Składnia języka

EPL

EUCIP Programming Language

Język programowania EPL (EUCIP Programming Language) został zaprojektowany na potrzeby weryfikacji rozumienia podstawowych zasad programowania na poziomie EUCIP Core. Oparty jest na podzbiorze języka C i jest zgodny z kluczowymi konstrukcjami programistycznymi, jakie można znaleźć we współczesnych językach programowania, takich jak Java i C++.

Najważniejszymi elementami, jakie uległy zmianie w porównaniu do pierwotnej składni języka C są:

- „Skrócone" operatory przypisania, takie jak += oraz &=.
- Zachowane zostały tylko proste identyfikatory (char, int oraz float).
- Usunięte zostały wszystkie specyfikatory klas pamięci (np. auto, static).
- Usunięto wszystkie elementy składni związane ze strukturami.
- Usunięto wszystkie instrukcje „nieproceduralne" (np. goto, case oraz switch).
- Usunięto wszystkie instrukcje preprocesora.

Założono, że pomimo uproszczonej składni, zachowana została większość elementarnych zagadnień z zakresu programowania, związanych z rozumowaniem algorytmicznym, rozumieniem składni oraz ogólną wiedzą dotyczącą współczesnego programowania.

Od osób zdających drugi moduł („B: Wytwarzanie") certyfikatu EUCIP Core, oczekuje się zrozumienia treści pytań wykorzystujących język EPL. W związku z tym, dozwolone jest posiadanie przez kandydatów w czasie trwania testów, pomocy w postaci kopii dokumentu - składni języka EPL.

Składnia języka EPL jest formalnie zdefiniowana w postaci następujących paragrafów:

1 Wyrażenia

wyrażenie:

podstawowe
wyrażenie
! wyrażenie
wyrażenie <operator binarny> wyrażenie
lwartość = wyrażenie
wyrażenie, wyrażenie

podstawowe:

identyfikator
stała
(wyrażenie)
podstawowe (wyrażenie- lista $_{opcjonalne}$ *)*
podstawowe [wyrażenie]

lwartość:

identyfikator
podstawowe [wyrażenie]
(lwartość)

Operatory pierwszego stopnia:

( ) [ ]

Mają najwyższy priorytet i grupują operandy od lewej do prawej strony.

Operatory unarne (jednoargumentowe) :

- !

mają priorytet niższy niż operatory pierwszego stopnia, ale wyższy od wszystkich operatorów binarnych (dwuargumentowych) i grupują operandy od prawej do lewej strony. Wszystkie operatory binarne grupują operandy od lewej do prawej strony, a ich priorytety maleją zgodnie z zawartością poniższej tabeli (w pierwszym rzędzie operatory o najwyższym priorytecie):

Operatory binarne:

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

Operator przypisania (=) grupuje operandy od prawej do lewej strony.
Operator przecinka (,) ma najniższy priorytet i grupuje operandy od lewej do prawej strony.

2 Deklaracje

deklaracja:

specyfikatory_deklaracji lista_deklaratora_inicjalizacji opcjonalne

specyfikatory deklaracji:

specyfikator_typu specyfikator_deklaracji opcjonalne

specyfikatory typu:

char
int
float
nazwa_typedef

lista deklaratora inicjalizacji:

deklarator inicjalizacji
deklarator inicjalizacji, lista deklaratora inicjalizacji

deklarator inicjalizacji:

inicjalizator deklaratora opcjonalne

deklarator:

identyfikator
(deklarator)
deklarator ( )
deklarator [wyrażenie_stałe opcjonalne *]*

inicjalizator:

= wyrażenie
= { lista inicjalizacyjna }
= { lista inicjalizacyjna, }

lista inicjalizacyjna:

wyrażenie
lista inicjalizacyjna, lista inicjalizacyjna
{ lista inicjalizacyjna }

3 Instrukcje

***instrukcja złożona**:*

{ lista_deklaracji opcjonalne *lista _instrukcji* opcjonalne *}*

lista deklaracji:

deklaracja
deklaracja lista_deklaracji

lista instrukcji:

instrukcja
instrukcja lista_instrukcji

instrukcja:

instrukcja_złożona
wyrażenie ;
if *(wyrażenie) instrukcja*
if *(wyrażenie) instrukcja* ***else*** *instrukcja*
while *(wyrażenie) instrukcja*
do *instrukcja* ***while*** *(wyrażenie);*
for *(wyrażenie1*$_{opcjonalne}$*; wyrażenie2*$_{opcjonalne}$*; wyrażenie3*$_{opcjonalne}$*) instrukcja*
return *;*
return *wyrażenie ;*
;

4 Definicje zewnętrzne

program:

definicja_zewnętrzna
definicja_zewnętrzna program

definicja zewnętrzna:

definicja_funkcji
definicja_danych

definicja funkcji:

specyfikator_typu $_{opcjonalne}$ *deklaracja_funkcji ciało_funkcji*

deklarator funkcji:

deklarator (lista_parametrów $_{opcjonalne}$ *)*

lista parametrów:

identyfikator
identyfikator, lista_parametrów

ciało funkcji:

lista_deklaracji_typów instrukcja_funkcji

instrukcja funkcji:

{ lista_deklaracji $_{opcjonalne}$ *lista_instrukcji }*

definicja danych:

specyfikator_typu $_{opcjonalne}$*, lista_deklaratora_inicjalizacji* $_{opcjonalne}$ *;*
specyfikator_typu $_{opcjonalne}$ *lista_deklaratora_inicjalizacji* $_{opcjonalne}$ *;*

5 Standard wejścia / wyjścia dla języka EPL

Printf („tekst”, lista_zmiennych)
Readf (lista_zmiennych)