INTRODUCERE ÎN PROGRAMARE.
SCHEME LOGICE ŞI LIMBAJ PSEUDOCOD.
FAMILIARIZAREA CU MEDIUL DE DEZVOLTARE CODEBLOCKS

1. Algoritmi, programe, programare

Algoritmul este un concept folosit pentru a desemna o mulțime finită de operații, complet ordonată în timp, care pornind de la date de intrare produce într-un timp finit date de ieșire. Cu alte cuvinte, algoritmul redă metoda de rezolvare a unei probleme într-un număr finit de paşi.

Programul este reprezentarea unui algoritm într-un limbaj de programare. Sunt cunoscute mai multe limbaje de programare, dezvoltate odată cu evoluția calculatoarelor. O anumită problemă poate fi mai uşor sau mai dificil de codificat într-un anumit limbaj de programare, întrucât multe din limbajele de programare de nivel înalt sunt orientate pe probleme.

Programarea este activitatea de elaborare a unui produs program. Ea are două ramuri importante:

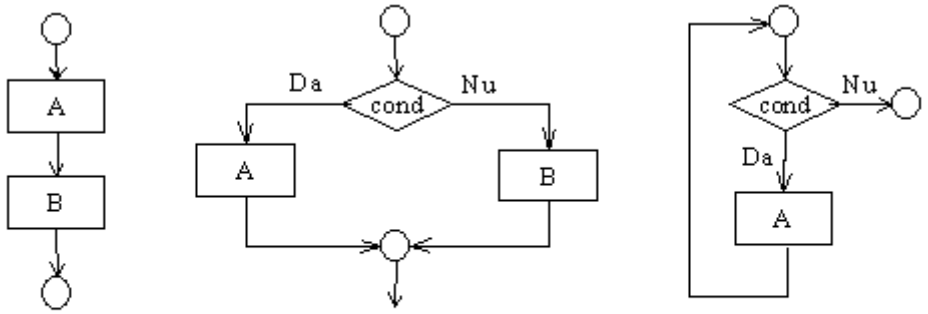
a) descrierea algoritmilor;
b) codificarea algoritmilor într-un anumit limbaj de programare.

Descrierea unui algoritm pentru rezolvarea unei probleme se poate face prin scheme logice sau într-un limbaj de descriere a algoritmilor, numit şi pseudocod.

2. Scheme logice și limbaj pseudocod

Schema logică este un mijloc de descriere a algoritmilor prin reprezentare grafică. Regulile de control ale algoritmului sunt descrise prin blocuri (figuri geometrice) reprezentând operațiile (pașii) algoritmului, iar ordinea lor de aplicare (succesiunea operațiilor) este indicată prin săgeți. Fiecărui tip de operație îi este consacrată o figură geometrică (un bloc tip) în interiorul căreia se va înscrie operația din pasul respectiv.

Orice algoritm poate fi descris într-o schemă logică folosind următoarele trei **structuri de control**:



(a) structura secventiala (b) structura alternativa (c) structura repetitiva

Prin execuţia unui algoritm descris printr-o schemă logică se înţelege efectuarea tuturor operaţiilor precizate prin blocurile schemei logice, în ordinea indicată de săgeţi.

În descrierea unui algoritm, deci şi într-o schemă logică, intervin variabile care marchează atât datele cunoscute iniţial, cât şi rezultatele dorite, precum şi alte rezultate intermediare necesare în rezolvarea problemei.

Limbajul **pseudocod** este un limbaj folosit în scopul proiectării algoritmilor şi este format din propoziţii asemănătoare propoziţiilor limbii române, care corespund structurilor de control folosite în construirea algoritmilor.

Prin execuţia unui algoritm descris în Pseudocod se înţelege efectuarea operaţiilor precizate de propoziţiile algoritmului, în ordinea citirii lor (de sus în jos şi de la stânga spre dreapta). Propoziţiile standard ale limbajului Pseudocod corespund structurilor de control prezentate anterior.

Structura secvenţială este redată prin concatenarea propoziţiilor, simple sau compuse, ale limbajului Pseudocod, care vor fi executate în ordinea întâlnirii lor în text.

Propoziţiile simple din limbajul Pseudocod sunt CITEŞTE, SCRIE, atribuire şi apelul de subprogram. Propoziţiile compuse corespund structurilor alternative şi repetitive.

Structura alternativă este redată în Pseudocod prin propoziţia DACĂ, prezentată în Pseudocod prin propoziţia CÂTTIMP.

3. Etapele rezolvării unei probleme şi stilul de programare

În general **etapele de rezolvare a unei probleme** sunt:

- analiza problemei, care constă în enunţul clar, precis al problemei de rezolvat, specificarea datelor de intrare şi ieşire
- proiectarea programului, care constă în stabilirea metodei de rezolvare şi întocmirea proiectului logic
- implementarea programului, care constă în codificarea într-un limbaj de programare, editarea fişierului sursă, compilarea, editarea de legături, execuţia şi testarea
- întocmirea documentaţiei
- întreţinerea programului

Respectarea acestor etape conduce la obţinerea unor performante legate de productivitatea programării şi de calitatea produsului program.

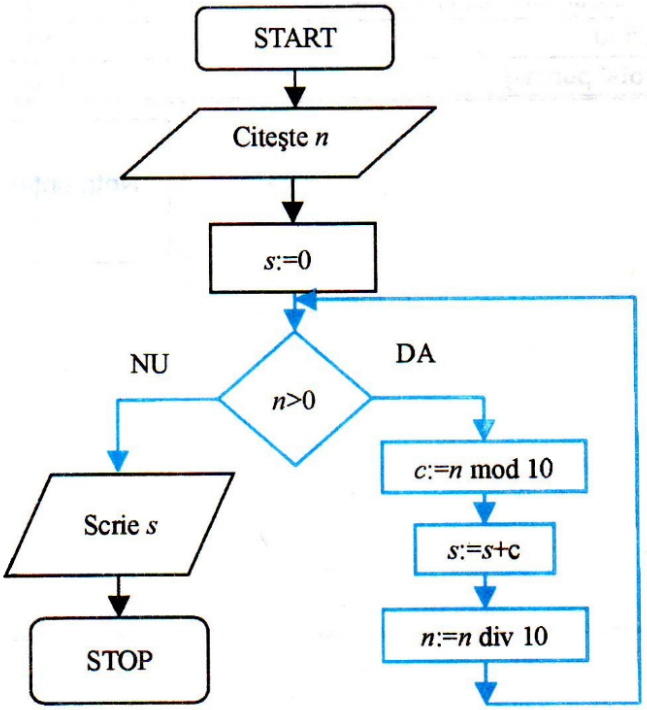
De asemenea se urmăreşte însuşirea unui **stil de programare**, caracterizat prin atribute care conferă unui program o anumită personalitate:

- evidenţierea structurii programului;
- modularizarea programului;
- abstractizarea datelor;
- claritatea programului;
- posibilitatea de modificare ulterioară cu efort mic;
- tratarea erorilor;
- generalitatea soluţiei.

4. Exemplu de rezolvare a unei probleme prin scheme logice și limbaj pseudocod

Enunțul problemei: Calculați și afișați suma cifrelor unui număr natural dat.

Schema logică:



Observație: Operatorii "mod"-*modulo* și "div"-*division* calculează restul respectiv câtul împărțirii unui număr întreg la alt număr întreg. Exemplu: 19 mod 6 = 1, 19 div 6 = 3.

Descrierea în pseudocod:

```
START
    CITEȘTE n
    s:=0
    CÂTTIMP n>0  execută
            c:=n mod 10
            s:=s+c
            n:=n div 10
    SFCÂT
    SCRIE s
STOP
```

5. Introducere în mediul de dezvoltare CodeBlocks

5.1 Primul program în C – afișarea textului „Hello World” pe ecran

```c
// primul program in C
#include <stdio.h>

int main()
{
  printf("Hello World!");
  return 0;
}
```

```
Hello World!
```

Structura programului de mai sus este următoarea:

- **linia 1:** `// primul program in C`
  - Cele două semne „ // ” indică un comentariu inserat de programator, care nu va avea efect asupra programului. Comentariul este valabil pe întreaga linie. Programatorii folosesc comentariile pentru a include scurte explicații sau observații în ceea ce privește codul ce urmează.
- **linia 2:** `#include <stdio.h>`
  - Liniile care încep cu semnul # sunt directive citite și interpretate de către preprocesor. În acest caz, directiva #include <stdio.h> permite operații cu funcții de intrare/ieșire din biblioteca „stdio.h”. Așadar afișarea pe ecran a textului „Hello World!” se realizează prin intermediul funcției de ieșire „printf”.
- **linia 3:** Linia liberă
  - Liniile libere nu au niciun efect asupra programului.
- **linia 4:** `int main()`
  - Aceasta linie reprezintă antetul unei funcții. În esență, o funcție conține un grup de instrucțiuni. Fiecare funcție primește un nume, aici numele fiind „main”. Detalii teoretice despre funcții vor fi prezentate într-un laborator ulterior. Antetul unei funcții are următoarele elemente: tipul returnat („int”), nume („main”) și lista de argumente între două paranteze („( )”). Funcția main este o funcție specială în toate programele C, fiindcă reprezintă funcția care este chemată atunci când pornește execuția programului. Execuția tuturor programelor în C încep de la funcția main.
- **liniile 5 & 8:** { și }
  - paranteza „{” reprezintă începutul unei funcții, iar paranteza „}” reprezintă sfârșitul unei funcții. Între aceste două paranteze se află corpul funcției format din secvența de instrucțiuni respectivă.

- **linia 6:** `printf("Hello World!");`
    - Linia reprezintă o instrucțiune C ce constă în apelul funcției „printf” folosită pentru afișarea textului „Hello world!”. După fiecare instrucțiune se inserează semnul punct-virgula „ ; ” cu rol de separare a instrucțiunilor.

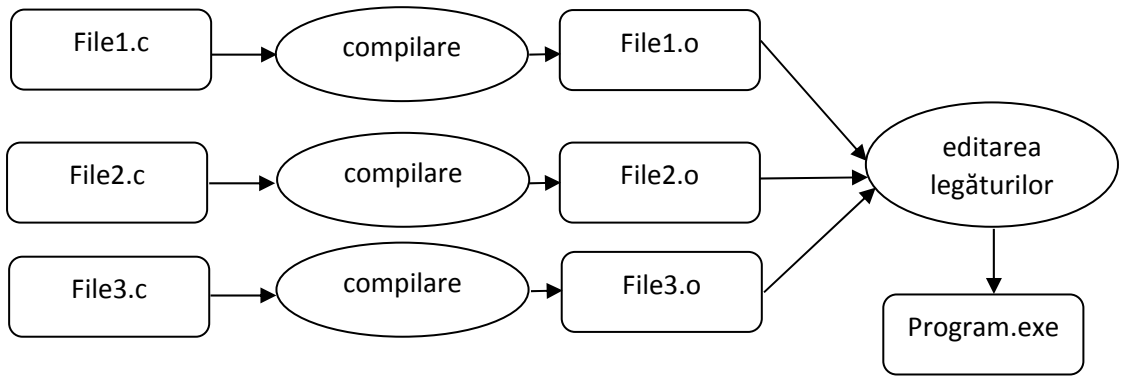
- **linia 7:** `return 0;`
    - Linia reprezintă valoarea returnată de către funcția main. Doar pentru funcția main, valoarea returnată semnifică un cod de eroare. În cazul de față, valoarea 0 semnifică terminare cu succes. Orice altă valoare diferită de 0 reprezintă un cod de eroare cu care se termină execuția programului.

5.2 Folosirea mediului de dezvoltare CodeBlocks

Codul unui program software scris într-un limbaj de programare precum C/C++ se numește „cod sursă”.

Fișierele unui program C

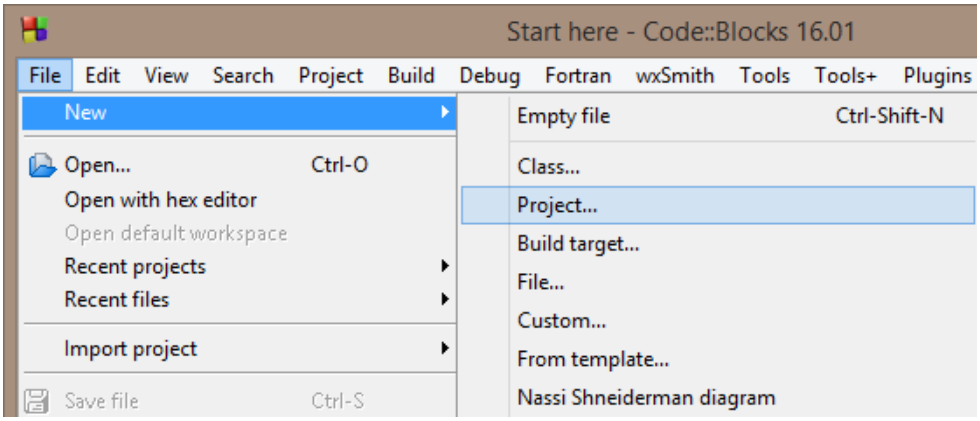
1. Fișiere **sursă**. Aceste fișiere conțin instrucțiuni C, iar extensia lor este „.c”. Ex: „main.c”
2. Fișiere **header.** Aceste fișiere conțin în principiu declarații de funcții. Acestea sunt folosite de către codul sursă pentru a accesa funcții externe. Aceste fișiere header sunt folosite de către preprocesor, care transformă codul sursă întru-un cod sursă extins. Fișierele header au extensia „.h”. Ex: „stdio.h”
3. Fișiere **obiect.** Codul sursă scris într-un limbaj înțeles doar de către programator este transformat în cod **obiect** de către compilator**.** Codul obiect (codul mașină) are formă binară și este stocat în fișiere cu extensia „.o”.
4. Fișiere **executabile.** Acestea reprezintă produsul unui program numit „linker”. Linker-ul editează legăturile din mai multe fișiere obiect și produce un fișier binar care poate fi executat, numit **executabil.** Extensia acestor tipuri de fișiere este „.exe” în sistemul de operare Windows.



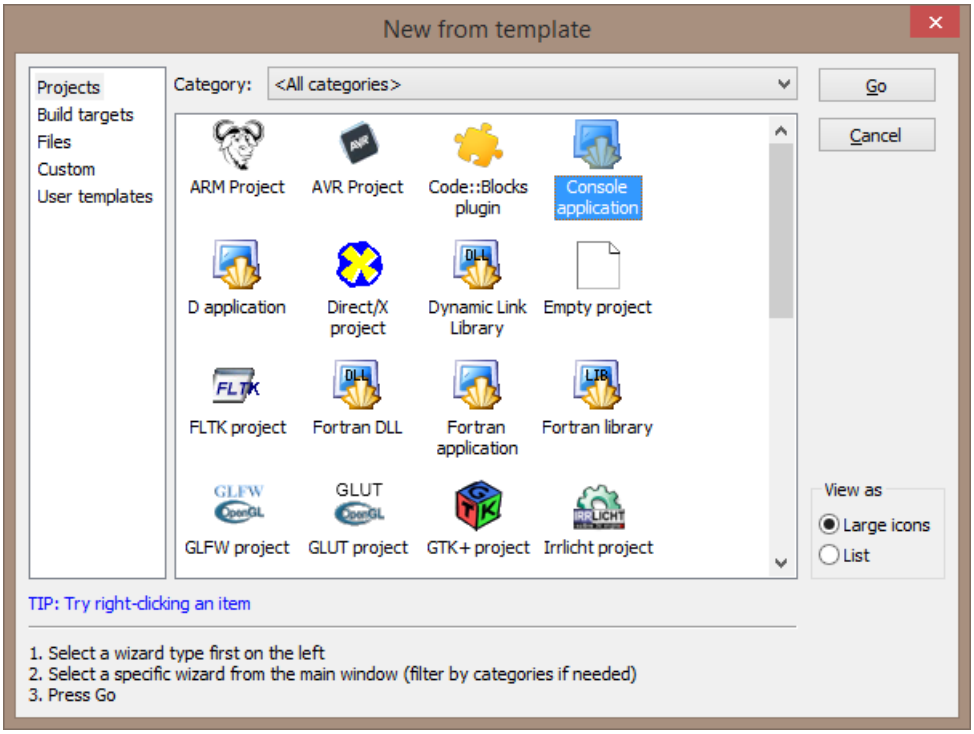
Primul proiect în CodeBlocks

Un proiect este o colecție de unul sau mai multe fișiere sursă (ex. File1.c, File2.c, etc.) și fișiere header (ex. File1.h, File2.h, etc.).

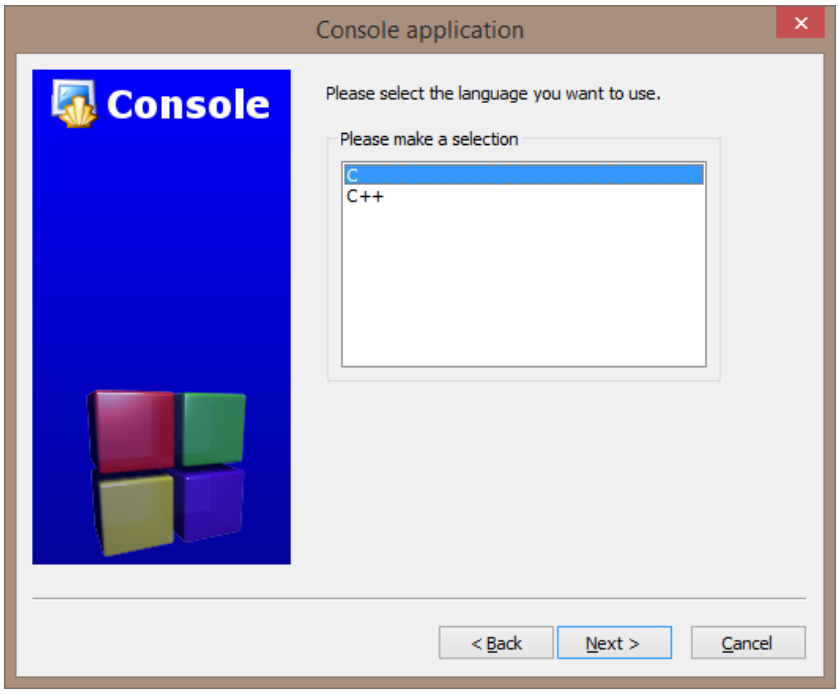
1. Pentru a crea un proiect, se va alege din meniu *File->New->Project.*



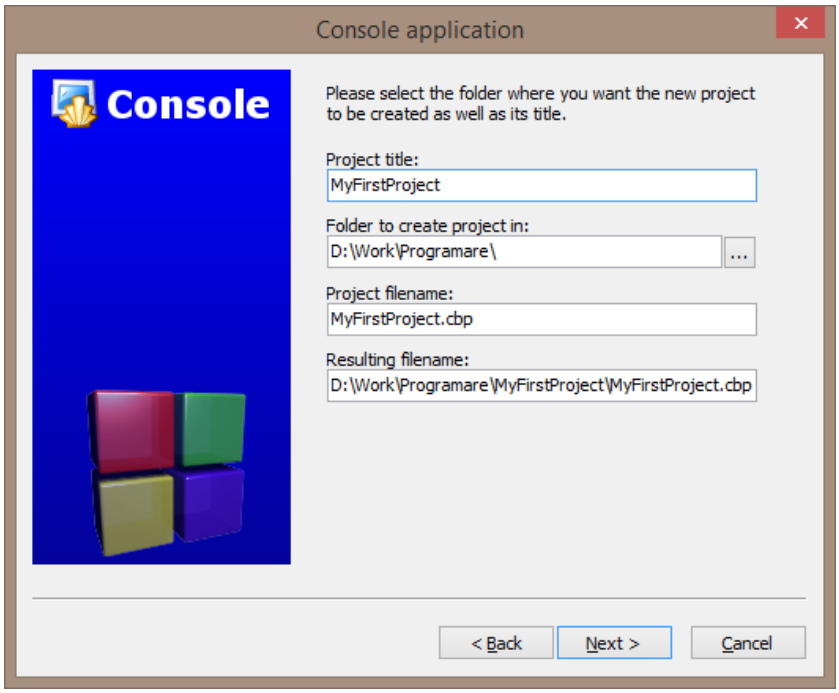
2. Se va deschide o fereastra nouă *New from template.* Se va alege *Console Application*, care va permite unui program să citească și să afișeze date la consolă. Apoi, se apasă butonul *Go*.



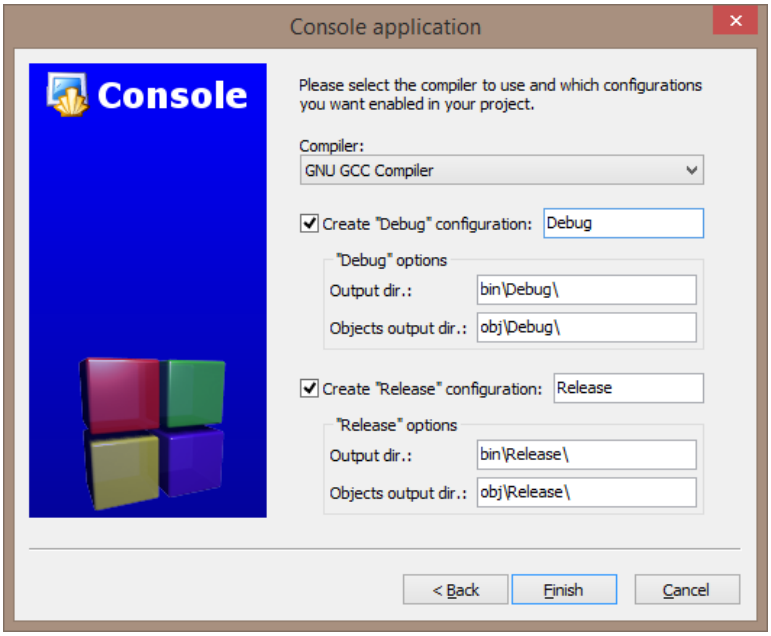
3. Următorul pas este alegerea limbajului de programare C. Se va da *Next.*



4. În următoarea fereastră se va scrie numele proiectului sub *Project Title*, și directorul unde se va salva proiectul sub *Folder to create project in*. În acest director va fi salvat proiectul cu extensia Code Block Project (.cbp).

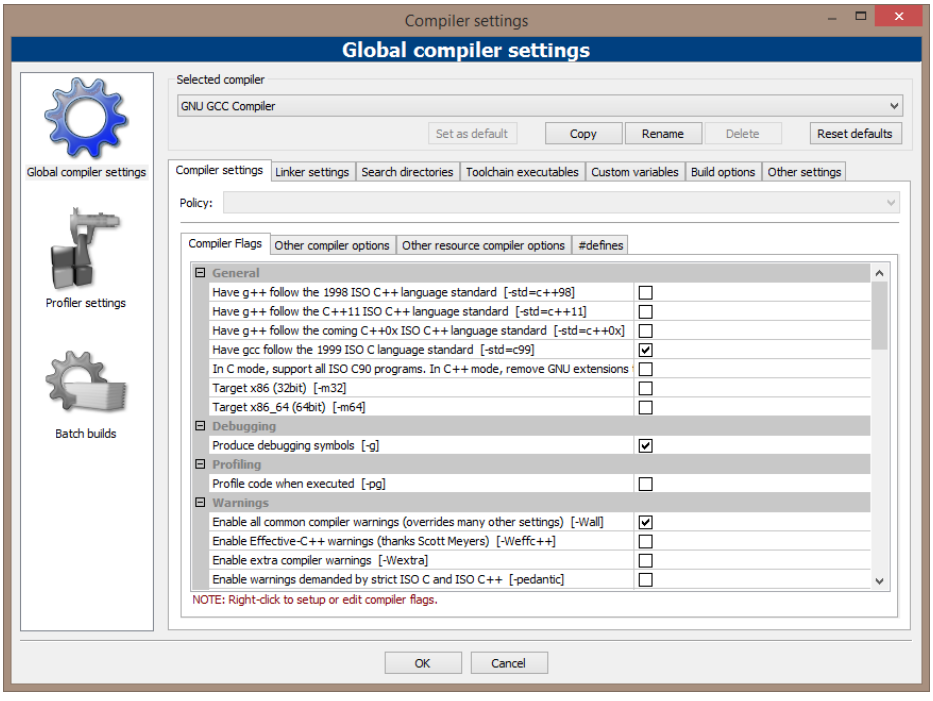


5. Următoarea fereastră intitulată *Console Application* permite alegerea compilatorului. Se recomandă folosirea setărilor implicite. Se vor bifa *Create „Debug" configuration* și *Create „Release" configuration.*



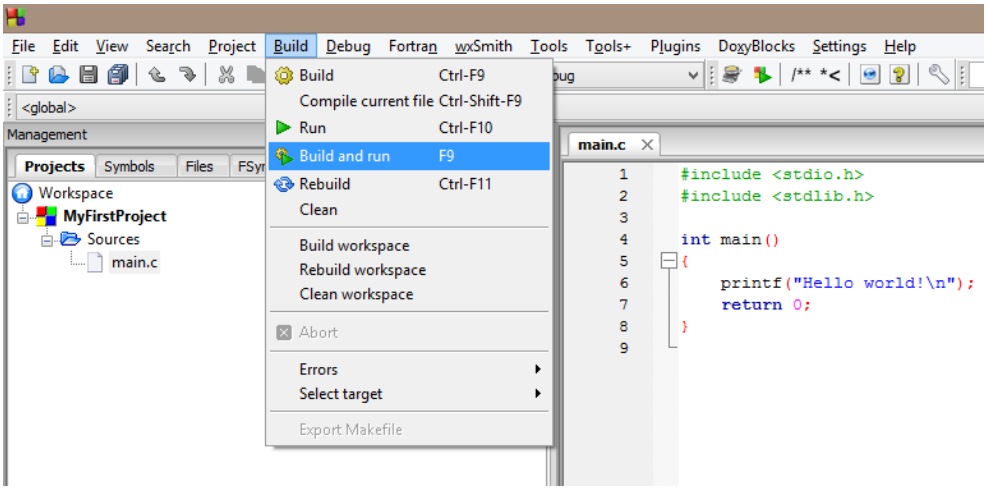
Configurațiile Release și Debug

Debugger-ul facilitează oprirea temporară a execuției unui program. În modul **Debug** se poate inspecta codul, starea variabilelor și se pot găsi eventualele erori în program. În configurația Debug, programul este compilat cu informație suplimentară: informația de debug. Programul nu este optimizat iar rularea acestuia în aceasta configurație mărește timpul de execuție. Pentru ca informația de debug să fie injectată trebuie să bifați „*Produce debugging symbols*" din meniul *Project->Build options*. De asemenea, dacă se dorește compilarea programului în standardul C99 și afișarea tuturor mesajelor de avertizare (*warnings*) se vor bifa căsuțele corespunzătoare. Dacă doriți ca aceste lucruri să fie realizate pentru toate proiectele, puteți să faceți setarea direct asupra compilatorului din meniul *Settings->Compiler*.



În configurația **Release** codul mașină generat nu conține informația de debug, iar programul este optimizat. Rularea programului în modul Release oferă performanțe maxime.

Procesul de rulare al programului



Compile current file

- compilarea codului sursă în cod obiect

Build

- compilare + editarea legăturilor: cod sursă -> cod obiect -> executabil. Acest proces va avea loc doar dacă s-au făcut modificări în fișiere de la ultimul build.

Build target

- permite alegerea configurației Debug/Release

Run

- rularea fișierului executabil

Clean

- ștergerea tuturor artefactelor create de *Build (*fișiere obiect, fișiere executabile)

Rebuild

- clean + compilare + editarea legăturilor. Acest proces va avea loc indiferent daca s-au făcut modificări în fișiere sau nu.

Descărcarea mediului de dezvoltare CodeBlocks și a compilatorului GCC precum și alte informații despre folosirea acestuia se pot găsi la: http://www.codeblocks.org

6. Mersul lucrării

6.1. Se vor înțelege conceptele introduse și exemplele prezentate.

6.2. Creați primul vostru proiect *HelloWorld* în limbajul C utilizând mediul de dezvoltare CodeBlocks.

6.3. Realizați schema logică și descrierea în limbaj pseudocod pentru calculul și afișarea valorii lui *n factorial* (*n*!). Valoarea lui *n* este citită și trebuie să reprezinte un număr natural mai mic decât 100, în caz contrar se va afișa un mesaj de eroare.

6.4. Realizați schema logică și descrierea în limbaj pseudocod pentru calculul și afișarea rădăcinilor reale/complexe ale ecuației de gradul doi $aX^2+bX+c=0$. Valorile coeficienților *a*, *b*, *c* se citesc și sunt numere reale.

6.5. Realizați schema logică și descrierea în limbaj pseudocod pentru afișarea descompunerii unui număr natural nenul *n* în factori primi. Dacă *n* este mai mic sau egal cu 0 se va afișa un mesaj de eroare. (Exemplu de rezultat în cazul *n*=4525400: 2^3 5^2 11^3 17^1)

6.6. Realizați schema logică și descrierea în limbaj pseudocod pentru afișarea primelor *n* numere prime. (Exemplu de rezultat în cazul *n*=10: 2 3 5 7 11 13 17 19 23 29)