ANSWERS TO THEORETICAL QUESTIONS USED FOR CSC600 EXAMS

CSc 600 exams include writing programs and answering theoretical questions. It seems that for many students writing programs is easier than answering theoretical questions. This text should help in answering theoretical questions.

A typical theoretical question is "*Define (or describe) and exemplify the concept of <something>*". Usually <something> is one of important general concepts used in programming languages. For example, "*describe and exemplify the concept of short-circuit evaluation of logic conditions*".

The first step in answering many questions is to give the *definition* of some concept. In Latin language, many centuries ago, the following was the rule of making definitions:

*"**Definitio fiat per genus proximum et differentiam specificam**"*

A liberal translation of this rule is "The definition should be based on the closest general concept and a specific difference", or "Each concept can be defined through its parent concept and a specific difference".

Suppose that the question is to define Fortran. The definition can be the following: "Fortran is ***a programming language*** that is ***used for scientific computation***". Here "a programming language" represents the parent concept, because Fortran belongs to the large class of programming languages. However, there are many other languages in this class, and to define Fortran we need a *specific difference*, i.e. something that will differentiate Fortran from other languages in its class. Therefore, the fact that Fortran is "*used for scientific computation*" is a specific difference that differentiates Fortran from COBOL (because COBOL is used for business, and not scientific problems), or from PROLOG (because PROLOG is used for solving logic problem and building expert systems and not for scientific computation), and from many other languages. Therefore, whenever you have to define something, please keep this "theory" in mind.

Let us now illustrate the above presentation with examples of theoretical questions.

Question:

Explain the concept of Fortran statement function and give an example of its use.

Answer:

Statement function (SF) is a single-line single-statement function definition that can be inserted before the first executable statement in Fortran (and BASIC) programs and subprograms. A general form of statement function is

```
function_name( list of arguments ) = expression
```

Main features of statement functions are:

- SF defines a function in a single statement
- The type of SF name defines the type of the returned value.
- The scope of SF arguments is the SF body only (consequently, argument names can be reused in the program that contains the SF)
- SF is local to the program unit in which it is defined
- SF must be defined before the first executable program statement
- SF can call all library functions and all previously defined SF's
- SF can access and use variables defined in its environment.

Example of using SF's to define hyperbolic functions:

```
sh(x)  = (exp(x) - exp(-x))/2.
ch(x)  = sqrt(sh(x)**2 + 1.)
th(x)  = sh(x) / ch(x)
cth(x) = 1. / th(x)
```

Question:

Describe and exemplify the concept of short-circuit evaluation of logic conditions.

Answer:

Short-circuit evaluation of logic conditions is a partial evaluation of logic conditions that is terminated as soon as the result becomes known.

Some programming languages support short-circuit evaluation (e.g. C), and some languages consider short-circuit evaluation an implementation dependent option (e.g. Pascal). Following is an example of unsafe program in Pascal:

```
while (i <= limit) and (list[i] <> value) do i:=i+1;
```

If the value is not in the list, at the end of the above linear search we have i=limit+1. If the compiler does not support short-circuit evaluation, it will evaluate (list[limit+1] <> value) and cause the out-of-bounds error. Therefore, Pascal programmers must not use the above linear search statement. On the other hand, C programmers can safely use the equivalent statement

```
while(i <= limit  &&  list[i] != value) ++i;
```