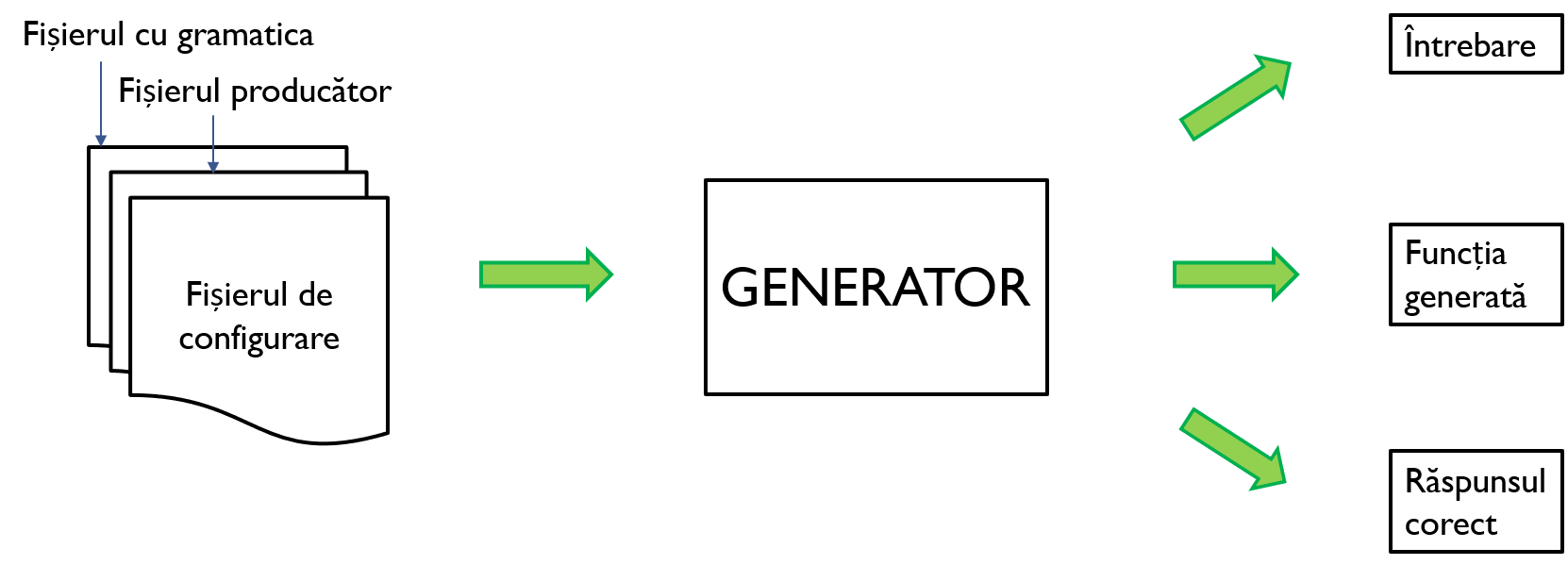
Există o multitudine de tipuri de exerciții care pot fi folosite pentru a evalua cunoștințele de programare ale unui student. În cadrul proiectului, au fost abordate exercițiile ce evaluează cunoașterea și aplicarea unor algoritmi, precum și exercițiile semantice ce constau în înțelegerea fluxului unui program.

Crearea primului tip de exerciții necesită ca generatorul să primească la intrare, un fișier de configurare în format JSON, care definește tiparul întrebării și porțiunile care vor fi înlocuite cu datele produse de programul C++ (fișierul producător), ce constă într-un algoritm precum parcurgerea în preordine/inordine/postordine a unui arbore sau determinarea arborelui parțial de cost minim (Algoritmul lui Kruskal).

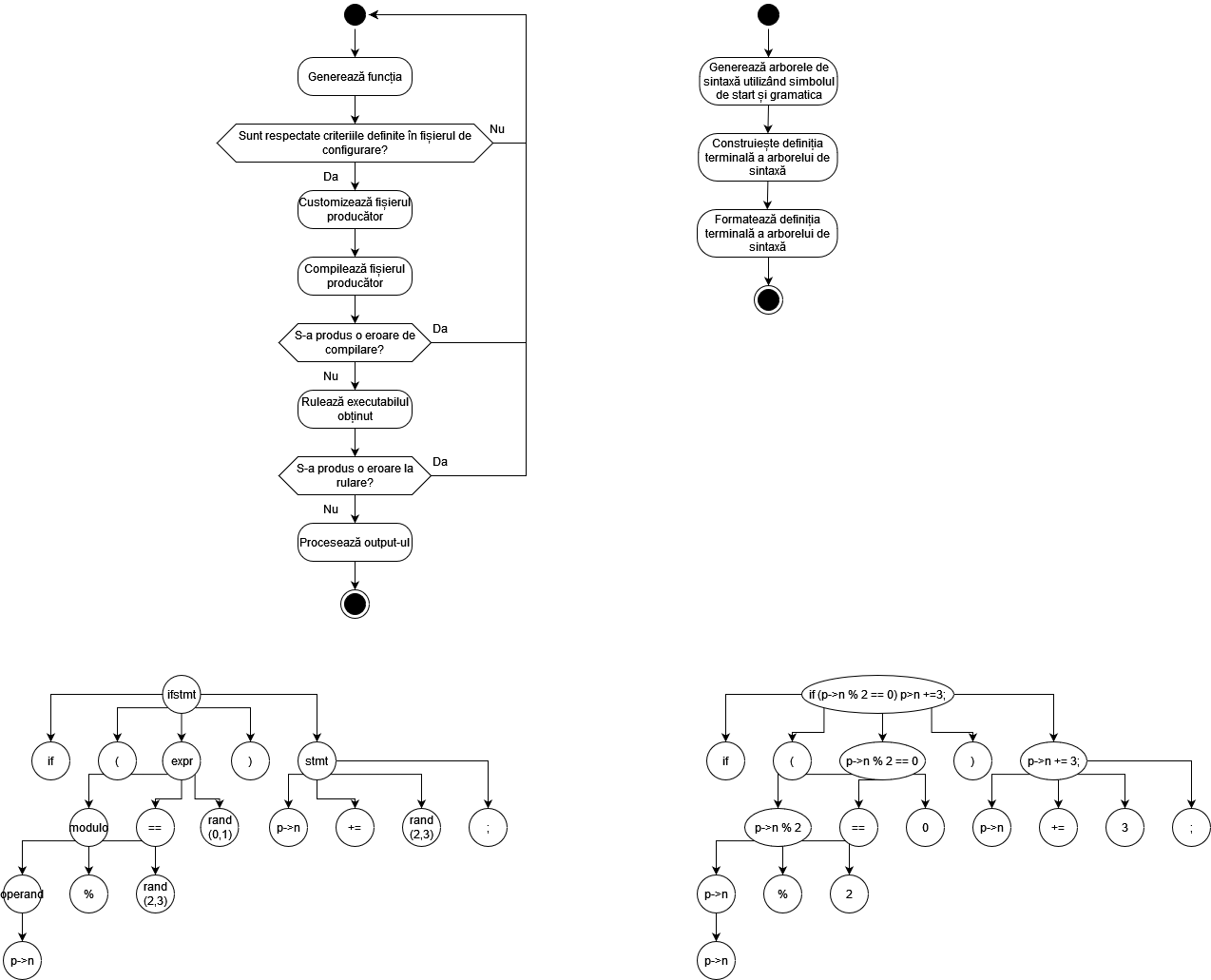


Al doilea tip de exerciții poate fi modelat printr-un triplet alcătuit dintr-o ipoteză, o secvență de cod și un răspuns corect. Generatorul primește ca date de intrare fișierul de configurare în format JSON, fișierul producător ce conține un program C++ care va fi customizat și fișierul cu gramatica ce va fi folosită pentru generarea funcției de customizare a fișierului producător în format JSON. Acesta verifică inițial dacă au fost stabiliți toți parametrii necesari în fișierul de configurare, dacă fișierul producător conține un punct de plecare (metoda main()) și corectitudinea gramaticii.

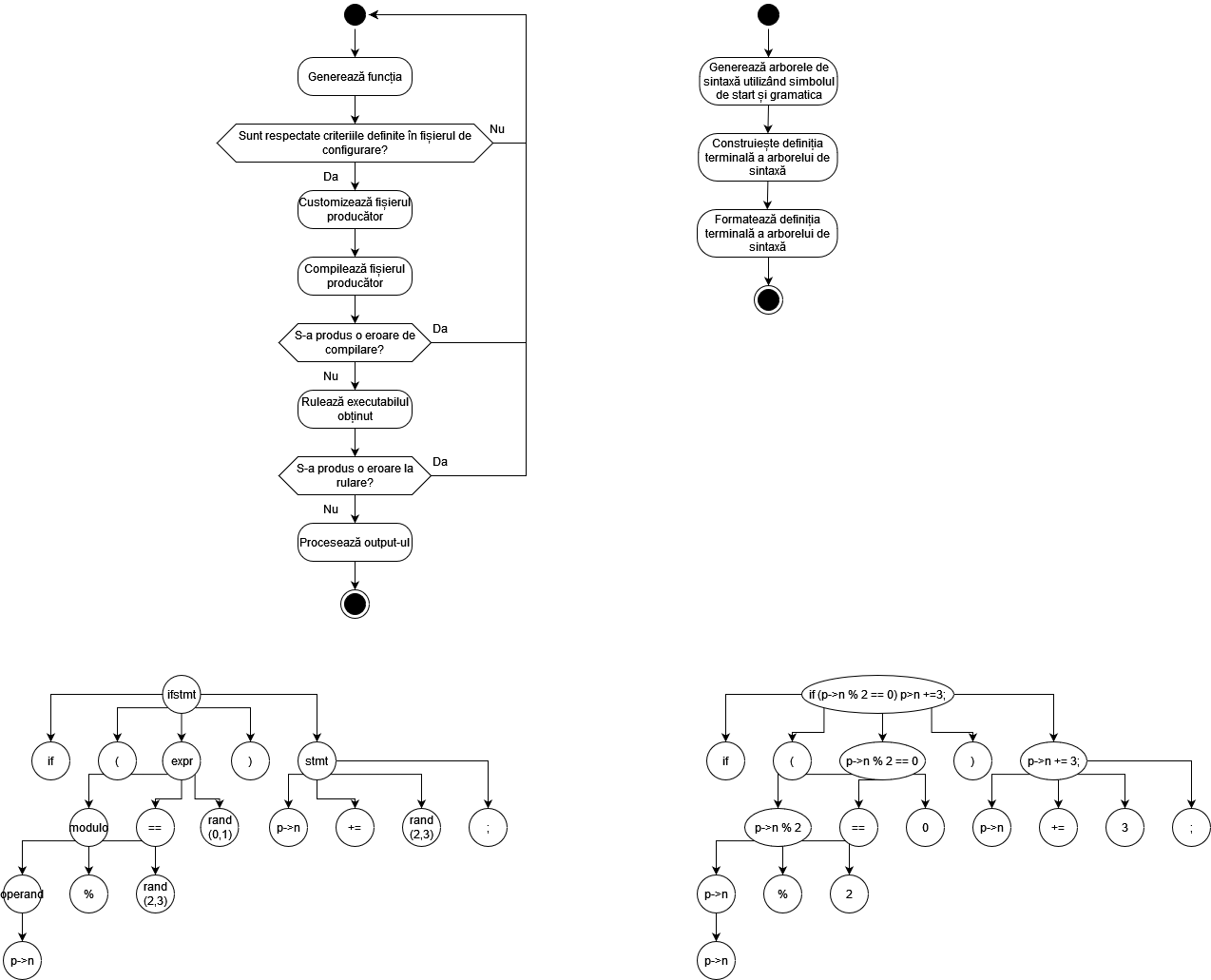


Pentru generarea întregului exercițiu, am definit un pipeline alcătuit din pașii următori: generarea funcției, personalizarea fișierului producător (șablonului), compilarea și execuția acestuia, precum și procesarea output-ului.

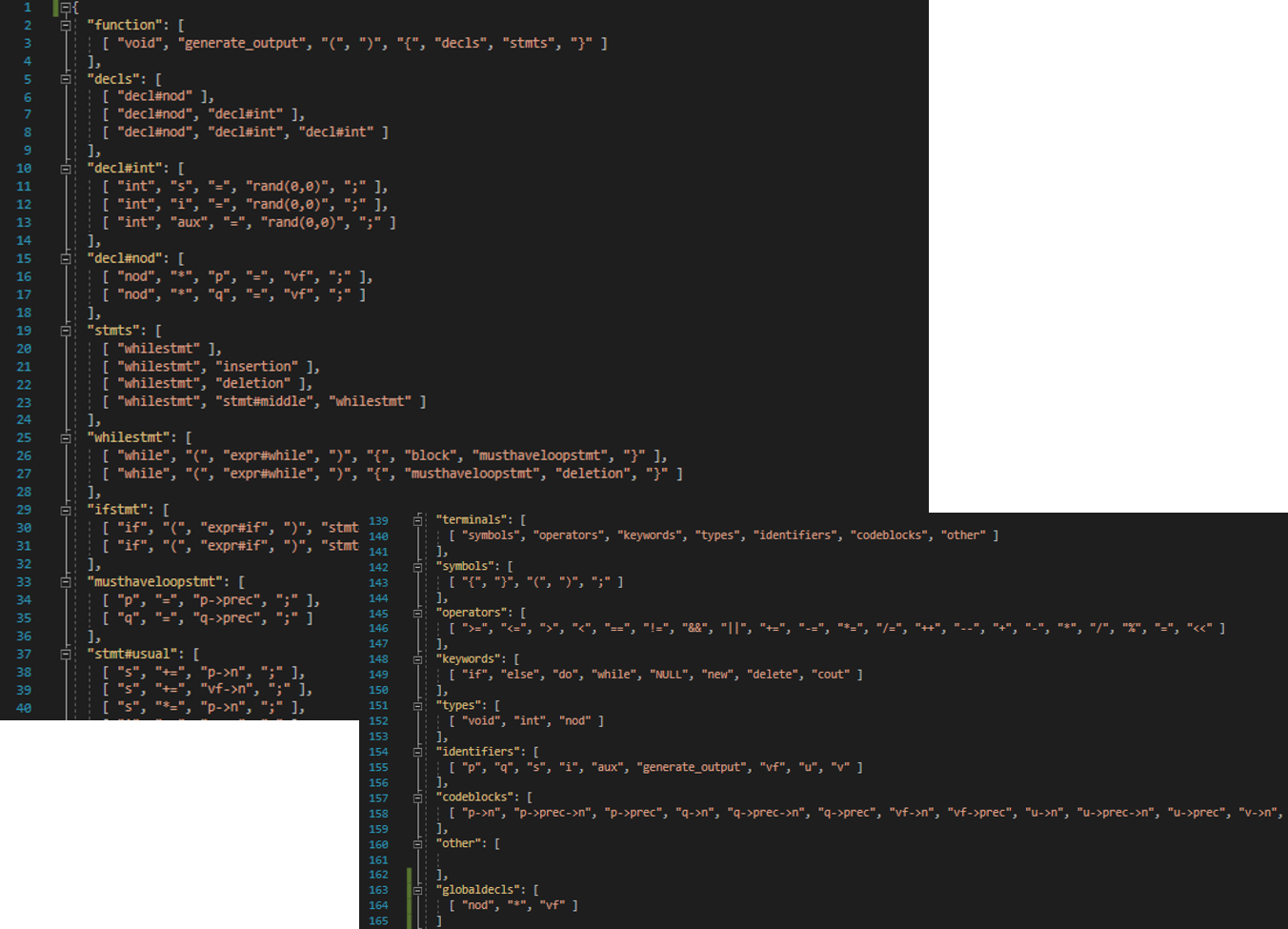
Prin utilizarea acestei abordări, pasul de generare a funcției produce membrul secvenței de cod al tripletului pe care l-am definit anterior. Ipoteza problemei și răspunsul corect sunt generate în etapa de procesare a output-ului, după compilarea și execuția șablonului. Pasul de generare a funcției folosește o gramatică cu ajutorul căreia se generează un arbore de sintaxă folosind o abordare top-down. Pe baza acestui arbore de sintaxă este construită o definiție terminală folosind o abordare de bottom-up.



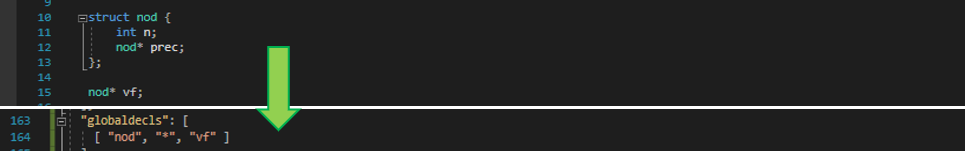
Generarea funcției se realizează pornind de la simbolul de start definit în fișierul de configurare și utilizând fișierul cu gramatica secvenței de cod. Mai întâi este creat un arbore de sintaxă, pe baza căruia este construită definiția terminală a acestuia. Definiția terminală este, la rândul ei, formatată pentru a putea fi ușor de citit și urmărit de către studenți, în conformitate cu convențiile implicite de scriere a codului utilizate de către utilitarul clangformat. Subsecvent formatării, se verifică dacă secvența de cod generată respectă regulile din fișierul de configurare.



Structură ierarhică a elementelor ce constituie secvența de cod care se dorește a fi generată este descrisă de o gramatică. Aceasta se mulează pe regulile precise ale limbajului de programare C++ care descriu structura sintactică a programelor corecte (un program este format din funcții, o funcție din declarații și instrucțiuni, o instrucțiune din expresii, etc.). Gramatica este specificată prin enumerarea regulilor de producție aferente simbolurilor. În cadrul proiectului, aceasta mai precizează terminalele (semne de punctuație, operatori, cuvinte cheie, tipuri de date, identificatori sau secvențe de cod) și variabilele globale utilizate în fișierul producător.



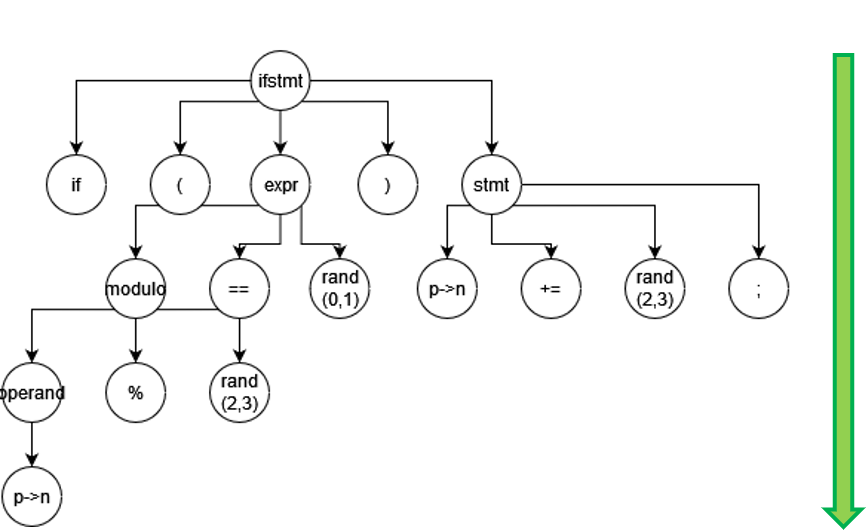
[poza cu gramatica partiala, si poza terminalele]

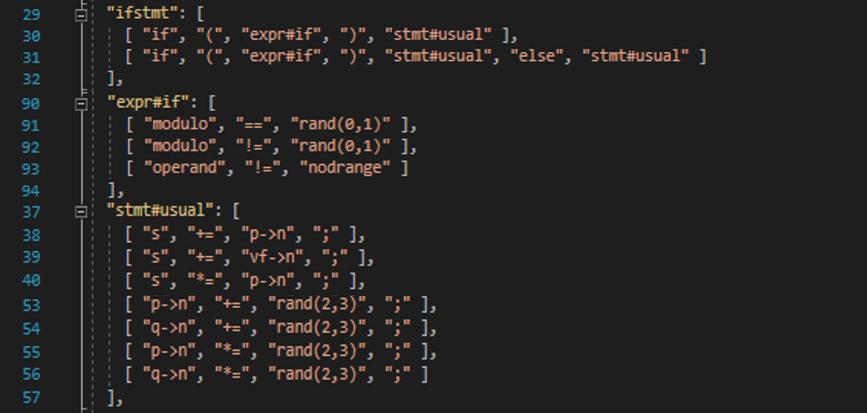
Crearea arborelui de sintaxă are în vedere generarea unui simbol, pe baza șirului de caractere definit ca simbol de start în fișierul de configurare și setarea acestuia drept rădăcină, precum și a structurilor necesare pentru obținerea unui cod calitativ, care să nu genereze erori de compilare (tabela de simboluri, cache-ul de instrucțiuni și stiva cu variabilele folosite în cadrul expresiilor). Premergător generării efective a arborelui, sunt inserate tipurile și identificatorii variabilelor globale din cadrul fișierelor producătoare, care trebuie menționate și în cadrul gramaticii.

[pozica cu variabilele din fișierele producătoare și din gramatici]

Fiecare nod asociat unui simbol conține o referință către părinte, un identificator (sirul de caractere al simbolului), o listă cu copiii si definiția terminală ce va fi evaluată după generare.

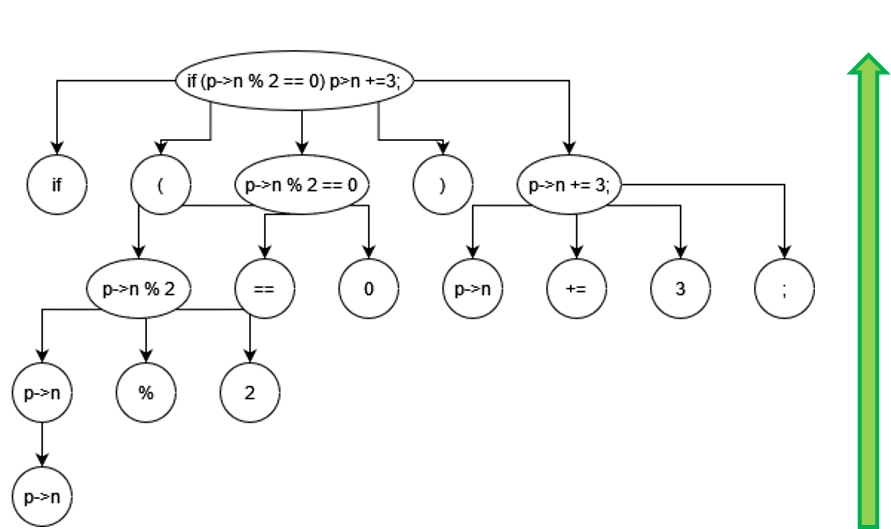
Generarea top-down a arborelui de sintaxă, se face pornind de la rădăcină (simbolul de start). Pentru un simbol este selectată aleator una dintre regulile de producție asociate și vor fi construiți copiii pentru simbolurile din corpul regulii de producție. De la stânga la dreapta, pentru fiecare copil este construit un subarbore. Un simbol nu mai este expandat dacă acesta este un terminal.





[figura top-down] [secventa din gramatica asociata]

Neterminalele și terminalele au asociate, implicit prin construcția lor, atribute sintetizate (identificatori și definiții terminale) precum și reguli semantice care descriu modul în care sunt întreprinse anumite acțiuni. Aceste reguli sunt executate doar la întâlnirea unor simboluri specifice. Valoarea unui atribut sintetizat a unui nod al arborelui este determinată de valorile atributelor copiilor săi. Atributele sintetizate sunt evaluate în timpul traversării bottom-up a arborelui de sintaxă, în momentul construcției definiției terminale.



[figura bottom-up]

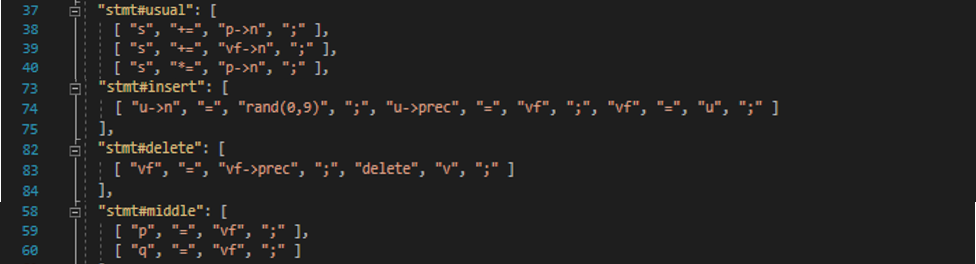
Dacă nodul evaluat este un terminal definiția sa terminală va fi însuși șirul de caractere din care este alcătuit. Dacă nodul este o macrodefiniție precum rand(X,X), definiția terminală va fi obținută prin intermediul unei funcții membru a generatorului, în acest caz returnând un număr aleator din intervalul precizat.

Dacă simbolul începe cu decl, va fi generată definiția sa terminală, verificându-se subsecvent dacă variabila declarată există deja în tabela de simboluri. Dacă generarea este reușită, tipul și identificatorul variabilei vor fi inserate în tabelă.



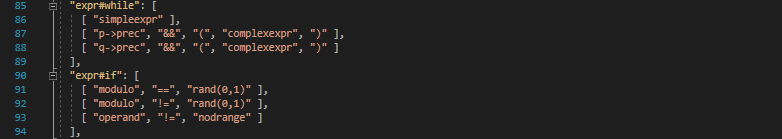
[figura cu gram]

Dacă simbolul începe cu stmt, va fi generată definiția sa terminală, verificându-se subsecvent dacă instrucțiunea există deja în cache-ul de instrucțiuni. Dacă nu există, vor fi obținute variabilele folosite în cadrul instrucțiunii și se va verifica dacă acestea se află în tabela de simboluri. Dacă se află în tabelă, acestea vor fi marcate ca fiind folosite. Cache-ul de instrucțiuni contine mereu ultimele 2 instrucțiuni generate.



[figura cu gram]

Dacă simbolul începe cu expr, va fi generată definiția sa terminală, verificându-se subsecvent dacă variabilele folosite există deja în tabela de simboluri. În cazul afirmativ, dacă părintele simbolului este o instructiune de tip bucla, acestea vor fi inserate în stiva cu variabilele folosite în expresii și sunt marcate ca fiind folosite în tabela de simboluri.



[figura cu gram]

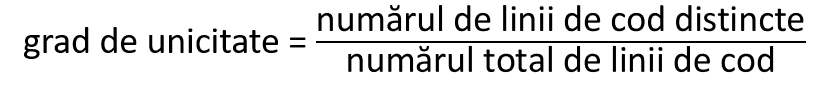
Pornind de la premiza că orice buclă are o instrucțiune care la un moment dat va cauza oprirea acesteia am definit simbolul musthaveloopstmt ca fiind instrucțiunea ce îndeplinește această caracteristică. Dacă simbolul începe cu must, va fi generată definiția sa terminală, verificându-se subsecvent dacă variabilele există în stiva cu variabilele folosite în expresii. Dacă există, va fi efectuată o operațiune de tip pop().



[figura cu gram]

În oricare dintre aceste cazuri, va fi semnalat dacă variabilele utilizate sunt nedeclarate. La final, sunt eliminate din arbore declarațiile de variabile care au fost nefolosite, verificând dacă în intrările asociate din tabela de simboluri au fost marcate ca fiind folosite.

În urma unei generări reușite, se verifică dacă numărul de linii de cod generate se încadrează între minimul și maximul stabilit în fișierul de configurare. Un utilizator neexperimentat poate crea o gramatică care să permită un grad mai ridicat de recursivitate, și prin urmare, să genereze secvențe de cod de lungimi mai mari decât cele pretabile pentru evaluare. De asemenea, mai este evaluat și gradul de unicitate calculat astfel:



[insert formula]

grad de unicitate = numărul de linii de cod distincte/numarul total de linii de cod

Dacă raportul nu depășește gradul de unicitate minim stabilit în fișierul de configurare, va trebui executată o regenerare a funcției.

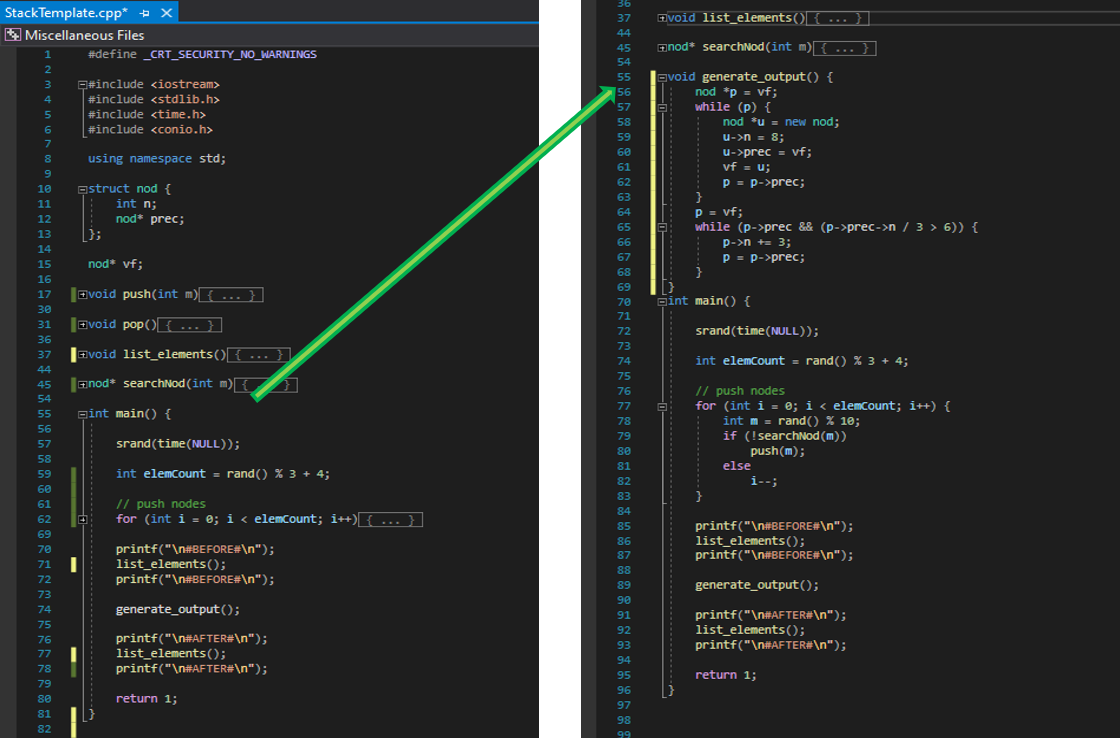
Implementarea structurilor și regulilor semantice menționate anterior, precum și verificările subsecvente au scopul de a genera un cod calitativ, care să nu se repete și să nu producă erori de compilare, scăzând astfel timpul de generare al unei întrebări. Dacă toate criteriile sunt respectate și generarea poate fi considerată reușită, definiția terminală este transmisă restului generatorului pentru procesare ulterioară.

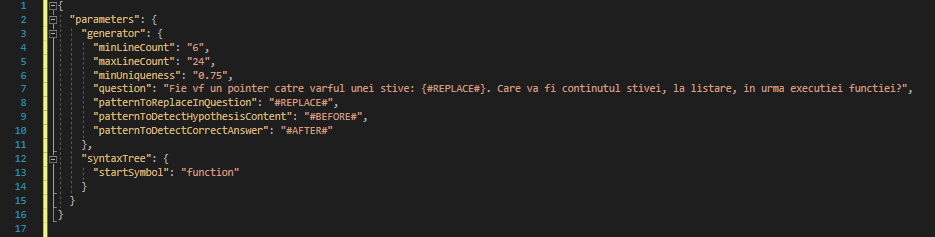
Funcția generată este inserată desupra metodei main() din tiparul desemnat, care va fi compilat și rulat. Fișierul producător (tiparul care va fi customizat cu funcția generată) trebuie:

* să conțină variabilele globale ce vor fi utilizate de funcția generată;
* să listeze detaliile ce vor fi inserate în ipoteza din fișierul de configurare;
* să execute funcția generată (inserată înaintea metodei main());
* să listeze răspunsul vizat de întrebarea care folosește șablonul respectiv.

Ambele listări sunt delimitate de tiparele definite în fișierul de configurare.

În cadrul proiectului, pentru întrebările ,, Fie vf un pointer către vârful unei stive: .... Care va fi conținutul stivei, la listare, în urma execuției funcției?’’ și ,, Fie prim un pointer către primul nod al următoarei liste: .... Care va fi conținutul listei, la listare, în urma execuției secvenței următoare?’’, variabilele globale sunt reprezentate de pointerii de tip nod către vârful unei stive, respectiv către primul și ultimul nod ai unei liste. Detaliile ce sunt inserate în ipoteză sunt obținute din listarea conținutului stivei, respectiv listei. Funcția generată alterează conținutul structurilor de date, iar la final prin listarea conținutului curent al structurilor este obținut răspunsul corect.





Dacă se produce o eroare de compilare sau de rulare a tiparului personalizat, pipeline-ul va abandona starea curentă și va reporni construcția.