
  text-decoration: none;
}
```

Při použití je rodičovský selektor nahrazen rodičem. Je tedy možné jej použít i jako hodnotu funkce.

SCSS kód

```
.text-normal {
  color: black;

  :not(&) {
    color: red;
  }
}
```

Vygenerovaný CSS kód

```
.text-normal {
  color: black;
}

:not(.text-normal) {
  color: red;
}
```

Nesting je možné využít i pro zkrácení zápisu více vlastností se stejným prefixem. Syntaxe se ale v tomto případě mírně liší.

SCSS kód

```
body {
  font: {
    family: Arial;
    size: 1em;
    style: italic;
  }
}
```

Vygenerovaný CSS kód

```
body {
  font-family: Arial;
  font-size: 1em;
  font-style: italic;
}
```

Pro úplnost dodejme, že je možné pomocí at-pravidla `@at-root` potlačit nesting a zpřístupnit hlavní, nezanořenou úroveň.

Dědičnost

SCSS syntaxe umožňuje používat dědičnost.[10)] Pomocí at-pravidla

[10)] Označení dědičnost je v tomto případě mírně zavádějící, ale běžně se používá.

`@extend` je možné zdědit CSS vlastnosti.

SCSS kód

```
.box {
  border-width: 1px;
  border-style: solid;
  border-color: black;
}

.box--emphasize {
  @extend .box;
  border-color: red;
}
```

Vygenerovaný CSS kód

```
.box, .box--emphasize {
  border-width: 1px;
  border-style: solid;
  border-color: black;
}

.box--emphasize {
  border-color: red;
}
```

V některých situacích je žádoucí, aby ve vygenerovaném CSS nefigurovala pravidla, ze kterých bylo děděno. K tomu je možné využít symbol `%`.[11)]

SCSS kód

```
%box {
  border-width: 1px;
  border-style: solid;
}

.box--normal {
  @extend %box;
  border-color: black;
}

.box--emphasize {
  @extend %box;
  border-color: red;
}
```

[11)] Uvedený kód je pouze demonstrační. Z pohledu BEM metodiky, která je zde použita, by bylo lepší jej řešit bez dědičnosti.

Vygenerovaný CSS kód

```
.box--emphasize, .box--normal {
  border-width: 1px;
  border-style: solid;
}

.box--normal {
  border-color: black;
}

.box--emphasize {
  border-color: red;
}
```

Mixiny

Mixiny umožňují vytvářet opakovaně použitelné části CSS kódu. Pro definici mixiny se požívá at-pravidlo `@mixin` a pro její použití `@include`.

SCSS kód

```
@mixin flex-center {
  display: flex;
  justify-content: center;
  align-items: center;
}

.box {
  @include flex-center;
}
```

Vygenerovaný CSS kód

```
.box {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

Mixiny mohou mít i parametry, kterým je možné určit výchozí hodnoty. Argumenty je možné předávat pozičně, ale i jménem parametru.[12)]

[12)] Analogicky jako tomu je například v jazyce Python.

SCSS kód

```scss
@mixin flex-center($flex-direction: row) {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: $flex-direction;
}

.box--row {
  @include flex-center();
}

.box--column {
  @include flex-center(column);
}

.box--column {
  @include flex-center($flex-direction: column);
}
```

Vygenerovaný CSS kód

```css
.box--row {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: row;
}

.box--column {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}

.box--column {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}
```

Mixiny mohou mít i proměnlivý počet argumentů. Příklad ukážeme později.

Funkce

Stejně jako v jiných programovacích jazycích je možné v SCSS používat funkce. Pro definici funkce se požívá at-pravidlo `@function`.[13)]

[13)] Upozorněme, že SASS neumí počítat s různými jednotkami. Pokud bychom volání

SCSS kód

```scss
$base: 1em;

@function add-m($x: 1em) {
  @return $base + $x
}

.box {
  margin: add-m(2em);
}
```

Vygenerovaný CSS kód

```css
.box {
  margin: 3em;
}
```

V některých případech je žádoucí, aby argumenty předané mixinám a funkcím splňovaly určité podmínky. At-pravidla `@error` a `@warn` umožňují generovat chybu nebo varování při transpilaci. Pro testování zda je, či není splněna daná podmínka slouží podmínky, které si nyní ukážeme.

Konstrukce pro řízení běhu programu

SASS umožňuje využívat běžné konstrukce pro řízení běhu programu jako jsou podmínky a cykly. Pro zápis podmínek se využívají at-pravidla `@if` a `@else`. Příklad použití následuje.

SCSS kód

```scss
@mixin theme-colors($light-theme: true) {
  @if $light-theme {
    background-color: $light-background;
    color: $light-text;
  } @else {
    background-color: $dark-background;
    color: $dark-text;
  }
}
```

V SCSS existuje analogie ternárního operátoru.[14)]

funkce na řádku 8 nahradili `add-m(2em)` dojde k chybě. V CSS nativní funkce `calc()` takové omezení nemá. V SASS je možné toto omezení částečně obejít pomocí vestavěných modulů.

[14)] V ukázce se využívá toho, že pokud je v SCSS nastavena vlastnost na hodnotu

SCSS kód

```
$show-border: false;

.box {
  padding: 1em;
  border: if($show-border, 1px solid black, null);
}
```

Vygenerovaný CSS kód

```
.box {
  padding: 1em;
}
```

Pro zápis cyklů se používají at-pravidla `@each`, `@for` a `@while`. Příklad `@each` cyklu, který je analogií běžného foreach cyklu, následuje.

SCSS kód

```
$icons: "up", "right", "bottom", "left";

@each $icon in $icons {
  .icon--#{$icon} {
    background: url($icon+".png");
    width: 2em;
    height: 2em;
  }
}
```

Vygenerovaný CSS kód

```
.icon--up {
  background: url("up.png");
  width: 2em;
  height: 2em;
}

.icon--right {
  background: url("right.png");
  width: 2em;
  height: 2em;
}

.icon--bottom {
```

`null`, není předána do výsledného CSS.

```
  background: url("bottom.png");
  width: 2em;
  height: 2em;
}

.icon--left {
  background: url("left.png");
  width: 2em;
  height: 2em;
}
```

Použití cyklu `@for` ukážeme na slíbeném příkladu mixiny s volitelným počtem parametrů.[15)]

SCSS kód

```
@mixin sizes($width, $selectors...) {
  @for $i from 0 to length($selectors) {
    #{nth($selectors, $i + 1)} {
      width: $width * ($i+1);
    }
  }
}

@include sizes(20ch, ".small", ".medium", ".large");
```

Vygenerovaný CSS kód

```
.small {
  width: 20ch;
}

.medium {
  width: 40ch;
}

.large {
  width: 60ch;
}
```

Vestavěné moduly

SASS obsahuje několik vestavěných modulů, které implementují užitečnou funkcionalitu. Nalezneme zde například modul pro práci s barvou, řetězci, selektory nebo modul implementující základní matematické operace. Moduly se načítají pomocí at-pravidla `@use`.[16)]

[15)] Funkce `nth()` je součástí modulu `sass:list`. Moduly představíme později.

[16)] V ukázce je použito at-pravidlo `@debug`, jež vypíše výsledek SCSS výrazu do konzole.

```
SCSS kód

@use 'sass:math';

@debug math.ceil(4.2);
```

Podrobný popis všech vestavěných modulů je k dispozici v oficiální dokumentaci preprocesoru SASS.[17)]

Organizace kódu

SCSS kód je možné rozdělit do více souborů a ty následně vložit pomocí pravidel `@import`, `@forward` nebo `@use`. `@import` přidává SCSS kód k existujícímu kódu.[18)] `@forward` umožňuje načíst SCSS kód jako modulu pomocí `@use` a případně připojit namespace v podobě prefixu. Detailní rozbor použití `@forward` a `@use` již přesahuje základní znalosti SCSS.

4. Další příklady

Předchozí části textu jsme prezentovali potřebnou teorii pro používání SASS preprocesoru společně s jednoduchými ukázkami použití. Nyní si ukážeme několik dalších velmi praktických úkolů společně s jejich řešením.

4.1. Něco jednoduchého na rozehřátí

Velmi často potřebujeme vytvářenou webovou stránku parametrizovat. Příkladem, je vytvoření tříd pro vizualizaci různých bloků. Předpokládejme, že blok má následující HTML strukturu.

```
<div class="box--info">
  <h2>Lorem ipsum</h2>
  <p>Duis ante orci, molestie vitae vehicula venenatis, tincidunt ac pede.
      Curabitur sagittis hendrerit ante.</p>
</div>
```

Nyní budeme chtít vytvořit sadu tříd `.box-info`, `.box-error` a další, které budou vizualizovat tyto bloky. Bloky se budou lišit pouze barvou textu, který bude kvůli kontrastu většinou bílý a barvou pozadí. Všechny bloky budou mít vnitřní okraj nastavený na hodnotu `1em`. Následuje kód pro realizaci uvedených tříd.

[17)] sass-lang.com/documentation

[18)] Jedná se o analogii CSS at-pravidla `@import`, ale na rozdíl od něj je přidání kódu prováděno při transpilaci.

SCSS kód

```
$padding: 1em;

@mixin box($color: white, $bg-c: bg-c) {
  padding: 1em;
  color: $color;
  background-color: $bg-c;

  h2 {margin: 0;}
  p {margin-bottom: 0;}
}


.box--info {
  @include box($bg-c: #0284c7)
}

.box--error {
  @include box($bg-c: #b91c1c)
}

.box--neutral {
  @include box($color: #0284c7, $bg-c: #ffffff)
}
```

Ve výše uvedeném kódu je použita mixina (řádky 3–10) s parametry, která umožňuje opakované použití. V mixině je využit nesting (řádky 8 a 9) pro reset výchozí vizualizace elementů `h2` a `p`. Dále kód obsahuje globální proměnou (řádek 1) pro vnitřní okraj bloků.

Výše uvedený úkol je možné mnoha způsoby rozšířit a ukázat výhodu tohoto řešení. Například přidání rámečku k boxu případně ikonografiky je velmi jednoduché stačí modifikovat mixinu. Také se nabízí vyčlenit použité barvy na řádcích 13–23 do proměnných.

4.2. Reset a globální nastavení

Je běžné, že se zdrojové kódy opakovaně používají. Typicky při tvoření stránky stojíme před rozhodnutím, zda provedeme tzv. reset, tedy že nastavme výchozí hodnoty všem elementům. Například chceme všem elementů nastavit hodnoty `margin` a `padding` na `0`.

SCSS kód

```
$reset: true;

@if $reset {
  * {
    margin: 0;
```

```
    padding: 0;
  }
}
```

Analogicky můžeme předchozí kód změnit tak, aby zobrazoval všem elementům rámeček. To se může hodit při ladění webové stránky.

SCSS kód

```
$show-border: true;
$border: 1px solid gray;

@if $show-border {
  * {
    border: $border;
    box-sizing: border-box;
  }
}
```

4.3. Generování atomických tříd

Při tvorbě webové stránky velmi často potřebujeme vytvořit několik atomických tříd,[19] tedy tříd, které obsahují pouze jednu deklaraci ve svém deklaračním bloku. Typicky, když vytváříme stránku s několika barvami, potřebujeme atomické třídy pro barvu textu, pozadí, případně rámečku. Psát ručně takovéto třídy je zdlouhavé nemluvě o případné změně kódu.

Tento úkol je možné řešit několika způsoby. Nejlepší řešení je využít modul `map`, který umožňuje zadat data ve tvaru klíč:hodnota. Ukázka řešení následuje.

SCSS kód

```
@use "sass:map";

// barvy
$colors: ("red-700": #b91c1c,
          "red-800": #991b1b,
          "red-900": #7f1d1d);

/* generování CSS pravidel */
@each $name, $color in $colors {
  .bg-#{$name} {
    background-color: $color;
  }

  .#{$name} {
```

[19] Můžeme se setkat i s označením utility třída.

```
    color: $color;
  }
}
```

5. Závěr

Stručně jsme představili preprocesor SASS, který významně rozšiřuje syntaxi CSS. Samotný preprocesor je velmi jednoduchý a není složité jej pochopit. Jeho začlenění do výuku umožní studentům efektivnější psaní CSS kódu. Pro úplnost ještě zmíníme zajímavou alternativu CSS preprocesorů a to stylované komponenty.[20)] Zjednodušeně řečeno stylované komponenty umožňují generovat CSS kód pomocí JavaScriptu. JavaScript nahrazuje omezený kód preprocesoru, čímž získáme výrazně větší možnosti generování CSS. Dodejme, že se ale nejedná o technologii, která je stejně rozšířená jako CSS preprocesory.

Silniční síť (Úlohy z MO kategorie P, 49. část)

PAVEL TÖPFER

Matematicko-fyzikální fakulta UK, Praha

V 35. ročníku Matematické olympiády byla do soutěže zařazena nová kategorie P (programování) se soutěžními úlohami zaměřenými na algoritmizaci a programování. Stalo se tak ve školním roce 1985/86. V té době se u nás začaly objevovat první mikropočítače, ale nebyly ještě rozšířeny natolik, aby je měl k dispozici každý student a na každé škole. V počátečních ročnících soutěže nebyly proto v kategorii P zadávány žádné praktické úlohy. Soutěžící odevzdávali řešení všech soutěžních úloh pouze v písemné formě, svoje programy psali na papír bez odladění na počítači.

V porovnání s úlohami zadávanými v MO-P v současné době, byly tehdejší soutěžní úlohy o něco snazší. To odpovídalo situaci, kdy se výuka informatiky a programování na střední školy teprve postupně zaváděla a její úroveň byla oproti dnešku podstatně nižší. Přesto ale i mezi úlo-

[20)] styled-components.com