Erratum: Emulátor zvukových syntezátorů

Toto erratum se týká oprav k bakalářské práci na téma **Emulátor zvukových syntezátorů**. Obsahuje výčet všech oprav překlepů a chyb v rozložení dokumentu. K žádným jiným změnám nedošlo. Ke každé změně uvádím výskyt v původním dokumentu. Číslování stran původního a opraveného dokumentu se začíná rozcházet od strany 21, kde došlo k odstranění celé zduplikované sekce zabírající asi čtvrtinu stránky. Chyb bylo opravdu hodně a důvodem bylo odevzdávání elektronické verze na poslední chvíli bez podrobnější korektury.

`Abstrakt:` Jazyk Cynth je omezený tak, aby za běhu nedocházelo k žádné **dynamicé** alokaci, ale zároveň umožňuje komplexní programování za překladu a práci se staticky alokovanými datovými **strukrurami**.

`Změna:` Jazyk Cynth je omezený tak, aby za běhu nedocházelo k žádné **dynamické** alokaci, ale zároveň umožňuje komplexní programování za překladu a práci se staticky alokovanými datovými **strukturami**.

`Str. 3:` Původním cílem bylo vytvořit **nástroj nástroj** na tvoření emulátorů zvukových syntezátorů

`Změna:` Původním cílem bylo vytvořit **nástroj** na tvoření emulátorů zvukových syntezátorů

`Str. 4:` který má **alospoň** základní znalost jazyka C, nebo podobných či odvozených jazyků. Nejde o úplnou **specikikaci** jazyka.

`Změna:` který má **alespoň** základní znalost jazyka C, nebo podobných či odvozených jazyků. Nejde o úplnou **specifikaci** jazyka.

`Str. 5:` Takto definovaný generátor bere celočíselný (Int) **prametr** t a vrací ho jako reálné číslo (Float).

`Změna:` Takto definovaný generátor bere celočíselný (Int) **parametr** t a vrací ho jako reálné číslo (Float).

`Str. 6:` *Upraven formát číslic ve výstupu programu.*

`Str. 6:` a **implmentačně** to vypadá tak, že si buffer pamatuje pozici

`Změna:` a **implementačně** to vypadá tak, že si buffer pamatuje pozici

---

`Str. 6:` bez uvedení hodnoty nebo definovat s **expliciní** inicializační hodnotou. Proměnné deklarované (bez **expliciní** inicializační hodnoty) jsou implicitně inticializovány na nulovou hodnotu.

`Změna:` bez uvedení hodnoty nebo definovat s **explicitní** inicializační hodnotou. Proměnné deklarované (bez **explicitní** inicializační hodnoty) jsou implicitně inicializovány na nulovou hodnotu.

`Str. 7:` S jednoduchými typy lze **prvádět** podobné operace, jako v C. Tabulka 1.1 uvádí přehled různých operátorů seřazených od nejvyšší **precednece** k nejnižší. Tedy např. operace násobení váže operandy silněji než sčítání, ale mezi násobením a dělením není rozdíl v **precedneci**.

`Změna:` S jednoduchými typy lze **provádět** podobné operace, jako v C. Tabulka 1.1 uvádí přehled různých operátorů seřazených od nejvyšší **precedence** k nejnižší. Tedy např. operace násobení váže operandy silněji než sčítání, ale mezi násobením a dělením není rozdíl v **precedenci**.

`Str. 7:` *Výrazy nerovností v tabulce 1.1 rozděleny do dvou řádků kvůli čitelnosti. Sjednocena velikost písma v tabulce.*

`Str. 7:` Konverze **meze** jednoduchými typy jsou také podobné těm v C

`Změna:` Konverze **mezi** jednoduchými typy jsou také podobné těm v C

`Str. 8:` Deklarace pole (bez **expliciní** inicializační hodnoty) vytvoří pole zaplněné nulami daného typu.

`Změna:` Deklarace pole (bez **explicitní** inicializační hodnoty) vytvoří pole zaplněné nulami daného typu.

`Str. 8:` Indexuje se ale **po zpátku**, tedy od nuly do záporných čísel. Nula **označuje** nejnovější vzorek

`Změna:` Indexuje se ale **pozpátku**, tedy od nuly do záporných čísel. Nula **označuje** nejnovější vzorek

---

`Str. 9:` Funkce lze také vytvořit **anonynmně**, jak je popsáno v předchozí kapitole.

`Změna:` Funkce lze také vytvořit **anonymně**, jak je popsáno v předchozí kapitole.

`Str. 9:` Takové zachycení probíhá kopií, až na speciální staticky **alokovoané** typy (buffery a IO typy).

`Změna:` Takové zachycení probíhá kopií, až na speciální staticky **alokované** typy (buffery a IO typy).

`Str. 9:` *Změna typu proměnné add1 v příkladu kódu.*

`Str. 9:` IO typy v tomto úvodu **vynchám**, jelikož nejsou prakticky využitelné v aktuální verzi. Jsou ale **do podrobna** popsány v další kapitole.

`Změna:` IO typy v tomto úvodu **vynechám**, jelikož nejsou prakticky využitelné v aktuální verzi. Jsou ale **dopodrobna** popsány v další kapitole.

`Str. 9-10:` Uzávorkování výrazů je tedy jen **expliciní** uzávorkování 1-tice.

`Změna:` Uzávorkování výrazů je tedy jen **explicitní** uzávorkování 1-tice.

`Str. 10:` ale v některých případech syntaxe vynucuje explicitně **uvávorkovanou** n-tici.

`Změna:` ale v některých případech syntaxe vynucuje explicitně **uzávorkovanou** n-tici.

`Str. 11:` Příkazy v blocku jsou **narozdíl** od C jen odděleny středníky, ne ukončeny.

`Změna:` Příkazy v blocku jsou **na rozdíl** od C jen odděleny středníky, ne ukončeny.

`Str. 13:` je třeba provést konvoluci s **implulzní** odezvou nějakého filtru.

`Změna:` je třeba provést konvoluci s **impulzní** odezvou nějakého filtru.

`Str. 14:` V reálné situaci by molo jít třeba i o konvoluci signálu **si** pevně danou **implulzní** odezvou filtru. Takovou **implulzní** odezvu by mohlo být vhodnější mít předpočítanou v poli.

`Změna:` V reálné situaci by molo jít třeba i o konvoluci signálu **s** pevně danou **impulzní** odezvou filtru. Takovou **impulzní** odezvu by mohlo být vhodnější mít předpočítanou v poli.

`Str. 14:` V souboru config.mk lze nastavit **paramerty** pro překladač výsledného C kódu.

`Změna:` V souboru config.mk lze nastavit **parametry** pro překladač výsledného C kódu.

`Str. 15:` k reprezentaci rekurzivního volání funkce**.)** Dále jde o tzv. symboly **definovné** jako řetězce

`Změna:` k reprezentaci rekurzivního volání funkce**).** Dále jde o tzv. symboly **definované** jako řetězce

---

`Str. 15:` Zbylé tokeny jsou specifické řetězce **definovné** regulárními výrazy.

`Změna:` Zbylé tokeny jsou specifické řetězce **definované** regulárními výrazy.

`Str. 16:` **Merezy**, konce řádků a komentáře (blank, endl, comment, multicom) parser ignoruje. Ostatní tokeny z výčtu výše nesou sémantickou hodnotu danou odpovídajícími **řetězcemi**

`Změna:` **Mezery**, konce řádků a komentáře (blank, endl, comment, multicom) parser ignoruje. Ostatní tokeny z výčtu výše nesou sémantickou hodnotu danou odpovídajícími **řetězci**

`Str. 16:` Číselné literály (int i float) **umožnují** zápis ve vědecké notaci.

`Změna:` Číselné literály (int i float) **umožňují** zápis ve vědecké notaci.

`Str. 19:` příliš matoucí pro uživatele zvyklé na **exvivlentní** syntaxi v C.

`Změna:` matoucí pro uživatele zvyklé na **ekvivalentní** syntaxi v C.

`Str. 19:` Vyhodnocený výraz má vždy tzv. výslednou hodnotu**,** Pojmy výraz a hodnota jsou záměnné.

`Změna:` Vyhodnocený výraz má vždy tzv. výslednou hodnotu**.** Pojmy výraz a hodnota jsou záměnné.

`Str. 20:` je tím myšleno, že **se** vrací přesně stejnou hodnotu, stejného typu.

`Změna:` je tím myšleno, že vrací přesně stejnou hodnotu, stejného typu.

`Str. 20:` Float hodnoty lze vytvořit **destinným** literálem (token float).

`Změna:` Float hodnoty lze vytvořit **desetinným** literálem (token float).

`Str. 20:` To znamená, že při předávání se jejich hodnota kopíruje, a ta původní zůstává **nedotčená**

`Změna:` To znamená, že při předávání se jejich hodnota kopíruje, a ta původní zůstává **nedotčena**

`Str. 20:` Samotná reference se předává hodnotou a nelze vytvořit další referenci odkazující na **ní**.

`Změna:` Samotná reference se předává hodnotou a nelze vytvořit další referenci odkazující na **ni**.

`Str. 21:` IO typy **obahující** n-tice jsou ekvivalentní n-ticím IO typů **obahujících** odpovídající 1-tice. Buffery reprezentují referenci na **peveně** daný počet hodnot

`Změna:` IO typy **obsahující** n-tice jsou ekvivalentní n-ticím IO typů **obsahujících** odpovídající 1-tice. Buffery reprezentují referenci na **pevně** daný počet hodnot

---

`Str. 21:` tedy obsahují spustitelnou **paramertizovatelnou** část programu

`Změna:` tedy obsahují spustitelnou **parametrizovatelnou** část programu

---

`Str. 21:` zda jde o imutabilní (neboli konstantní) hodnotu přidáním keywordu **cost** za daný typ.
`Změna:` zda jde o imutabilní (neboli konstantní) hodnotu přidáním keywordu **const** za daný typ.

`Str. 21:` pojmy identita a shoda splývají, jelikož mají pevně **uerčenou** mutabilitu.
`Změna:` pojmy identita a shoda splývají, jelikož mají pevně **určenou** mutabilitu.

`Str. 21:` příkaz složený z deklarací **reprezentujích** definované proměnné a výrazů **reprezentujích**
`Změna:` složený z deklarací **reprezentujících** definované proměnné a výrazů **reprezentujících**

`Str. 21:` *Odstranění duplikované sekce.*

`Str. 22:` V aktuální verzi není **implimentován** žádný feedback na vrácenou hodnotu.
`Změna:` V aktuální verzi není **implementován** žádný feedback na vrácenou hodnotu.

`Str. 23:` V případě příkazu **delkarace**, je tato proměnná tzv. viditelná
`Změna:` V případě příkazu **deklarace**, je tato proměnná tzv. viditelná

`Str. 23:` Block vytváří **tzv. tzv.** scope, což je část programu,
`Změna:` Block vytváří **tzv.** scope, což je část programu,

`Str. 23:` Nulová hodnota typu Bool je false a **nulové hodnoty** typu Int, resp. Float,
`Změna:` Nulová hodnota typu Bool je false a **nulová hodnota** typu Int, resp. Float,

---

`Str. 23:` **Synakticky** jsou cíle tvořeny libovolným výrazem z kategorie expr_post.
`Změna:` **Syntakticky** jsou cíle tvořeny libovolným výrazem z kategorie expr_post.

`Str. 23:` Výrazy **reprezentují** l-hodnoty v Cynthu nejsou.
`Změna:` Výrazy **reprezentující** l-hodnoty v Cynthu nejsou.

`Str. 24:` se tak v uvedeném pořadí přiřadí do **jednotlivích** 1-tic z cílů.
`Změna:` se tak v uvedeném pořadí přiřadí do **jednotlivých** 1-tic z cílů.

`Str. 24:` Argumenty jsou dány explicitně **uzávorkovánou** n-ticí. Výstupní typy, **výstupní** parametry
`Změna:` Argumenty jsou dány explicitně **uzávorkovanou** n-ticí. Výstupní typy, **vstupní** parametry

---

`Str. 24:` **Narozdíl** od C není block ve funkci sémanticky odlišný od jiných blocků.
`Změna:` **Na rozdíl** od C není block ve funkci sémanticky odlišný od jiných blocků.

`Str. 24:` Následně se vytvoří další scope pro **těla** funkce.

`Změna:` Následně se vytvoří další scope pro **tělo** funkce.

`Str. 25:` Výsledek **fukce** se označuje jako vrácená hodnota.

`Změna:` Výsledek **funkce** se označuje jako vrácená hodnota.

`Str. 25:` životnost **popsisuje** vlastnost hodnot, které ani nemusí být přiřazeny do proměnných.

`Změna:` životnost **popisuje** vlastnost hodnot, které ani nemusí být přiřazeny do proměnných.

`Str. 26:` Prvek **dadný** výrazem musí být 1-tice (alespoň v aktuální verzi) jednoduchého typu

`Změna:` Prvek **daný** výrazem musí být 1-tice (alespoň v aktuální verzi) jednoduchého typu

---

`Str. 26:` začínající počáteční hodnotou a **nepřesahujeící** horní nebo dolní mez.

`Změna:` začínající počáteční hodnotou a **nepřesahující** horní nebo dolní mez.

`Str. 26:` Výše popsané prvky polí jsou určeny i pro použití **jako** v lokacích, ale v aktuální verzi

`Změna:` Výše popsané prvky polí jsou určeny i pro použití v lokacích, ale v aktuální verzi

---

`Str. 26:` v případě pole nebo bufferu na pozici dané **toutou** hodnotou.

`Změna:` v případě pole nebo bufferu na pozici dané **touto** hodnotou.

`Str. 27:` Subscript output jako výraz **typu** je chybou.

`Změna:` Subscript output **typu** jako výraz je chybou.

`Str. 27:` V Cynthu jsou implementovány **násnledující** řídící struktury:

`Změna:` V Cynthu jsou implementovány **následující** řídící struktury:

`Str. 27:` V případě smyček je **evivalentní** označení „větev“ i „tělo“.

`Změna:` V případě smyček je **ekvivalentní** označení „větev“ i „tělo“.

`Str. 27:` vyhodnotí (resp. vykoná, pokud je if příkazový) **se** pozitivní větev

`Změna:` vyhodnotí **se** (resp. vykoná, pokud je if příkazem) pozitivní větev

---

`Str. 27:` konstrukt pro podmínku bez negativní větve **zjendnodušuje** gramatiku

`Změna:` konstrukt pro podmínku bez negativní větve **zjednodušuje** gramatiku

---

`Str. 27:` Větev for loopu se vykoná pro každou n-tici hodnot na **sejných** pozicích v daných polích.

`Změna:` Větev for loopu se vykoná pro každou n-tici hodnot na **stejných** pozicích v daných polích.

`Str. 27:` a tedy se prvky **přiřazeují** hodnotou.

`Změna:` a tedy se prvky **přiřazují** hodnotou.

`Str. 28:` Podobají se operátorům v C **syntkticky** a částečně i sémanticky, ale u některých z nich jsou značné rozdíly. **Precednece** všech operátorů je stejná jako v C.

`Změna:` Podobají se operátorům v C **syntakticky** a částečně i sémanticky, ale u některých z nich jsou značné rozdíly. **Precedence** všech operátorů je stejná jako v C.

`Str. 28:` **Oprátory** jsou polymorfické v tom smyslu, že akceptují různé typy.

`Změna:` **Operátory** jsou polymorfické v tom smyslu, že akceptují různé typy.

`Str. 28:` *Odebrání keywordu return z implementace modula v tabulce 2.1.*

`Str. 28:` Celočíselné dělení zaokrouhluje dolů **narozdíl** od C, které zaokrouhluje směrem k nule.

`Změna:` Celočíselné dělení zaokrouhluje dolů **na rozdíl** od C, které zaokrouhluje směrem k nule.

`Str. 29:` *Rozdělení výrazů a implementací nerovností čárkou kvůli čitelnosti v tabulce 2.3.*

`Str. 29:` Mají při tom sémantiku ekvivalentní **odpovídajímu** castu v C, až na konverzi z Floatu do Intu, která **narozdíl** od C provádí zaokrouhlení dolů.

`Změna:` Mají při tom sémantiku ekvivalentní **odpovídajícímu** castu v C, až na konverzi z Floatu do Intu, která **na rozdíl** od C provádí zaokrouhlení dolů.

`Str. 29:` konverze na konstantní **obasžené** hodnoty je **vpořádku**, ale naopak by znamenalo mutabilitu původně konstantních hodnot. Stejně tak konverze na menší velikost je **vpořádku**

`Změna:` konverze na konstantní **obsažené** hodnoty je **v pořádku**, ale naopak by znamenalo mutabilitu původně konstantních hodnot. Stejně tak konverze na menší velikost je **v** pořádku

---

`Str. 30:` Buffer velikosti n reprezentuje historii **poslndních** n vzorků nějakého signálu.

`Změna:` Buffer velikosti n reprezentuje historii **posledních** n vzorků nějakého signálu.

`Str. 30:` a **případne** přijímající parametr typu Int označující čas daný **početem** uběhlých vzorků

`Změna:` a **případně** přijímající parametr typu Int označující čas daný **počtem** uběhlých vzorků

`Str. 30 (nadpis):` Kompilační kostanty

`Změna:` Kompilační konstanty

`Str. 30:` různé optimalizace díky kompilačním **kostantám**

`Změna:` různé optimalizace díky kompilačním **konstantám**

`Str. 30:` aby funkce mohla být vyhodnocena striktně za **překleadu**, musí mít i konstantní parametry.

`Změna:` aby funkce mohla být vyhodnocena striktně za **překladu**, musí mít i konstantní parametry.

`Str. 31:` Využívají se jen obecně přístupné vlastnosti a standardní **klihovny** standardu

`Změna:` Využívají se jen obecně přístupné vlastnosti a standardní **knihovny** standardu

---

`Str. 31:` soubory se závislostmi mezi implementacemi a headery **vygenererované** pomocí GCC.

`Změna:` soubory se závislostmi mezi implementacemi a headery **vygenerované** pomocí GCC.

`Str. 31:` je určený ke statickému slinkování s **předkomilovaným**

`Změna:` je určený ke statickému slinkování s **předkompilovaným**

`Str. 32 (diagram):` Výsldný program

`Změna:` Výsledný program

`Str. 32:` Syntaktická **anlýza** dle vstupních tokenů vytvoří AST

`Změna:` Syntaktická **analýza** dle vstupních tokenů vytvoří AST

`Str. 33:` které se **aplikajují** přes std::visit

`Změna:` které se **aplikují** přes std::visit

`Str. 33:` přechod ze sémantických struktur do C **kódU** jako translation.

`Změna:` přechod ze sémantických struktur do C **kódu** jako translation.

`Str. 33:` Spojit extrakci jmen i překlad by ale **mohle** být nevýhodné.

`Změna:` Spojit extrakci jmen i překlad by ale **mohlo** být nevýhodné.

`Str. 34:` Po dokončení **deklakace** či evaluace se odstraní.

`Změna:` Po dokončení **deklarace** či evaluace se odstraní.

`Str. 34:` Pro nový scope se vytvoří **dceřinný** main context i s novým, odpovídajícím lookup contextem.

`Změna:` Pro nový scope se vytvoří **dceřiný** main context i s novým, odpovídajícím lookup contextem.

`Str. 37:` je hlavním **výslekdem** práce právě zjištění

`Změna:` je hlavním **výsledkem** práce právě zjištění