Data Processing I and Strings

Chan Siu Man
siuman@hkoi.org

Hong Kong Olympiad in Informatics

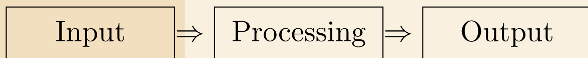
January 29, 2005

Agenda

1. Overview
2. Data Types
   - Overview
   - Pascal Data Types
   - C/C++ Data Types
   - General Suggestions
3. Processing
   - String Declaration
   - String Operations
   - String Conversion
   - Final Words on Strings
   - General Techniques
   - Arrays and Packed Data Types
4. Input/Output

Overview

Program flow:



Input:

- Input tokenization and parsing.

Processing:

- Sorting and searching;
- Simple calculation and manipulation; and
- Statistics computation.

Output:

- Output formatting.

- Data processing is not an algorithm. There is no general method for solving all problems. Tricks used vary from problem to problem.
- As long as the program does not exceed time limit, you are free to use any method that can give the correct output.
- Hint: modularity and familiarity with STL can give you huge advantages.

- Data processing gives you the foundation for computer programming.
- Trains your ability to:
  - Read the problems carefully and precisely;
  - Analyze and solve problems;
  - Choose the most suitable algorithms and data structures; and
  - Write programs that exactly implement your idea.
- Practice makes perfect. Practice data processing can make you code faster.

Data Types

It is important to employ the *right* data types for writing every programs. You may need to consider:

- Data ranges;
- Memory usage; and
- Time needed per operation.

Overview

Category	Pascal	C/C++
Ordinal	Integer, Char, Boolean, ...	int, char, bool, ...
Floating Point	Real, Double, Extended, ...	float, double, ...
Array	Array[*Range*] of *Type*	*Type*[*Range*]
String	PChar, String, ANSIString	char[*Range*], string
Packed Data Type	Record	struct
Others	Ptr, Set, ...	**Type*, enum, ...

The table is by no means complete and the correspondence is not exact.

Pascal Data Types (FreePascal)

Suggested data types: (use them if range fits and memory permits)

- `LongInt`: 32-bit signed integer
  $[-2^{31}, 2^{31} - 1] \iff [-2147483648, 2147483647]$
- `Int64`: 64-bit signed integer
  $[-2^{63}, 2^{63} - 1] \iff [-9223372036854775808, 9223372036854775807]$
- `Double`: IEEE-754 double type
- `Char`: 8-bit character
- `Boolean`: 8-bit truth value
- `String`: 256-byte string

Less common data types: (use them when memory is highly constrained)

- `Integer`: 16-bit signed integer
  $[-2^{15}, 2^{15}-1] \iff [-32768, 32767]$
- `Byte`: 8-bit unsigned integer
  $[0, 2^{8}-1] \iff [0, 255]$
- `Word`: 16-bit unsigned integer
  $[0, 2^{16}-1] \iff [0, 65535]$
- `Real`: architecture and compiler dependent floating number

C/C++ Data Types (GCC IA32/x86)

Use `unsigned` and `signed` to select the range.

- `long`: 32-bit signed integer
  $[-2^{31}, 2^{31}-1] \iff [-2147483648, 2147483647]$
- `long long`: 64-bit signed integer
  $[-2^{63}, 2^{63}-1] \iff$
  $[-9223372036854775808, 9223372036854775807]$
- `double`: IEEE-754 double type
- `char`: 8-bit signed integer
- `bool`: 8-bit "truth value"

General Suggestions

- Use ordinal types if possible, even for problems concerning decimals.
  - More accurate; and
  - Operate faster generally.
- Use most accurate floating type if possible.
  - Rounding error is unavoidable. More bits means less error is accumulated.

Runtime and Computational Error

Beware of the followings:

- Overflow Error
  - Check the extreme values.
  - Pay special attention to multiplication. Type casting to longer data type may be needed to compute intermediate values.
  - If memory permits, use `LongInt` and `long`. Use `Int64` and `long long` for larger problems.
- Underflow Error
  - Write subroutines for comparison of floating point numbers.
- Division by Zero
  - Pay special attention to division and modulus.

Fix-Sized String

String is stored and manipulated differently in Pascal and C/C++.

- Pascal: `String`
  - 1 byte storing length and 255 bytes storing array of characters.
  - Total size is 256 bytes by default.
  - `Var s: String; { occupies 256 bytes }`
  - `Var t: String[20]; { occupies 21 bytes }`
- C/C++: character traits
  - Null-terminating array of characters. Last byte (Null-character) denotes the end.
  - Though no header file is needed for declaration of strings, `<string.h>` has better be included for their manipulation.
  - `char s[256]; /* occupies 256 bytes */`
  - `char t[21]; /* occupies 21 bytes */`

Dynamic-Sized String

- Pascal: `ANSIString`
  - Can be manipulated by almost all functions defined for `string`.
  - `var s: ANSIString;`
- C++: `string`
  - Implemented by STL. Header file `<string>` has to be included.
  - Manipulated differently from character traits.
  - `string s;`

String Assignments

- Pascal: normal assignment
  - `s := 'abcde';`
  - `t := s;`
  - `s[2] := '9';`
- C: by `char *strcpy(char *dest, const char *src);`
  - `strcpy(s, "abcde");`
  - `strcpy(t, s);`
  - `s[1] = '9';`
- C++: normal assignment
  - `s = "abcde";`
  - `t = s;`
  - `s[1] = '9';`

Substring

- Pascal: use

  `Function Copy(Const S: String; Index: Integer; Count: Integer): String;`

  - `s := copy(t, 2, 4);`
- C: by `char *strncpy(char *dest, const char *src, size_t n);`
  - Beware of pointers. Modified character traits may not be terminated with Null character. Watch out!
  - `strncpy(s, t + 1, 4);`
- C++: use

  `basic_string string::substr(size_type pos = 0, size_type n = npos) const`

  - `s = t.substr(1, 4);`
  - `s = t.substr(1);`

String Length

- Pascal: use `Function Length(S: String): Integer;`
  - `k := length(s);`
- C: use `size_t strlen(const char *s);`
  - `k = strlen(s);`
- C++: use `size_type string::length() const`
  - `k = s.length();`

String Concatenation

- Pascal: use `Function Concat (S1,S2 [,S3, ... ,Sn]) : String;` or "+" operator.
  - `s := concat(s, t, u);`
  - `s := s + t + u;`
  - The above are equivalent.
- C: use `char *strcat(char *dest, const char *src);`
  - `strcat(s, t); strcat(s, u);`
  - Can join at most two character traits at the same time.
- C++: use `operator+` of `string`
  - `s = s + t + u;`
  - `s += t + u;`
  - The above are equivalent.

String Comparison

Compare string by lexicographical ordering.

- Pascal: usual comparison
  - `s > t`
  - `s <> t`
- C: use `int strcmp(const char *s1, const char *s2);`
  - `strcmp(s, t) > 0`
  - `strcmp(s, t) != 0`
- C++: usual comparison
  - `s > t`
  - `s != t`

String Insertion

- Pascal: use
  `Procedure Insert(Const Source: String; var S: String; Index: Integer);`
  - `insert(s, t, 3);`
- C: use a combination of `strcpy()` and `strncpy()` (or `memmove()`)
  - Troublesome!
- C++: use
  `basic_string& string::insert(size_type pos, const basic_string& s)`
  - `s.insert(2, t);`
  - There are more overloaded `string::insert(...)` for your convenience. Use with care.

String Deletion

- Pascal: use

  `Procedure Delete(var S: string; Index: Integer; Count: Integer);`

  - `delete(s, 3, 4);`
- C: use a combination of `strcpy()` and `strncpy()` (or `memmove()`)
  - Troublesome!
- C++: use

  `basic_string& string::erase(size_type pos = 0, size_type n = npos)`

  - `s.erase(2, 4);`
  - There are more overloaded `string::erase(...)` for your convenience. Use with care.

String Searching

Find a substring within a given string.

- Pascal: use
  `Function Pos(Const Substr: String; Const S: String): Integer;`
  - `k := pos('12', p);`
  - `pos('x', p) = 0`
  - 0 for not found.
- C: use `char *strstr(const char *haystack, const char *needle);`
  - `r = strstr(p, "12"); k = r - p;`
  - `strstr(p, "x") == NULL`
  - Null for not found.
- C++: use `size_type find(const charT* s, size_type pos = 0) const`
  - `k = p.find("12");`
  - `p.find("x") == string::npos`
  - `string::npos` for not found.

String Conversion

Conversion between numeric data types and strings.

- Pascal: use `Procedure Val(const S: string; var V; var Code: word);` and `Procedure Str(Var X[:NumPlaces[:Decimals]]; Var S: String);`
  - `val(t, k, e);`
  - `str(k, t);`
- C: use `int sscanf(const char *str, const char *format, ...);` and `int sprintf(char *str, const char *format, ...);`
  - `sscanf(t, "%d", &k);`
  - `sprintf(s, "%d", k);`
- C++: use `<sstream>`

Final Words on Strings

- Pascal
  - String of length one is not equal to character.
    `copy(s, 1, 1) <> s[1]`
  - Result will be truncated if index exceeds length.
- C
  - Be careful not to overwrite the `'\0'`;
- C++
  - Use `const charT* c_str() const` and `basic_string(const charT*)` to convert between `string` and character traits.

Do explore with strings!

General Techniques

- Read the question carefully.
  Data processing questions usually ha have lengthy, complicated and perhaps misleading “story”. Have a clear mind!
- Focus on how to process the input to give the output (algorithms).
- Decide how to store input data for manipulation (data structures).
- Enhance modularity, at least logically. Do input parsing, data processing and output formatting separately.

- Select the most suitable data structures and algorithms. Consider both sides at the same time, as they are dependent on each other.
- Roughly estimate the complexity of the algorithm used, in terms of
  - Run time complexity; and
  - Memory complexity.
- Try to analyze the algorithm using so-called "Big-O" notation.
- The most important parameter is the actual number of operations and the estimated actual running time.
- Remember to KISS (Keep It Simple, Stupid).

Arrays and Packed Data Types

- Whenever you want to copy and paste source codes, consider using looping with arrays. Arrays give you the power to iterate.
- Use `Record`/`struct` to pack you data so as to facilitate manipulation, e.g. swapping.
- To store sequences of related data, you can choose between *parallel arrays* and *array of records*. Array of records is preferred to parallel arrays, as it combines the best of both worlds.

Input/Output

- Input may be awkwardly formatted and difficult to parse.
- You can store the most recent input line by line in memory, i.e. to *buffer* the input; then split the input into a sequence of tokens according to some deliminators, i.e. to *tokenize* the input, with string processing techniques.
- Output formatting may be demanding and difficult to write.
- You may form the output part by part, by successively writing to the memory, i.e. to *buffer* the output. When the output is completed, the program output all the lines in the buffer.
- To master input/output, some study of the manual/reference to the programming language used is indispensable. Google search!