Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №2 по теме: «Производные структуры данных. Структура данных „строка“»

Выполнил:  
студент группы ПВ-21  
Адаменко И. И.

Проверил:  
профессор  
Синюк В. Г.

Белгород  
2013

Цель работы: изучение встроенной структуры данных типа «строка», разработка и использование производных структур данных строкового типа.

## Задания

1. Для СД типа «строка» определить:
   1. Абстрактный уровень представления СД:
      1. Характер организованности и изменчивости.
      2. Набор допустимых операций.
   2. Физический уровень представления СД:
      1. Схему хранения.
      2. Объем памяти, занимаемый экземпляром СД.
      3. Формат внутреннего представления СД и способ его интерпретации.
      4. Характеристику допустимых значений.
      5. Тип доступа к элементам.
   3. Логический уровень представления СД. Способ описания СД и ее экземпляра на языке программирования.
2. Реализовать СД строкового типа в соответствии с вариантом индивидуального задания в виде модуля на языке Pascal. Определить и обработать исключительные ситуации.
3. Разработать программу на языке Pascal для решения задачи в соответствии с вариантом индивидуального задания с использованием модулей, полученных в результате выполнения п. 2.

### Индивидуальное задание

|  |  |  |
| --- | --- | --- |
| Номер варианта | Номер формата | Задача |
| 1 | 1 | 1 |

### Задача

Заголовок: procedure Copies(st1: t\_string; var st2: t\_string; number: byte);  
Назначение: копирование строки s1 в строку s2 n раз.  
Входные параметры: s1, n.  
Выходные параметры: s2.

### Формат

Спецификация СД на языке Pascal:

|  |
| --- |
| **UNIT** SD\_STRING; |
|  |
| *{-----------------------------------------------------------------------------}* |
| **INTERFACE** |
| *{-----------------------------------------------------------------------------}* |
|  |
| **const** *{определение исключительных ситуаций}* |
|  |
| **type** t\_string = **array** [1..256] **of char**; *{признак конца строки —   символ с кодом 0}* |
|  |
| **function** Comp(st1, st2: t\_string; **var** fl: **shortint**): **boolean**; |
| **function** Length(st: t\_string): **bytebyte**; |
| **function** Pos(st2, st1: t\_string): **word**; |
|  |
| **procedure** WriteToStr(**var** st: t\_string; s: **string**); |
| **procedure** WriteFromStr(**var** s: **string**; st: t\_string); |
| **procedure** InputStr(**var** st: t\_string); |
| **procedure** OutputStr(**const** st: t\_string) |
| **procedure** Insert(st2: t\_string; **var** st: t\_string; index: **byte**); |
| **procedure** Concat(**const** st1, st2: t\_string; **var** st3: t\_string); |
| **procedure** Copy(st1: t\_string; index, count: **byte**; **var** st2: t\_string); |
|  |
| **var** |
| StrError: *{тип переменной ошибки}* |

## Характеристика СД типа «строка»

1. Абстрактный уровень представления СД:
   1. По характеру организованности и изменчивости данный тип упорядоченный и статический.
   2. К типу применимы такие операции, как: определение длины, присваивание, конкатенация, выделение подстроки, поиск вхождения и другие.
2. Физический уровень представления СД:
   1. Структура данных хранится последовательно.
   2. Размер памяти, занимаемой каждым экземпляром СД зависит от максимального количества элементов входящих в него и определяется формулой: , где — максимальное количество элементов в экземпляре.
   3. Значение переменной этого типа хранится в памяти как фиксированная последовательность байт, каждый из которых интерпретируется в символ.
   4. Допустимыми значениями для данной СД являются последовательности символов; каждый элемент СД представляет один символ.
   5. Тип доступа к элементам данной СД прямой.
3. На логическом уровне представления СД, то есть, на языке программирования Pascal экземпляр СД описывается следующим образом: var s: t\_string;

## Текст модуля для реализации СД типа «строка»

|  |  |
| --- | --- |
| 001 | **UNIT** SD\_STRING; |
| 002 |  |
| 003 | *{------------------------------------------------------------------------}* |
| 004 | **INTERFACE** |
| 005 | *{------------------------------------------------------------------------}* |
| 006 |  |
| 007 | **uses** CRT; |
| 008 |  |
| 009 | **const** |
| 010 | errors: **array** [1..3] **of** **string** =   (*'Source string is less than needed'*,   *'Destination string is overflow'*,  *'Destination index out of range values'*); |
| 011 | N = 255; |
| 012 |  |
| 013 | **type** |
| 014 | t\_string = **array** [1..N + 1] **of char**; |
| 015 |  |
| 016 | **function** Comp(st1, st2: t\_string; **var** fl: **shortint**): **boolean**; |
| 017 | **function** Length(st: t\_string): **byte**; |
| 018 | **function** Pos(st2, st1: t\_string): **byte**; |
| 019 |  |
| 020 | **procedure** WriteToStr(**var** st: t\_string; s: **string**); |
| 021 | **procedure** WriteFromStr(**var** s: **string**; st: t\_string); |
| 022 | **procedure** InputStr(**var** st: t\_string); |
| 023 | **procedure** OutputStr(**const** st: t\_string); |
| 024 | **procedure** Insert(st2: t\_string; **var** st1: t\_string; index: **byte**); |
| 025 | **procedure** Concat(**const** st1, st2: t\_string; **var** st3: t\_string); |
| 026 | **procedure** Copy(st1: t\_string; index, count: **byte**; **var** st2: t\_string); |
| 027 | **procedure** CheckErrors; |
| 028 |  |
| 029 | **var** |
| 030 | StrError: **byte**; |
| 031 |  |
| 032 | *{------------------------------------------------------------------------}* |
| 033 | **IMPLEMENTATION** |
| 034 | *{------------------------------------------------------------------------}* |
| 035 |  |
| 036 | **function** Comp(st1, st2: t\_string; **var** fl: **shortint**): **boolean**; |
| 037 | **var** |
| 038 | i: **byte**; |
| 039 |  |
| 040 | **begin** |
| 041 |  |
| 042 | i := 1; |
| 043 |  |
| 044 | **while** (**boolean**(st1[i]) **and** **boolean**(st2[i])   **and** (st1[i] = st2[i])) **do** |
| 045 | inc(i); |
| 046 |  |
| 047 | **if** (st1[i] = st2[i]) **then** |
| 048 | **begin** |
| 049 | Comp := TRUE; |
| 050 | fl := 0; |
| 051 | **end** |
| 052 | **else** |
| 053 | **begin** |
| 054 |  |
| 055 | Comp := FALSE; |
| 056 | **if** (st1[i] > st2[i]) **then** |
| 057 | fl := 1 |
| 058 | **else** |
| 059 | fl := 2; |
| 060 | **end**; |
| 061 |  |
| 062 | **end**; |
| 063 |  |
| 064 | **function** Length(st: t\_string): **byte**; |
| 065 | **var** |
| 066 | i: **byte**; |
| 067 |  |
| 068 | **begin** |
| 069 |  |
| 070 | i := 1; |
| 071 | **while** (**boolean**(st[i])) **do** |
| 072 | inc(i); |
| 073 |  |
| 074 | Length := i - 1; |
| 075 |  |
| 076 | **end**; |
| 077 |  |
| 078 | **function** Pos(st2, st1: t\_string): **byte**; |
| 079 | **var** |
| 080 | i, len1, len2: **byte**; |
| 081 | fl: **shortint**; |
| 082 | finded: **boolean**; |
| 083 | tmp\_st: t\_string; |
| 084 |  |
| 085 | **begin** |
| 086 |  |
| 087 | finded := FALSE; |
| 088 | i := 1; |
| 089 | len1 := Length(st1); |
| 090 | len2 := Length(st2); |
| 091 |  |
| 092 | **while** (**not** finded **and** **boolean**(st1[i])  **and** (len1 - i >= len2)) **do** |
| 093 | **if** (st2[1] = st1[i]) **then** |
| 094 | **begin** |
| 095 | Copy(st1, i, len2, tmp\_st); |
| 096 | **if** (StrError = 0) **then** |
| 097 | finded := Comp(st2, tmp\_st, fl) |
| 098 | **else** |
| 099 | StrError := 0; |
| 100 | inc(i); |
| 101 | **end** |
| 102 | **else** |
| 103 | inc(i); |
| 104 |  |
| 105 | **if** (finded) **then** |
| 106 | Pos := i – 1 |
| 107 | **else** |
| 108 | Pos := 0; |
| 109 |  |
| 110 | **end**; |
| 111 |  |
| 112 | **procedure** WriteToStr(**var** st: t\_string; s: **string**); |
| 113 | **var** |
| 114 | i, len, j: **byte**; |
| 115 |  |
| 116 | **begin** |
| 117 |  |
| 118 | len := length(s); |
| 119 |  |
| 120 | **if** (len + Length(st) > N) **then** |
| 121 | **begin** |
| 122 | StrError := 2; |
| 123 | exit; |
| 124 | **end** |
| 125 | **else** |
| 126 | **begin** |
| 127 | i := 1; |
| 128 | **while** (**boolean**(st[i])) **do** inc(i); |
| 129 |  |
| 130 | j := 1; |
| 131 | **while** (j <= len) **do** |
| 132 | **begin** |
| 134 | st[i] := s[j]; |
| 134 | inc(i); |
| 135 | inc(j); |
| 136 | **end**; |
| 137 | **end**; |
| 138 |  |
| 139 | **end**; |
| 140 |  |
| 141 | **procedure** WriteFromStr(**var** s: **string**; st: t\_string); |
| 142 | **var** |
| 143 | i, len: **byte**; |
| 144 |  |
| 145 | **begin** |
| 146 |  |
| 147 | len := length(s); |
| 148 |  |
| 149 | **if** (len + Length(st) > N) **then** |
| 150 | **begin** |
| 151 | StrError := 2; |
| 152 | exit; |
| 153 | **end** |
| 154 | **else** |
| 155 | **begin** |
| 156 | i := 1; |
| 157 | **while** (**boolean**(st[i])) **do** |
| 158 | **begin** |
| 159 | s := s + st[i]; |
| 160 | inc(i); |
| 161 | **end**; |
| 162 | **end**; |
| 163 |  |
| 164 | **end**; |
| 165 |  |
| 166 | **procedure** InputStr(**var** st: t\_string); |
| 167 | **var** |
| 168 | tmp\_st: t\_string; |
| 169 | len: **byte**; |
| 170 | c: **char**; |
| 171 |  |
| 172 | **begin** |
| 173 |  |
| 174 | len := Length(st); |
| 175 | **if** (len = N) **then** |
| 176 | **begin** |
| 177 | StrError := 2; |
| 178 | exit; |
| 179 | **end** |
| 180 | **else** |
| 181 | **begin** |
| 182 | Copy(st, 1, len, tmp\_st); |
| 183 |  |
| 184 | read(c); |
| 185 | inc(len); |
| 186 | **while** ((c <> **#10**) **or** (c <> **#13**)) **do** |
| 187 | **begin** |
| 188 | **if** (len <= N) **then** |
| 189 | tmp\_st[len] := c |
| 190 | **else** |
| 191 | **begin** |
| 192 | StrError := 2; |
| 193 | exit; |
| 194 | **end**; |
| 195 |  |
| 196 | inc(len); |
| 197 | **end**; |
| 198 | **end**; |
| 199 |  |
| 200 | tmp\_st[len] := **#0**; |
| 201 | st := tmp\_st; |
| 202 |  |
| 203 | **end**; |
| 204 |  |
| 205 | **procedure** OutputStr(**const** st: t\_string); |
| 206 | **var** |
| 207 | i, len: **byte**; |
| 208 |  |
| 209 | **begin** |
| 210 |  |
| 211 | i := 1; |
| 212 | len := Length(st); |
| 213 |  |
| 214 | **while** (i <= len) **do** |
| 215 | **begin** |
| 216 | write(st[i]); |
| 217 | inc(i); |
| 218 | **end**; |
| 219 |  |
| 220 | **end**; |
| 221 |  |
| 222 | **procedure** Insert(st2: t\_string; **var** st1: t\_string; index: **byte**); |
| 223 | **var** |
| 224 | len, len2: **byte**; |
| 225 | tmp\_st: t\_string; |
| 226 |  |
| 227 | **begin** |
| 228 |  |
| 229 | len := Length(st1); |
| 230 | len2 := Length(st2); |
| 231 |  |
| 232 | **if** (len + len2 > N) **then** |
| 233 | **begin** |
| 234 | StrError := 2; |
| 235 | exit; |
| 236 | **end** |
| 237 | **else** **if** ((index > N) **or** (index < 1)) **then** |
| 238 | **begin** |
| 239 | StrError := 3; |
| 240 | exit; |
| 241 | **end** |
| 242 | **else** |
| 243 | **begin** |
| 244 | Copy(st1, index, len2, tmp\_st); |
| 245 | st1[index] := **#0**; |
| 246 | Concat(st1, st2, st1); |
| 247 | Concat(st1, tmp\_st, st1); |
| 248 | **end**; |
| 249 |  |
| 250 | **end**; |
| 251 |  |
| 252 | **procedure** Concat(**const** st1, st2: t\_string; **var** st3: t\_string); |
| 253 | **var** |
| 254 | len, len2, len3, i: **byte**; |
| 255 | tmp\_st: t\_string; |
| 256 |  |
| 257 | **begin** |
| 258 |  |
| 259 | len := Length(st1); |
| 260 | len2 := Length(st2); |
| 261 | len3 := Length(st3); |
| 262 |  |
| 263 | **if** (len + len2 + len3 > N) **then** |
| 264 | **begin** |
| 265 | StrError := 2; |
| 266 | exit; |
| 267 | **end** |
| 268 | **else** |
| 269 | **begin** |
| 270 | Copy(st3, 1, len3, tmp\_st); |
| 271 | inc(len3); |
| 272 |  |
| 273 | i := 1; |
| 274 | **while** (i <= len) **do** |
| 275 | **begin** |
| 276 | tmp\_st[len3] := st1[i]; |
| 277 | inc(i); |
| 278 | inc(len3); |
| 279 | **end**; |
| 280 |  |
| 281 | i := 1; |
| 282 | **while** (i <= len2) **do** |
| 283 | **begin** |
| 284 | tmp\_st[len3] := st2[i]; |
| 285 | inc(i); |
| 286 | inc(len3); |
| 287 | **end**; |
| 288 | **end**; |
| 289 |  |
| 290 | tmp\_st[i] := **#0**; |
| 291 | st3 := tmp\_st; |
| 292 |  |
| 293 | **end**; |
| 294 |  |
| 295 | **procedure** Copy(st1: t\_string; index, count: **byte**; **var** st2: t\_string); |
| 296 | **var** |
| 297 | len, last, i: **byte**; |
| 298 |  |
| 299 | **begin** |
| 300 |  |
| 301 | len := Length(st1); |
| 302 | last := index + count; |
| 303 | **if** (len < (last - 1)) **then** |
| 304 | **begin** |
| 305 | StrError := 1; |
| 306 | exit; |
| 307 | **end** |
| 308 | **else** |
| 309 | **begin** |
| 310 | i := 1; |
| 311 | **while** (index <= last) **do** |
| 312 | **begin** |
| 313 | st2[i] := st1[index]; |
| 314 | inc(i); |
| 315 | inc(index); |
| 316 | **end**; |
| 317 | st2[i] := **#0**; |
| 318 | **end**; |
| 319 |  |
| 320 | **end**; |
| 321 |  |
| 322 | **procedure** CheckErrors; |
| 323 | **begin** |
| 324 |  |
| 325 | **if** (StrError <> 0) **then** |
| 326 | **begin** |
| 327 | textcolor(14); |
| 328 | writeln(*'Error! '*, errors[StrError]); |
| 329 | readln; |
| 330 | halt; |
| 331 | **end**; |
| 332 |  |
| 333 | **end**; |
| 334 |  |
| 335 | **begin** |
| 336 |  |
| 337 | StrError := 0; |
| 338 |  |
| 339 | **end**. |

## Текст программы для решения задачи

|  |  |
| --- | --- |
| 01 | **program** sd\_lab\_2; |
| 02 |  |
| 03 | **uses** SD\_STRING, CRT; |
| 04 |  |
| 05 | **procedure** Copies(st1: t\_string; **var** st2: t\_string; number: **byte**); |
| 06 | **var** |
| 07 | len, i, j, k: **byte**; |
| 08 |  |
| 09 | **begin** |
| 10 |  |
| 11 | len := Length(st1); |
| 12 |  |
| 13 | **if** ((len \* number) > N) **then** |
| 14 | **begin** |
| 15 | StrError := 2; |
| 16 | exit; |
| 17 | **end** |
| 18 | **else** |
| 19 | **begin** |
| 20 | k := 1; |
| 21 | i := 1; |
| 22 | **while** (i <= number) **do** |
| 23 | **begin** |
| 24 | j := 1; |
| 25 | **while** (j <= len) **do** |
| 26 | **begin** |
| 27 | st2[k] := st1[j]; |
| 28 | inc(j); |
| 29 | inc(k); |
| 30 | **end**; |
| 31 | inc(i); |
| 32 | **end**; |
| 33 | **end**; |
| 34 |  |
| 35 | st2[k] := **#0**; |
| 36 |  |
| 37 | **end**; |
| 38 |  |
| 39 | **var** |
| 40 | st1, st2: t\_string; |
| 41 |  |
| 42 | **begin** |
| 43 |  |
| 44 | WriteToStr(st1, *'This is the first string!'*); |
| 45 | CheckErrors; |
| 46 | WriteToStr(st2, *'This is the second string!'*); |
| 47 | CheckErrors; |
| 48 |  |
| 49 | textcolor(White); |
| 50 | write(*'First string: < '*); |
| 51 | textcolor(Green); |
| 52 | OutputStr(st1); |
| 53 | textcolor(White); |
| 54 | writeln(*' >'*); |
| 55 |  |
| 56 | write(*'Second string: < '*); |
| 57 | textcolor(Green); |
| 58 | OutputStr(st2); |
| 59 | textcolor(White); |
| 60 | writeln(*' >'*); |
| 61 |  |
| 62 | Copies(*'Deleted! '*, st1, 10); |
| 63 | CheckErrors; |
| 64 |  |
| 65 | write(*'First string, after Copies(10): < '*); |
| 66 | textcolor(Green); |
| 67 | OutputStr(st1); |
| 68 | textcolor(White); |
| 69 | writeln(*'>'*); |
| 70 | readln; |
| 71 |  |
| 72 | Copies(*'Deleted! '*, st2, 100); |
| 73 | CheckErrors; |
| 74 |  |
| 75 | write(*'First string, after Copies(100): < '*); |
| 76 | textcolor(Green); |
| 77 | OutputStr(st1); |
| 78 | textcolor(White); |
| 79 | writeln(*'>'*); |
| 80 | readln; |
| 81 |  |
| 82 | **end**. |