EUCIP
European Certification of
Informatics Professionals

EPL Syntax

EPL (EUCIP Programming Language) is designed to accommodate the need to test basic understanding of programming at the EUCIP core level. It is based on a subset of C, and is consistent with core programming constructs found in contemporary programming languages such as Java and C++. The main elements that have been removed from the original C syntax are:

"Shorthand" assignment operators such as += and &=.
Only simple identfiers (**char**, **int** and **float**) have been retained.
All sc specifiers (e.g. **auto**, **static**) have been removed.
All syntax pertaining to structures has been removed
"Non-procedural" statements (e.g. **goto**, **case** and **switch**) have been removed.
All pre-processor statements have been removed

It is believed that even with this simplified syntax most of the core aspects of programming related to algorithmic understanding, syntactic understanding and the flavour of contemporary programming have been retained.

It is expected that candidates to the core level "Build" module will be able to understand and answer questions that utilize EPL. Candidates are permitted to have a reference copy of the syntax when undergoing testing.

The EPL syntax is formally defined in the following paragraphs:

1. Expressions

expression:
> *primary*
> *- expression*
> *! expression*
> *expression binop expression*
> *lvalue = expression*
> *expression , expression*

primary:
> *identifier*
> *constant*
> *(expression)*
> *primary (expression- list$_{opt}$)*
> *primary [expression]*

lvalue:
> *identifier*
> *primary [expression]*
> *(lvalue)*

The primary-expression operators

> ( )   [ ]

have highest priority and group left-to-right. The unary operators

        -     !

have priority below the primary operators but higher than any binary operator, and group right-to-left. Binary operators all group left-to-right, and have priority decreasing as indicated (highest priority topmost):

> *binop:*
>   *   /
>   +   -
>   <   >   <=   >=
>   ==   !=
>   &&
> ||

Assignment operator (=) groups right-to-left.

The comma operator (,) has the lowest priority, and groups left-to-right.

2. Declarations

> *declaration:*
>   *decl-specifiers init-declarator- list$_{opt}$;*
>
> *decl-specifiers:*
>   *type-specifier decl-specifiers$_{opt}$*
>
> *type-specifier:*
>   **char**
>   **int**
>   **float**
>   *typedef-name*
>
> *init-declarator-list:*
>   *init-declarator*
>   *init-declarator , init-declarator- list*
>
> *init-declarator:*
>   *declarator initializer$_{opt}$*
>
> *declarator:*
>   *identifier*
>   *(declarator)*
>   *declarator* ()
>   *declarator* [*constant-expression$_{opt}$*]

initializer:
        = expression
        = { initializer-list }
        = { initializer-list , }

initializer-list:
        expression
        initializer- list , initializer-list
        { initializer-list }

3. Statements

compound-statement:
        {declaration-list$_{opt}$ statement-list$_{opt}$}

declaration- list:
        declaration
        declaration declaration- list

statement- list:
        statement
        statement statement-list

statement:
        compound-statement
        expression ;
        if *(expression) statement*
        if *(expression) statement* **else** *statement*
        while *(expression) statement*
        do *statement* **while** *(expression);*
        for *(expression-1$_{opt}$; expression-2$_{opt}$; expression-3$_{opt}$) statement*
        return *;*
        return *expression ;*
        ;

4. External definitions

program:
        external-definition
        external-definition program

external-definition:
        function-definition
        data-definition

function-definition:
      type-specifier$_{opt}$ function-declarator function-body

function-declarator:
      declarator (parameter- list $_{opt}$)

parameter-list:
      identifier
      identifier , parameter-list

function-body:
      type-decl-list function-statement

function-statement:
      { declaration-list $_{opt}$ statement-list }

data-definition:
      type-specifier $_{opt,}$ init-declarator-list $_{opt}$;
      type-specifier $_{opt}$ init-declarator-list $_{opt}$;

5. *Input/Output Standard for EPL*

printf ("message", variables list)
readf (variables list)