

### Tema: Cunoștință cu interfața grafică Dev C++. Scrierea programelor simple.

#### Obiective:

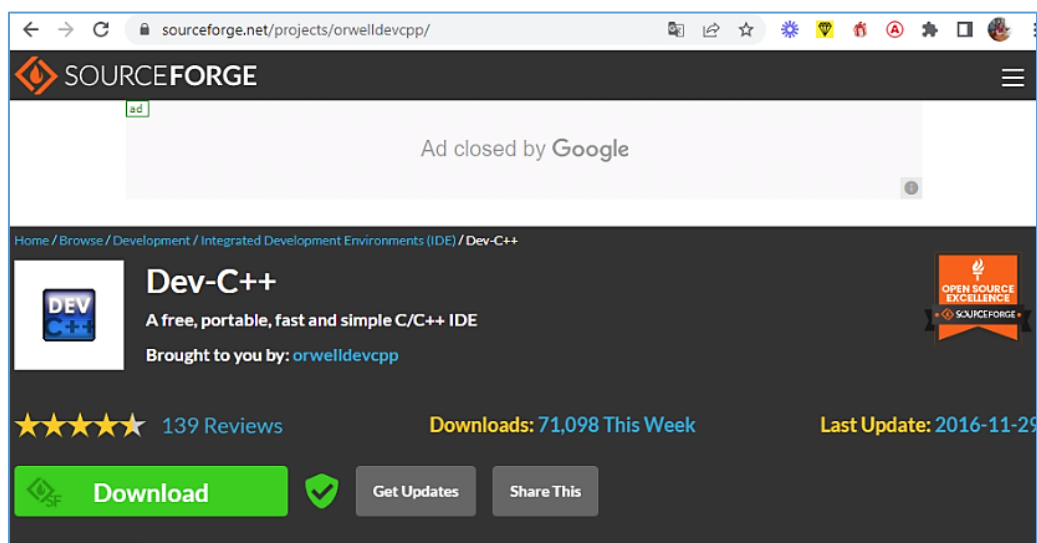
- să realizeze instalarea unui mediu de programare;
- să recunoască structura unui program;
- să descrie esența proceselor de compilare și interpretare;
- să interpreteze corect mesajele de eroare.

#### 1.1. Instalarea mediului de programare DEV C++

Mediul de dezvoltare integrat Dev-C++ este un program gratuit și destinat programatorilor pentru a dezvolta software în limbajului de programare C/C++. Cu toate că este gratuit, Dev-C++ are un număr de caracteristici care îl fac extrem de atractiv pentru utilizare. Fără a intra în detalii, vom remarca în mod special că interfața grafică este foarte flexibilă, dar intuitivă și ușor de folosit, include toate facilitările necesare unui mediu de programare performant: editor, biblioteca funcțiilor, compilator și executor, depanator, crearea proiectelor, adăugare de biblioteci etc.).


În cadrul orelor de laborator se folosește mediul Dev-C++ pentru programare în limbajul C/C++, iar compilarea codului de program va avea ca rezultat obținerea de fișiere executabile Win32.

Mediul de programare DEV-C++ poate fi descărcat gratuit de pe adresa <https://sourceforge.net/projects/orwelldevcpp/>



**Figura 1.1.** Descărcarea mediului de programare DEV-C++

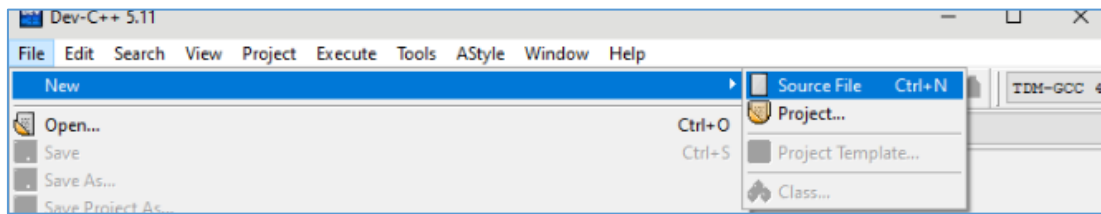
Continuați instalarea programului cu lansarea fișierului

 Dev-Cpp 5.11 TD....exe

#### 1.2. Crearea, redactarea și salvarea programului

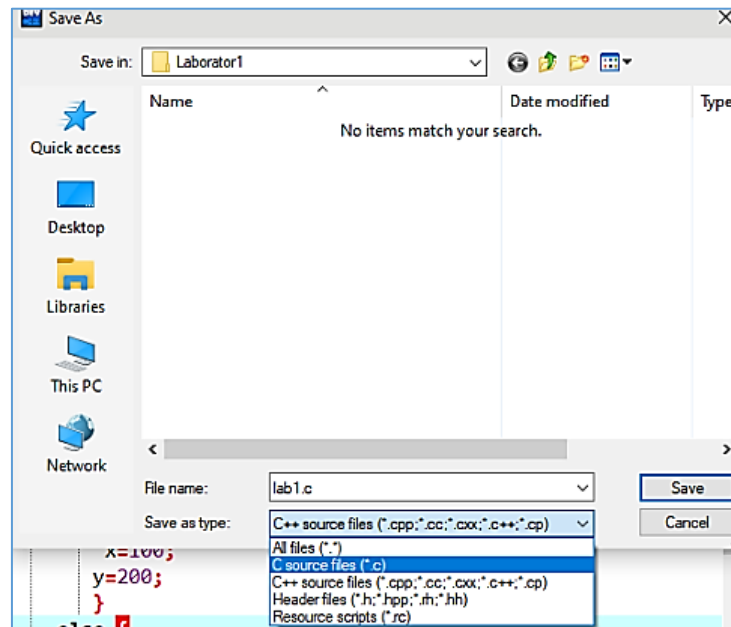
Fișierului sursă al programului, în DEV-C++, poate fi creat în două moduri:

- Crearea unui singur fișier sursă (File / New / Source File, a se vedea Figura 1.2).



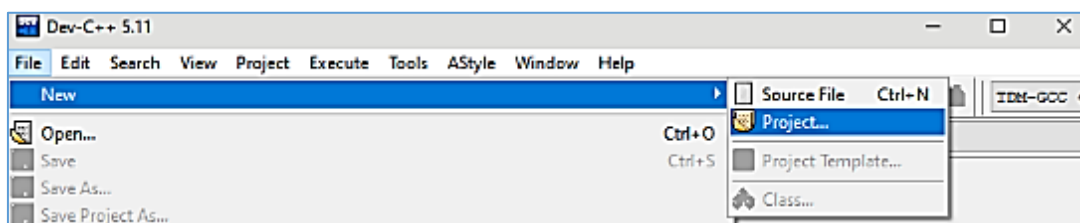
**Figura 1.2.** Crearea unui fișier sursă

Fișierul sursă trebuie salvat (File / Save As...) pe disc se atribuie un nume distinct de exemplu *lab1*. Fișierele salvate ca fișiere C trebuie salvate cu extensia *.c* (a se vedea Figura 1.3).



**Figura 1.3.** Salvarea fișierului sursă cu extensia *.c* în mapa *Laborator1*

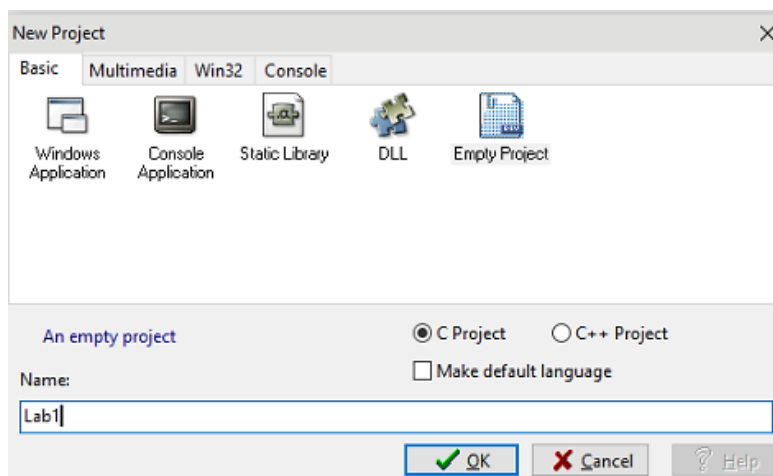
- II. O altă modalitate eficientă, utilizată în special atunci când avem mai multe fișiere sursă, este crearea unui Proiect ( File / New / Project..., a se vedea Figura 1.4). Proiectul permite înglobarea diferitor fișiere și editarea acestora cu *editorul de legături* după compilare, controlul parametrilor disponibili în mediul de programare, dar și aplicarea depanatorului.



**Figura 1.4.** Crearea unui proiect

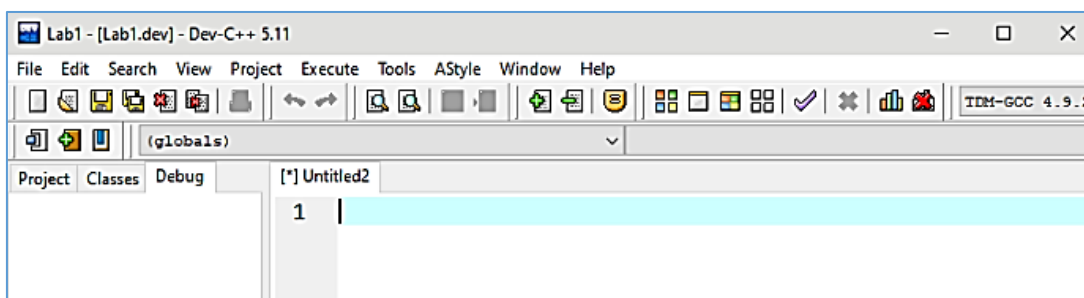
Crearea proiectului continuă cu selectarea opțiunii *Empty Project* (a se vedea Figura 1.5), stipularea denumirii proiectului (de exemplu *Lab1*) și selectarea extensiei fișierului: *C Project* – cod

scris în limbajul C. În continuare se deschide fereastra de lucru *Save As*, se creează o mapă nouă *Laborator1* în care este salvat proiectul *Lab1* cu extensia *.dev*.



**Figura 1.5.** Crearea unui proiect gol

Ca urmare, interfața grafică include proiectul *Lab1* cu un singur fișier numit implicit *Untitled* în care se va scrie codul de program (a se vedea Figura 1.6).



**Figura 1.6.** Fereastra de lucru DEV-C++

Exemplul 1: În această fereastră de lucru se culege textul programului prezentat în Figura 1.7.

```

[*] Untitled2
1  #include <stdio.h>
2  int main(void) {
3      int a, b, c, d, x, y;
4      a=1;
5      b=2;
6      c=a+b;
7      d=a*b;
8      if (c==d) {
9          x=100;
10         y=200;
11     }
12     else {
13         x=200;
14         y=100;
15     }
16     printf("%d %d\n",x,y);
17     return 0;
18 }
19

```

**Figura 1.7.** Codul de program scris în limbajul C

La culegerea și redactarea ulterioară a textului, se pot utiliza tastele de bază și combinații de taste: *Home*, *End*, *Ctrl<-*, *Ctrl->*, *Ctrl+Y*, *Ctrl+Z* (anulează ultima acțiune), *Ctrl+C* (copie blocul selectat), *CTRL+X* (permută blocul), *CTRL+V* (inserează blocul copiat) ș.a.

### 1.3. Compilarea și executarea programului

Fișierul scris cu extensia *.c* este un fișier text și include instrucțiuni înțelese de programator. Procesul de conversie a fișierului sursă C în limbajul mașinii este cunoscut sub numele de *compilare* și include două componente: *analiza și sinteza*.

*Componenta de analiză studiază fișierul sursă din punct de vedere:*

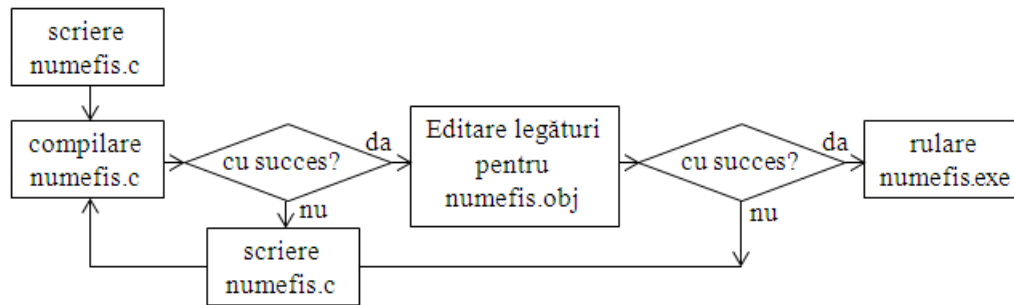
- lexical – textul sursă se descompune în cuvinte individuale numite atomi (identificatori, cuvinte-cheie, operatori, ... ) și se verifică corectitudinea lexicală a programului;
- sintactic – șirul de atomi lexicali este analizat pentru a depista structuri sintactice, cum ar fi expresii, instrucțiuni, declarații etc. și se verifică corectitudinea sintactică a programului;
- semantic – se validează expresiile limbajului sintactic corecte, luând în considerație semnificația lor. De exemplu, expresia “unu”+2 sintactic este corectă, dar nu are sens. În funcție de limbajul utilizat, fie se semnalează o eroare datorată incompatibilității dintre tipuri, fie 2 este transformat implicit în șir de caractere.

*Componenta de sinteză efectuează:*

- optimizarea codului, astfel încât programul rezultat să fie cât mai performant. Presupune eliminarea variabilelor inutile. De exemplu, se exclude codul care apare după instrucțiunea *return*, întreaga expresie *if* este înlocuită doar cu una din ramurile ei, expresiile constante sunt înlocuite cu valori etc.
- generarea codului final, transformă instrucțiunile înscrise în instrucțiuni mașină. *Dacă sunt depistate erori, compilatorul afișează mesaje* cu descrierea erorilor, în așa caz, se revine la fișierul sursă și se redactează greșelile, apoi fișierul sursă se salvează și se recompilează din nou. *Dacă nu sunt depistate erori*, se creează fișierul obiect cu extensia **.OBJ**.

La examinarea conținutului directorului, după compilarea cu succes a programului, se pot găsi

unul sau mai multe fișiere cu extensia **.OBJ**.

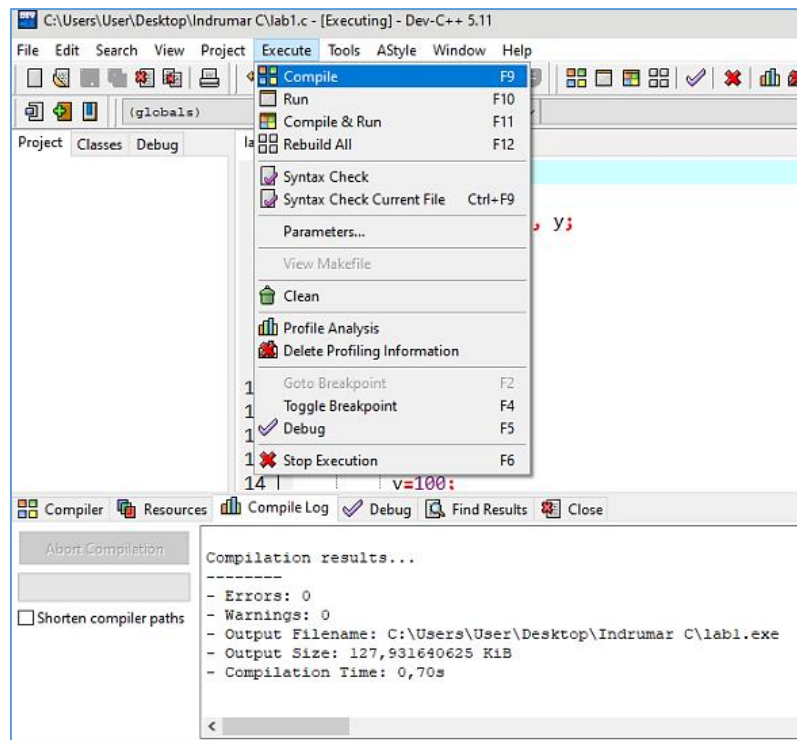


**Figura 1.8.** Etapele compilării codului de program

Fișierul obiect nu poate fi executat, deoarece conținutul lui nu este complet. Fișierul executabil este format de un program denumit *editorul de legături*, care combină instrucțiunile din fișierul obiect cu funcțiile pe care le deține compilatorul (cum ar fi *printf*, *scanf*, *pow* s.a.). Figura 1.8 ilustrează procesul de creare a fișierului executabil.

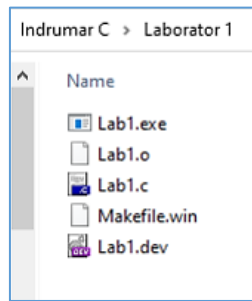
La fel ca în cazul procesului de compilare, pe parcursul căruia se pot detecta erori, procesul de editare a legăturilor poate să identifice erori.

Exemplul 2: Codul de program prezentat în Figura 1.7 se compilează *Execute / Compile* (a se vedea Figura 1.9).



**Figura 1.9.** Compilarea cu succes a codului de program

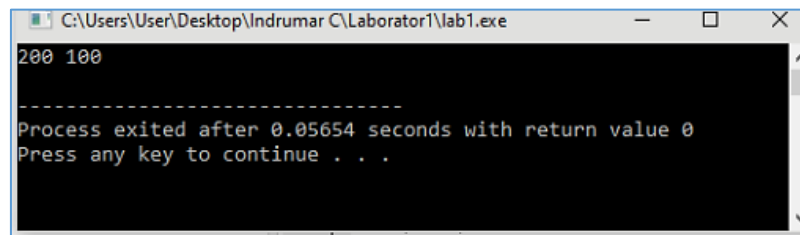
Ca urmare a acțiunilor întreprinse, este solicitată denumirea fișierului (în cazul nostru *Lab1*), care este salvat în mapa *Laborator1* creată anterior. La deschiderea mapei *Laborator1* sunt identificate mai multe fișiere care au fost create la etapa compilării codului de program (a se vedea Figura 1.10).



**Figura 1.10.** Setul de fișiere create la etapa compilării codului de program

Pe ecran va apărea mesajul *Compilation results...*, în care se reflectă mersul compilării. La finisarea normală a compilării în partea de jos a interfeței grafice, mesajul prezentat trebuie să includă *Errors:0, Warnings: 0* (a se vedea Figura 1.9).

După compilarea programului fără erori, codul de program este lansat (*Execute / Run (F10)*). Pe ecran apare rezultatul execuției codului de program (a se vedea Figura 1.11).



**Figura 1.11.** Rezultatul obținut la rularea programului *Lab1*

Dacă programul nu necesită introducerea datelor (astfel este programul prezentat mai sus), nici nu se observă, cum acest program se realizează. Pe ecranul negru va fi afișat rezultatul (în cazul nostru aceasta trebuie să fie: *200 100*). Pentru a ieși din regimul de rezultat, se tastează oricare tastă.

#### 1.4. Diagnosticarea erorilor și avertizările emise de compilator și editorul de legături

La această etapă de îndeplinire a lucrării de laborator, propunem să se experimenteze cu mesajele compilatorului și editorului de legături.

Exemplul 3: În prealabil, recomandăm să se facă o copie a fișierului-sursă. Pentru aceasta se accesează *File / Save as...*, se introduce un nume nou, de exemplu, *lab01x.c*. Acum sunt două copii a programului în două fișiere (*lab1.c* și *lab01x.c*). În fișierul *lab1.c* se păstrează varianta corectă a programului, iar în *lab01x.c* se vor introduce erori.

Se deschide fișierul *lab01x.c*. și se introduce următoarele modificări în textul programului (acum și mai departe va fi indicat numărul rândului în care se introduc modificări):

```
6  c=a+b1;
9  x=100; a123
18 /* } */
```

Se continuă cu compilarea programului (*F9*). Drept urmare, în partea de jos, se obțin mesaje de eroare ( a se vedea Figura 1.12):

Compiler (6) Resources Compile Log Debug Find Results Close			
Line	Col	File	Message
		C:\Users\User\Desktop\Indrumar C\Laborator 1\Lab0...	In function 'main':
6	12	C:\Users\User\Desktop\Indrumar C\Laborator 1\Lab01x.c	[Error] 'b1' undeclared (first use in this function)
6	12	C:\Users\User\Desktop\Indrumar C\Laborator 1\Lab01x.c	[Note] each undeclared identifier is reported only once for each f...
9	18	C:\Users\User\Desktop\Indrumar C\Laborator 1\Lab01x.c	[Error] unknown type name 'a123'
17	7	C:\Users\User\Desktop\Indrumar C\Laborator 1\Lab01x.c	[Error] expected declaration or statement at end of input
28		C:\Users\User\Desktop\Indrumar C\Laborator 1\Makefil...	recipe for target 'Lab01x.o' failed

**Figura 1.12.** Mesaje de eroare

Primul rând – antet. În următoarele rânduri: numărul rândului, coloana, numele fișierului și diagnosticarea erorii.

Mesajul pentru rândul 6 – „*Simbol necunoscut 'b1'*” (în expresie se utilizează variabila *b1* care nu este declarată).

Nota informativă pentru rândul 6 – fiecare identificator nedeclarat este raportat o singură dată pentru fiecare funcție în care apare.

Mesaj pentru rândul 9 – „*Simbol nedeterminat 'a123'*” (textul *a123* seamănă cu numele unei variabile care nu este declarată).

Mesajul pentru rândul 17 – „*lipsește declarația de sfârșit*” – lipsa acoladelor vor fi depistate numai la ultimul operator al programului.

Selectarea unei erori prin dublu click duce la plasarea cursorului în textul programului pe rândul indicat în mesajul corespunzător.

## 1.5. Subiecte de evaluare

### Exerciții:

1. Să se implementeze programele cu exemplele prezentate.
2. Să se scrie/salveze un program care afișează următorul text:

*EXEMPLU de program, Nr.1*

*Disciplina: Limbaje de programare*

*Autor: studentul ....*

3. Să se scrie/salveze un program care să afișeze un caracter, un șir de caractere, o valoare numerică întreagă și o valoare numerică reală.

### Verificarea cunoștințelor:

1. Relatați despre procedura de instalare / lansare a mediului de programare DEV-C++.
2. Enumerați componentele principale ale unui program scris în limbajul C.
3. Identificați tastele principale și combinațiile de taste utilizate la redactarea unui text.
4. Descrieți modalitățile de salvare și salvare repetată a unui program.
5. Relatați despre compilarea și executarea unui program.
6. Identificați mesajele emise de compilator.
7. Identificați fișierele obținute la executarea codului de program. Explicați destinația acestor fișiere.
8. Identificați numele funcției cu care se începe execuția programului C.

9. Relatați despre componentele procesului de compilare: analiză și sinteză.
10. Enumerați fișierele ce se formează la compilarea programelor C.
11. Descrieți destinația editorului de legături.
12. Identificați directiva utilizată pentru includerea fișierelor antet.
13. Identificați semnificația parantezelor unghiulare < > care încadrează numele unui fișier antet.

## 1.6. Bibliografie

1. Herbert Schildt, *C++ manual complet*, Editura Teora, București, 1999.

Disponibil: <https://drive.google.com/file/d/1BrQtITgykcWk03xxtl3Q0-nvcRRFAsy7/view?usp=sharing>

3. Dev-C++ Official Website. Disponibil: <https://www.bloodshed.net/>

4. How to Install Dev-C++ and the GLUT Libraries for Compiling OpenGL Programs with ANSI C.

Disponibil: <https://chortle.ccsu.edu/bloodshed/howtogl.html>

5. Allan Spale. Using Bloodshed Dev-C++ for OpenGL-GLUT Programming. Disponibil

<https://www.ev1.uic.edu/aspale/dvl/dev-cpp/>