Compiladores/Projecto de Compiladores/Projecto 2015-2016/Manual de Referência da Linguagem "zu"

A linguagem **zu** é uma linguagem imperativa e é apresentada de forma intuitiva neste manual. São apresentadas características básicas da linguagem (tipos de dados, manipulação de nomes); convenções lexicais; estrutura/sintaxe; especificação das funções; semântica das instruções; semântica das expressões; e, finalmente, alguns exemplos.

Tipos de Dados

A linguagem é fracamente tipificada (são efectuadas algumas conversões implícitas). Existem 4 tipos de dados, todos compatíveis com a linguagem C, e com alinhamento em memória sempre a 32 bits:

- Tipos numéricos: os **inteiros**, em complemento para dois, ocupam 4 bytes; os **reais**, em vírgula flutuante, ocupam 8 bytes (IEEE 754).
- As **cadeias de caracteres** são vectores de caracteres terminados por ASCII NULL (**0x00**, **\0**). Variáveis e literais deste tipo só podem ser utilizados em atribuições, impressões, ou como argumentos/retornos de funções.
- Os **ponteiros** representam endereços de objectos e ocupam 4 bytes. Podem ser objecto de operações aritméticas (deslocamentos) e permitem aceder ao valor apontado.

Os tipos suportados por cada operador e a operação a realizar são indicados na definição das expressões.

Manipulação de Nomes

Os nomes (identificadores) correspondem a constantes, variáveis e funções. Nos pontos que se seguem, usa-se o termo entidade para as designar indiscriminadamente, explicitando-se quando a descrição for válida apenas para um subconjunto.

Espaço de nomes e visibilidade dos identificadores

O espaço de nomes global é único, pelo que um nome utilizado para designar uma entidade num dado contexto não pode ser utilizado para designar outras (ainda que de natureza diferente).

Os identificadores são visíveis desde a declaração até ao fim do alcance: ficheiro (globais) ou função (locais). A reutilização de identificadores em contextos inferiores encobre declarações em contextos superiores: redeclarações locais podem encobrir as globais até ao fim de uma função. É possível utilizar símbolos globais nos contextos das funções, mas não é possível defini-los (ver símbolos globais).

Validade das variáveis

As entidades globais (declaradas fora de qualquer função), existem durante toda a execução do programa. As variáveis locais a uma função existem apenas durante a sua execução. Os argumentos formais são válidos enquanto a função está activa.

Convenções Lexicais

Para cada grupo de elementos lexicais (*tokens*), considera-se a maior sequência de caracteres constituindo um

elemento válido.

Caracteres brancos

São considerados separadores e não representam nenhum elemento lexical: **mudança de linha** ASCII LF (**0x0A**, **\n**), **recuo do carreto** ASCII CR (**0x0D**, **\r**), **espaço** ASCII SP (**0x20**, ␣) e **tabulação horizontal** ASCII HT (**0x09**, **\t**).

Comentários

Existem dois tipos de comentários, que também funcionam como elementos separadores:

- **explicativos** -- começam com **//** e acabam no fim da linha; e
- **operacionais** -- começam com **/*** e terminam com ***/**, podendo estar aninhados.

Se as sequências de início fizerem parte de uma cadeia de caracteres, não iniciam um comentário (ver definição das cadeias de caracteres).

Palavras chave

A linguagem não tem palavras chave. Todas as palavras correspondem a identificadores.

O identificador **zu**, embora não reservado, corresponde à função principal.

Tipos

Os seguintes elementos lexicais designam tipos em declarações (ver gramática): **#** (inteiro), **%** (real), **$** (cadeia de caracteres).

Os tipos correspondentes a ponteiros são delimitados por **<** e por **>** (ver gramática).

Operadores de expressões

São considerados operadores os elementos lexicais apresentados na definição das expressões.

Delimitadores e terminadores

Os seguintes elementos lexicais são delimitadores/terminadores: **,** (vírgula), **;** (ponto e vírgula), **!** e **!!** (operações de impressão), e **(** e **)** (delimitadores de expressões).

Identificadores (nomes)

São iniciados por uma letra ou por **_** (sublinhado), seguindo-se 0 (zero) ou mais letras, dígitos ou **_** (sublinhado). O comprimento do nome é ilimitado e dois nomes são distintos se houver alteração de maiúscula para minúscula, ou vice-versa, de pelo menos um carácter.

Literais

São notações para valores constantes de alguns tipos da linguagem (não confundir com constantes, i.e., identificadores que designam elementos cujo valor não pode ser alterado durante a execução do programa).

Inteiros

Um literal inteiro é um número não negativo. Uma constante inteira pode, contudo, ser negativa: números negativos são construídos pela aplicação do operador menos unário (**-**) a um literal positivo.

Literais inteiros decimais são constituídos por sequências de 1 (um) ou mais dígitos de **0** a **9**, em que o primeiro dígito não é **0** (zero), excepto no caso do número 0 (zero). Neste caso, é composto apenas pelo dígito **0** (zero) (em qualquer base).

Literais inteiros hexadecimais começam sempre com a sequência **0x**, seguida de um ou mais digitos de **0** a **9** ou de **a** a **f** (sem distinguir maiúsculas de minúsculas). As letras de **a** a **f** representam os valores de 10 a 15 respectivamente. Exemplo: **0x07**.

Se não for possível representar o literal inteiro na máquina, devido a um overflow, deverá ser gerado um erro lexical.

Reais em vírgula flutuante

Os literais reais são expressos em notação científica (tal como em C). Exemplo: **12.34e-24** = 12.34 x 10^{-24}.

Um literal sem **.** (ponto decimal) nem parte exponencial é do tipo inteiro.

Cadeias de caracteres

As cadeias de caracteres são delimitadas por aspas (**"**) e podem conter quaisquer caracteres, excepto ASCII NULL (**0x00 \0**). Nas cadeias, os delimitadores de comentários não têm significado especial. Se for escrito um literal que contenha **\0**, então a cadeia termina nessa posição. Exemplo: **"ab\0xy"** tem o mesmo significado que **"ab"**.

É possível designar caracteres por sequências especiais (iniciadas por \), especialmente úteis quando não existe representação gráfica directa. As sequências especiais correspondem aos caracteres ASCII LF, CR e HT (**\n**, **\r** e **\t**, respectivamente), aspa (**\"**), barra (****), ou a quaisquer outros especificados através de 1 ou 2 digitos hexadecimais (e.g. **\0a** ou apenas **\a** se o carácter seguinte não representar um dígito hexadecimal).

Elementos lexicais distintos que representem duas ou mais cadeias consecutivas são representadas na linguagem como uma única cadeia que resulta da concatenação. Exemplo: **"ab"** ␣ **"cd"** é o mesmo que **"abcd"**.

Ponteiros

O único literal admissível para ponteiros é **0**, indicando o ponteiro nulo.

Gramática

A gramática da linguagem está resumida abaixo. Considerou-se que os elementos em tipo fixo são literais, que os parênteses curvos agrupam elementos, que elementos alternativos são separados por uma barra vertical, que elementos opcionais estão entre parênteses rectos, que os elementos que se repetem zero ou mais vezes estão entre ⟨ e ⟩. Alguns elementos usados na gramática também são elementos da linguagem descrita se representados em tipo fixo (e.g., parênteses).

ficheiro	→	⟨ *declaração* ⟩
declaração	→	*variável* **;** | *função*
variável	→	*tipo identificador* [**!** | **?**] [**=** *expressão*]
função	→	(*tipo* | **!**) *identificador* [**!** | **?**] **(** [*variáveis*] **)** [**=** *literal*] [*corpo*]

variáveis	→	*variável* ⟨ **,** *variável* ⟩
tipo	→	**#** | **%** | **$** | **<** *tipo* **>**
corpo	→	*bloco*
bloco	→	**{** ⟨ *declaração* ⟩ ⟨ *instrução* ⟩ **}**
instrução	→	*expressão* **;** | *expressão* **!** | *expressão* **!!**
	→	**><** | **<>** | **!!!**
	→	*instrução-condicional* | *instrução-de-iteração* | *bloco*
instrução-condicional	→	**[** *expressão* **]** **#** *instrução*
	→	**[** *expressão* **]** **?** *instrução* [**:** *instrução*]
instrução-de-iteração	→	**[** [*variáveis*] **;** [*expressões*] **;** [*expressões*] **]** *instrução*
	→	**[** [*expressões*] **;** [*expressões*] **;** [*expressões*] **]** *instrução*
expressões	→	*expressão* ⟨ **,** *expressão* ⟩

Tipos, identificadores, literais e definição de expressões

Algumas definições foram omitidas da gramática: tipos de dados, *identificador* (ver identificadores), *literal* (ver literais); *expressão* (ver expressões).

Left-values

Os *left-values* são posições de memória que podem ser modificadas (excepto onde proibido pelo tipo de dados). Os elementos de uma expressão que podem ser utilizados como *left-values* encontram-se individualmente identificados na semântica das expressões.

Ficheiros

Um ficheiro é designado por principal se contiver a função principal (a que inicia o programa).

Declaração de variáveis

Uma declaração de variável indica sempre um tipo de dados e um identificador.

Exemplos:

- Inteiro: **#i**
- Real: **%r**
- Cadeia de caracteres: **$s**
- Ponteiro para inteiro: **<#>p1** (equivalente a **int*** em C)
- Ponteiro para real: **<%>p2** (equivalente a **double*** em C)
- Ponteiro para cadeia de caracteres: **<$>p3** (equivalente a **char**** em C)
- Ponteiro para ponteiro para inteiro: **<<#>>p4** (equivalente a **int**** em C)

Símbolos globais

Por omissão, os símbolos são privados a um módulo, não podendo ser importados por outros módulos.

O marcador **!** permite declarar um identificador como público, tornando-o acessível a partir de outros módulos.

O marcador **?** (opcional para funções) permite declarar num módulo entidades definidas em outros módulos. As entidades não podem ser inicializadas numa declaração importada.

Exemplos:

- Declarar variável privada ao módulo: **%pi = 22/7 /*** [2] ***/**
- Declarar variável pública: **%pi! = 22/7 /*** [3] ***/**
- Usar definição externa: **%pi?**

Inicialização

Quando existe, é uma expressão que segue o sinal **=** ("igual"): inteira, real, ponteiro. Entidades reais podem ser inicializadas por expressões inteiras (conversão implícita). A expressão de inicialização deve ser um literal se a variável for global.

As cadeias de caracteres são (possivelmente) inicializadas com uma lista não nula de valores sem separadores. Estes valores são sempre constantes, independentemente de o identificador que as designa ser constante ou não.

Exemplos:

- Inteiro (literal): **#i = 3**
- Inteiro (expressão): **# i = j+1**
- Real (literal): **%r = 3.2**
- Real (expressão): **%r = i - 2.5 + f(3)**
- Cadeia de caracteres (literal): **$s = "olá"**
- Cadeia de caracteres (literais): **$s = "olá" "mãe"**
- Ponteiro (literal): **<<<%>>>p = 0**
- Ponteiro (expressão): **<#>p = q + 1**

Funções

Uma função permite agrupar um conjunto de instruções num corpo, executado com base num conjunto de parâmetros (os argumentos formais), quando é invocada a partir de uma expressão.

Declaração

As funções são sempre designadas por identificadores constantes precedidos do tipo de dados devolvido pela função. Se a função não devolver um valor, usar-se o marcador de tipo especial **!** para o indicar.

As funções que recebam argumentos devem indicá-los no cabeçalho. Funções sem argumentos definem um cabeçalho vazio. Não é possível aplicar os qualificadores de exportação/importação **!** ou **?** (ver símbolos globais) às declarações dos argumentos de uma função.

A declaração de uma função sem corpo é utilizada para tipificar um identificador exterior ou para efectuar declarações antecipadas (utilizadas para pré-declarar funções que sejam usadas antes de ser definidas, por exemplo, entre duas funções mutuamente recursivas). Caso a declaração tenha corpo, define-se uma nova

função (neste caso, não pode utilizar-se o qualificador **?**).

Invocação

A função só pode ser invocada através de um identificador que refira uma função previamente declarada ou definida.

Se existirem argumentos, na invocação da função, o identificador é seguido de uma lista de expressões delimitadas por parênteses curvos. Esta lista é uma sequência, possivelmente vazia, de expressões separadas por vírgulas. O número e tipo de parâmetros actuais deve ser igual ao número e tipo dos parâmetros formais da função invocada. A ordem dos parâmetros actuais deverá ser a mesma dos argumentos formais da função a ser invocada.

De acordo com a convenção Cdecl, a função chamadora coloca os argumentos na pilha e é responsável pela sua remoção, após o retorno da chamada. Assim, os parâmetros actuais devem ser colocados na pilha pela ordem inversa da sua declaração (i.e., são avaliados da direita para a esquerda antes da invocação da função e o resultado passado por cópia/valor). O endereço de retorno é colocado no topo da pilha pela chamada à função.

Corpo

O corpo de uma função consiste num bloco que contém declarações (opcionais) seguidas de instruções (opcionais). Não é possível aplicar os qualificadores de exportação (**!**) ou de importação (**?**) (ver símbolos globais) dentro do corpo de uma função.

O valor devolvido por uma função, através de atribuição ao *left-value* especial com o nome da função, deve ser do tipo declarado.

Se existir um valor declarado por omissão para o retorno da função (indicado pela notação **=** seguindo a assinatura da função), então deve ser utilizado se não for especificado outro. A especificação do valor de retorno por omissão é obrigatoriamente um literal do tipo indicado. É um erro especificar um valor de retorno se a função for declarada como não retornando um valor (tipo indicado como **!**). Uma função cujo retorno seja inteiro ou ponteiro retorna 0 (zero) por omissão (i.e., se não for especificado o valor de retorno). Em todos os outros casos, o valor de retorno é indeterminado se não for definido explicitamente.

Uma instrução **!!!** causa a interrupção da função e o retorno do seu valor actual ao chamador.

Qualquer bloco (usado, por exemplo, numa instrução condicional ou de iteração) pode definir variáveis.

Função principal e execução de programas

Um programa inicia-se com a invocação da função **zu** (sem argumentos). Os argumentos com que o programa foi chamado podem ser obtidos através de funções **#argc()** (devolve o número de argumentos); **$argv(#n)** (devolve o n-ésimo argumento como uma cadeia de caracteres) (**n>0**); e **$envp(#n)** (devolve a n-ésima variável de ambiente como uma cadeia de caracteres) (**n>0**).

O valor de retorno da função principal é devolvido ao ambiente que invocou o programa. Este valor de retorno segue as seguintes regras (sistema operativo): 0 (zero), execução sem erros; 1 (um), argumentos inválidos (em número ou valor); 2 (dois), erro de execução. Os valores superiores a 128 indicam que o programa terminou com um sinal. Em geral, para correcto funcionamento, os programas devem devolver 0 (zero) se a execução foi bem sucedida e um valor diferente de 0 (zero) em caso de erro.

A biblioteca de run-time (RTS) contém informação sobre outras funções de suporte disponíveis, incluindo chamadas ao sistema (ver também o manual da RTS).

Instruções

Excepto quando indicado, as instruções são executadas em sequência.

Blocos

Cada bloco tem uma zona de declarações de constantes e variáveis locais (facultativa), seguida por uma zona com instruções.

A visibilidade das variáveis é limitada ao bloco em que foram declaradas. As entidades declaradas podem ser directamente utilizadas em sub-blocos ou passadas como argumentos para funções chamadas dentro do bloco. Caso os identificadores usados para definir as variáveis locais já estejam a ser utilizados para definir outras entidades ao alcance do bloco, o novo identificador passa a referir uma nova entidade definida no bloco até ao que ele termine (a entidade previamente definida continua a existir, mas não pode ser directamente referida pelo seu nome). Esta regra é também válida relativamente a argumentos de funções (ver corpo das funções).

Instrução condicional

Esta instrução tem comportamento idêntico ao da instrução **if-else** em C.

Instrução de iteração

Esta instrução tem comportamento idêntico ao da instrução **for** em C.

Instrução de terminação

A instrução **><** termina o ciclo mais interior em que a instrução se encontrar, tal como a instrução **break** em C. Esta instrução só pode existir dentro de um ciclo, sendo a última instrução do seu bloco.

Instrução de continuação

A instrução **<>** reinicia o ciclo mais interior em que a instrução se encontrar, tal como a instrução **continue** em C. Esta instrução só pode existir dentro de um ciclo, sendo a última instrução do seu bloco.

Instrução de retorno

A instrução **!!!**, se existir, é a última instrução do seu bloco. Ver comportamento na descrição do corpo de uma função.

Expressões como instruções e operações de impressão

As expressões utilizadas como instruções são avaliadas, mesmo que não produzam efeitos secundários, e, eventualmente, o seu valor impresso (quando seguidas de **!** ou **!!** -- impressão sem/com mudança de linha). Valores numéricos (inteiros ou reais) são impressos em decimal. As cadeias de caracteres são impressas na codificação nativa. Ponteiros não podem ser impressos.

Expressões

Uma expressão é uma representação algébrica de uma quantidade: todas as expressões têm um tipo e devolvem um valor.

Existem expressões primitivas e expressões que resultam da avaliação de operadores.

A tabela seguinte apresenta as precedências relativas dos operadores: é a mesma para operadores na mesma linha, sendo as linhas seguintes de menor prioridade que as anteriores. A maioria dos operadores segue a

semântica da linguagem C (excepto onde explicitamente indicado). Tal como em C, os valores lógicos são 0 (zero) (valor falso), e diferente de zero (valor verdadeiro).

Tipo de Expressão	Operadores	Associatividade	Operandos	Semântica
primária	**( ) [ ]**	não associativos	-	parênteses curvos, indexação, reserva de memória
unária	**+ - ?**	não associativos	-	identidade e simétrico, indicação de posição
multiplicativa	*** / %**	**da esquerda para a direita**	**inteiros, reais**	**C (% é apenas para inteiros)**
aditiva	**+ -**	**da esquerda para a direita**	**inteiros, reais, ponteiros**	**C: se envolverem ponteiros, calculam: (i) deslocamentos, i.e., um dos operandos deve ser do tipo ponteiro e o outro do tipo inteiro; (ii) diferenças de ponteiros, i.e., apenas quando se aplica o operador - a dois ponteiros do mesmo tipo (o resultado é o número de objectos do tipo apontado entre eles). Se a memória não for contígua, o resultado é indefinido.**
comparativa	**< > <= >=**	**da esquerda para a direita**	**inteiros, reais**	**C**
igualdade	**== !=**	**da esquerda para a direita**	**inteiros, reais, ponteiros**	**C**
"não" lógico	**~**	**não associativo**	**inteiros**	**C**
"e" lógico	**&**	**da esquerda para a direita**	**inteiros**	**C: o 2º argumento só é avaliado se o 1º não for falso.**
"ou" lógico	**|**	**da esquerda para a direita**	**inteiros**	**C: o 2º argumento só é avaliado se o 1º não for verdadeiro.**
atribuição	**=**	**da direita para a esquerda**	**todos os tipos**	**O valor da expressão do lado direito do operador é guardado na posição indicada pelo *left-value* (operando esquerdo do operador). Podem ser atribuídos valores inteiros a *left-values* reais (conversão automática). Nos outros casos, ambos os tipos têm de concordar.**

Expressões primitivas

As expressões literais e a invocação de funções foram definidas acima.

Identificadores

Um identificador é uma expressão se tiver sido declarado. Um identificador pode denotar uma variável ou uma constante.

Um identificador é o caso mais simples de um *left-value*, ou seja, uma entidade que pode ser utilizada no lado esquerdo (*left*) de uma atribuição. O valor de retorno de uma função é definido por um *left-value* especial (ver também definições relativas ao corpo das funções).

Leitura

A operação de leitura de um valor inteiro ou real pode ser efectuado pela expressão **@**, que devolve o valor lido, de acordo com o tipo esperado (inteiro ou real). Caso se use como argumento dos operadores de impressão (**!** ou **!!**, deve ser lido um inteiro.

Exemplos: **a = @** (leitura para **a**), **f(@)** (leitura para argumento de função), **@!!** (leitura e impressão).

Parênteses curvos

Uma expressão entre parênteses curvos tem o valor da expressão sem os parênteses e permite alterar a prioridade dos operadores. Uma expressão entre parênteses não pode ser utilizada como *left-value* (ver também a expressão de indexação).

Expressões resultantes de avaliação de operadores

Indexação

A indexação devolve o valor de uma posição de memória indicada por um ponteiro. Consiste de uma expressão ponteiro seguida do índice entre parênteses rectos. Se a expressão ponteiro for um *left-value*, então a expressão indexação poderá também ser um '*left-value* (excepto nas condições proibidas pelo tipo).

Exemplo (acesso à posição indicada por **p**): **p[0]**

Identidade e simétrico

Os operadores identidade (**+**) e simétrico (**-**) aplicam-se a inteiros e reais. Têm o mesmo significado que em C.

Reserva de memória

A expressão reserva de memória **[]** devolve o ponteiro que aponta para a zona de memória, na pilha da função actual, contendo espaço suficiente para o número de reais indicados pelo seu argumento inteiro.

Exemplo (reserva vector com 5 reais, apontados por **p**): **<%>p=[5]**

Expressão de indicação de posição

O operador sufixo **?** aplica-se a *left-values*, retornando o endereço correspondente.

Exemplo (indica o endereço de **a**): **a?**

Exemplos

Os exemplos não são exaustivos e não ilustram todos os aspectos da linguagem. Podem obter-se outros na página da disciplina.

Programa com vários módulos

Definição da função *factorial* num ficheiro (**factorial.zu**):

```
#factorial!(#n) = 1 {
  [n > 1] ? factorial = n * factorial(n-1); : factorial = 1;
}
```

Exemplo da utilização da função *factorial* num outro ficheiro (**main.zu**):

```
// external builtin functions
#argc?()
```

```
$argv?(#n)
#atoi?($s)

// external user functions
#factorial?(#n)

// the main function
#zu!() = 0 {
  #f = 1;
  "Teste para a função factorial"!!
  [argc() == 2] # f = atoi(argv(1));
  f! "! = "! factorial(f)!!
}
```

Outros testes

Estão disponíveis outros pacotes de testes.

Omissões e Erros

Casos omissos e erros serão corrigidos em futuras versões do manual de referência.