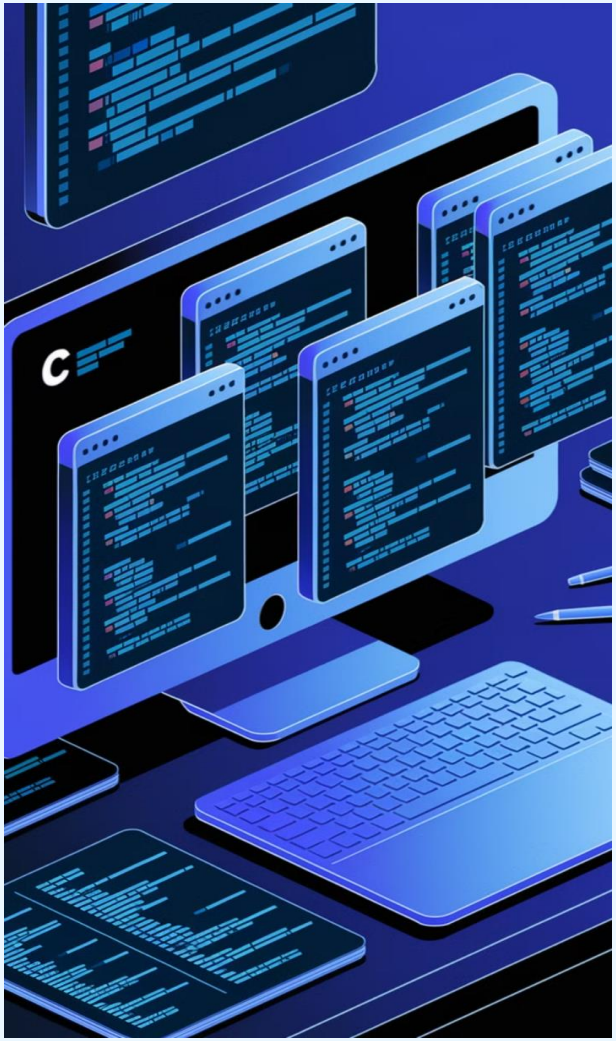
Модулі. Конструювання багатофайлових (мульти…) програм.

Модульне програмування в C/C++ - основа розробки сучасного програмного забезпечення. Воно відкриває шлях до гнучкої та масштабованої архітектури.



Поняття модульності

Програми, що вирішують реальні практичні задачі, мають великий обсяг та складаються з тисяч функцій.

Записувати всі функції програми в один файл незручно, оскільки зі збільшенням програми в ньому стає важко орієнтуватися, а також стає неможливою спільна робота кількох програмістів.

Технологія модульного програмування полягає у розбитті великої задачі на частини, з наступною реалізацією кожної підзадачі в окремому файлі, який називають **модуль**.

Поняття модульності

Модуль - незалежні компоненти з чітко визначеними інтерфейсами. Вони дозволяють розділити логіку на функціональні блоки.

Під модулем у програмуванні розуміється згруповані за своїм функціональним призначенням сукупність констант, типів даних, функцій, що обробляють ці дані, та незалежні від основної програми, які компілюються окремо та допускають повторне використання.

Модуль містить дані та функції для їх обробки. Для того щоб його використовувати достатньо знати інтерфейс, а не всі деталі реалізації. Приховування деталей реалізації називається **інкапсуляцією.**

Базові поняття модульного програмування

Переваги багатофайлової архітектури:

- ✓ спрощення розробки,
- ✓ покращення читабельності коду
- ✓ можливість повторного використання компонентів.

Декомпозиція коду - розділення системи на логічні частини. Кожна частина вирішує окрему задачу.

Базові поняття модульного програмування

На практиці один програмний продукт може включати модулі, написані різними мовами програмування.

На вибір мови впливає складність реалізації алгоритму.

Для того, щоб зрозуміти принцип побудови багатомодульної програми, спочатку потрібно уявити велику програму, весь код якої міститься в одному файлі.

Оскільки будь-яка функція повинна бути оголошена перед її першим викликом, а кожна функція програми може викликати будь-яку іншу функцію, то **на самому початку програми** повинні бути **розміщені оголошення всіх функцій**, крім функції **main**.

Багатофайлові проекти

Модульність в мові C/C ++ підтримується за допомогою директив препроцесора, просторів імен, класів пам'яті, винятків і роздільної компіляції (строго кажучи, роздільна компіляція не є елементом мови, а відноситься до його реалізації).

У мові C/C++ модулі на рівні синтаксису не виділяються.

Реалізація модулів можлива з використанням **роздільної компіляції** та **файлів заголовків**.

Програма на C/C++ може складатись з декількох файлів, кожний з яких вміщує підпрограми та описи.

Головним файлом вважається той, що містить функцію **main()**.

Багатофайлові проекти

Роздільна компіляція – це обробка компілятором кожного файлу програми окремо.

Файли заголовків мають розширення **".h"** та вміщують описи глобальних констант, типів, змінних, підпрограм.

Описи констант, типів, змінних не відрізняються від тих, що розглядалися раніше.

Структура проєкту

Ієрархія проєкту

- src/ - вихідний код
- include/ - заголовні файли
- lib/ - зовнішні бібліотеки
- tests/ - модульні тести
- docs/ - документація

Імена файлів відображають їх призначення.

Модулі розділяються за функціональністю.

Для великих систем створюються окремі бібліотеки.

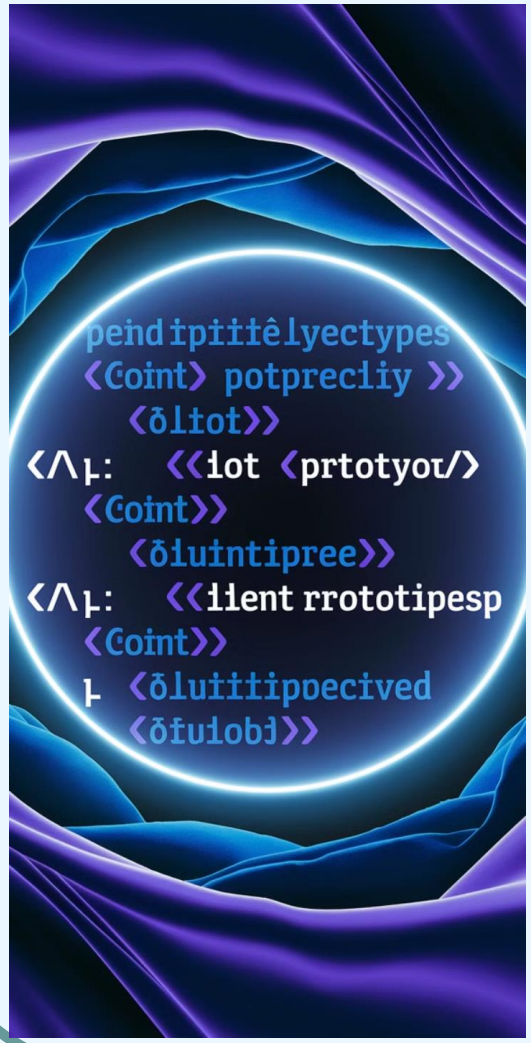
Багатофайлові проекти

Вихідні тексти сукупності функцій для вирішення будь-якої підзадачі, як правило, розміщуються в окремому модулі (файлі).

Такий файл називають **вихідним** (sources). зазвичай вінмає розширення .c або .cpp

Прототипи всіх функцій вихідного файлу виносять в окремий так званий **файл заголовків** (header file), для нього прийнято використовувати розширення .h або .hpp.

Заголовні файли (.h)



Оголошення функцій і структур

Заголовні файли містять прототипи функцій.

Вони визначають публічний інтерфейс модуля.

Include guards

```
#ifndef FILENAME_H
#define FILENAME_H
/* зміст */
#endif
```

Модулі не повинні створювати замкнені цикли включень. Використовуйте неповні типи.

Заголовні файли (.h)

При реалізації модулів для кожного модуля у C/C++ створюють **два файли з однаковим іменем** та з розширеннями **".h"** та **".c“/".cpp"**.

Файл заголовку з розширенням ***.h*** грає роль інтерфейсної частини модуля, а файл з розширенням .c/.cpp"– частини реалізації модуля.

Для використання модуля треба підключити відповідний файл заголовка за допомогою **#include**.

Файли заголовків (.h) — для чого потрібні?

Що таке header-файл?

Файл, який містить спільні оголошення для кількох модулів програми.

Що включають?

- Підключення інших заголовків та бібліотек
- Прототипи функцій
- Структури
- Масиви
- Покажчики
- Макроси
- Глобальні змінні
- Інші оголошення

Навіщо використовуються?

- Уникають дублювання коду в різних модулях
- Спрощують доступ до ресурсів модуля
- Виносять службову інформацію з основного коду
- Уникають перевантаження вихідного файлу

Переваги

- Централізоване керування інтерфейсом модуля
- Зручність підтримки великого проєкту
- Підтримка модульної структури коду

Файли заголовків

В заголовки прийнято розміщувати:

- визначення типів, що задаються користувачем, констант, шаблонів;
- оголошення (прототипи) функцій;
- оголошення зовнішніх глобальних змінних (з модифікатором extern);
- простори імен.

Файли заголовків

Для підпрограм у файлах заголовків наводять тільки заголовок, після якого ставлять ";":

extern t f(t1 x1, ..., tn xn);

де t, ti – тип, xi – змінні.

Ключове слово **extern** вказує на те, що підпрограма реалізована у зовнішньому файлі.

Файли заголовків підключають до інших файлів за допомогою директиви #include. Наприклад,

#include “myfile.h”

Включення

Ми вже знайомі з цією директивою та постійно користувались нею для підключення стандартних бібліотек.

Різниця із підключенням власних файлів заголовків в тому, що для стандартних бібліотек використовують символи “< >” замість лапок:

#include <stdio.h>

Директиви препроцесорної обробки

Директиви – інструкції препроцесора. Обробка програми препроцесором здійснюється перед її компіляцією. Призначення препроцесорної обробки:

- включення у файл, що компілюється, інших файлів,
- визначення символічних констант і макросів,
- встановлення режиму умовної компіляції програми і умовного виконання директив препроцесора.

Директиви препроцесорної обробки

Синтаксис:

– директиви повинні починатися з символу "#".

– після директив препроцесора не ставиться " ;".

Основні директиви:

#include – включає копії вказаного файла в те місце програми, де знахо-диться ця директива;

#define – здійснює макропідстановку – заміняє всі входження ідентифі-катора у програмі на текст, що слідує в директиві за ідентифікатором;

#undef – анулює визначення символічних констант і макросів;

#if, #elif, #else, #endif – здійснюють умовну компіляцію;

#ifdef і #ifndef – використовуються замість **#if – defined (…)**;

Основні директиви:

#line – повідомляє компілятору про зміну імені програми і порядку нумерації рядків;

#error – вказує компілятору, що треба надрукувати повідомлення про помилку і перервати компіляцію (зазвичай використовують у середині умовних директив);

#pragma – дозволяє керувати можливостями компіляції.

Основні директиви:

Існує і ряд **інших директив**.

Визначені макроси ANSI:

__**DATE**__ – рядок у формі mmm.dd.yyyy – дат створення даного файлу;

__**TIME**__ – час початку обробки поточного файлу у форматі hh:mm:ss ;

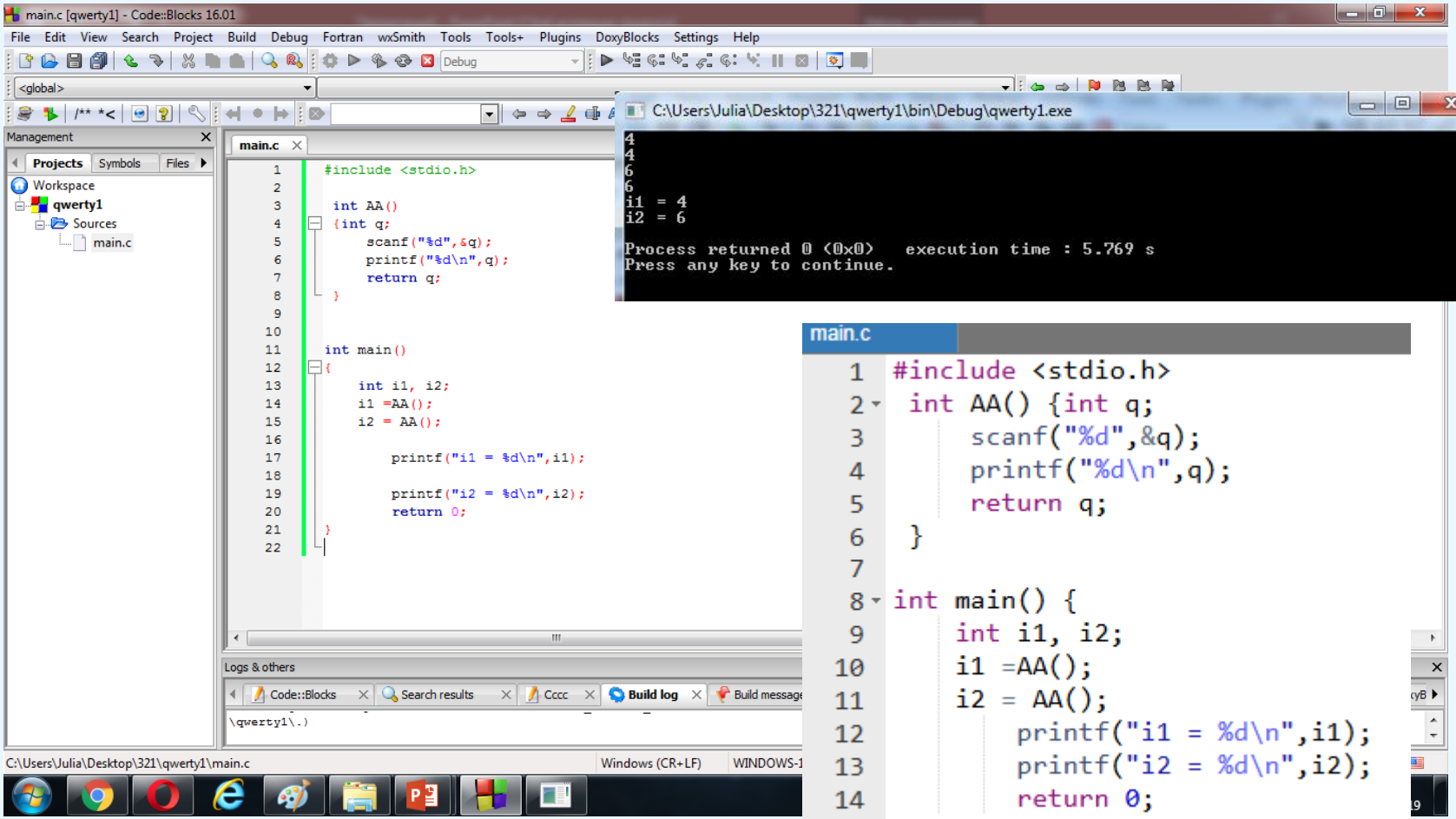
__**FILE**__ – ім’я поточного файлу;

__**LINE**__ – номер поточного рядка;

__**STDC**__ – визначений, якщо встановлено режим сумісності з ANSI.

Створення власної бібліотеки

Приклад початкового коду



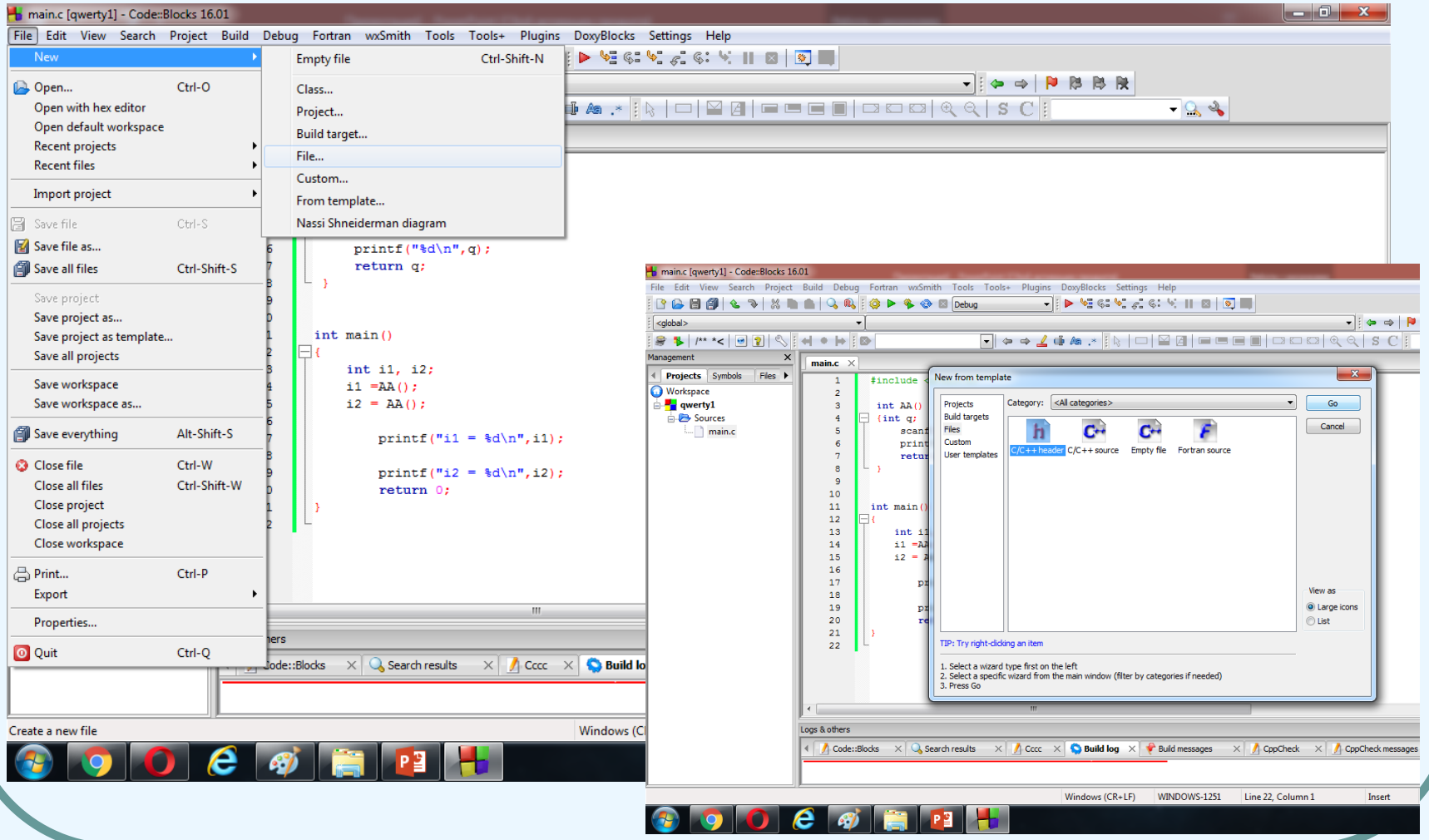

```c
#include <stdio.h>
 int AA() {int q;
      scanf("%d",&q);
      printf("%d\n",q);
      return q;
 }

int main() {
     int i1, i2;
     i1 =AA();
     i2 = AA();
          printf("i1 = %d\n",i1);
          printf("i2 = %d\n",i2);
          return 0;
}
```

1 крок. Розробити програму

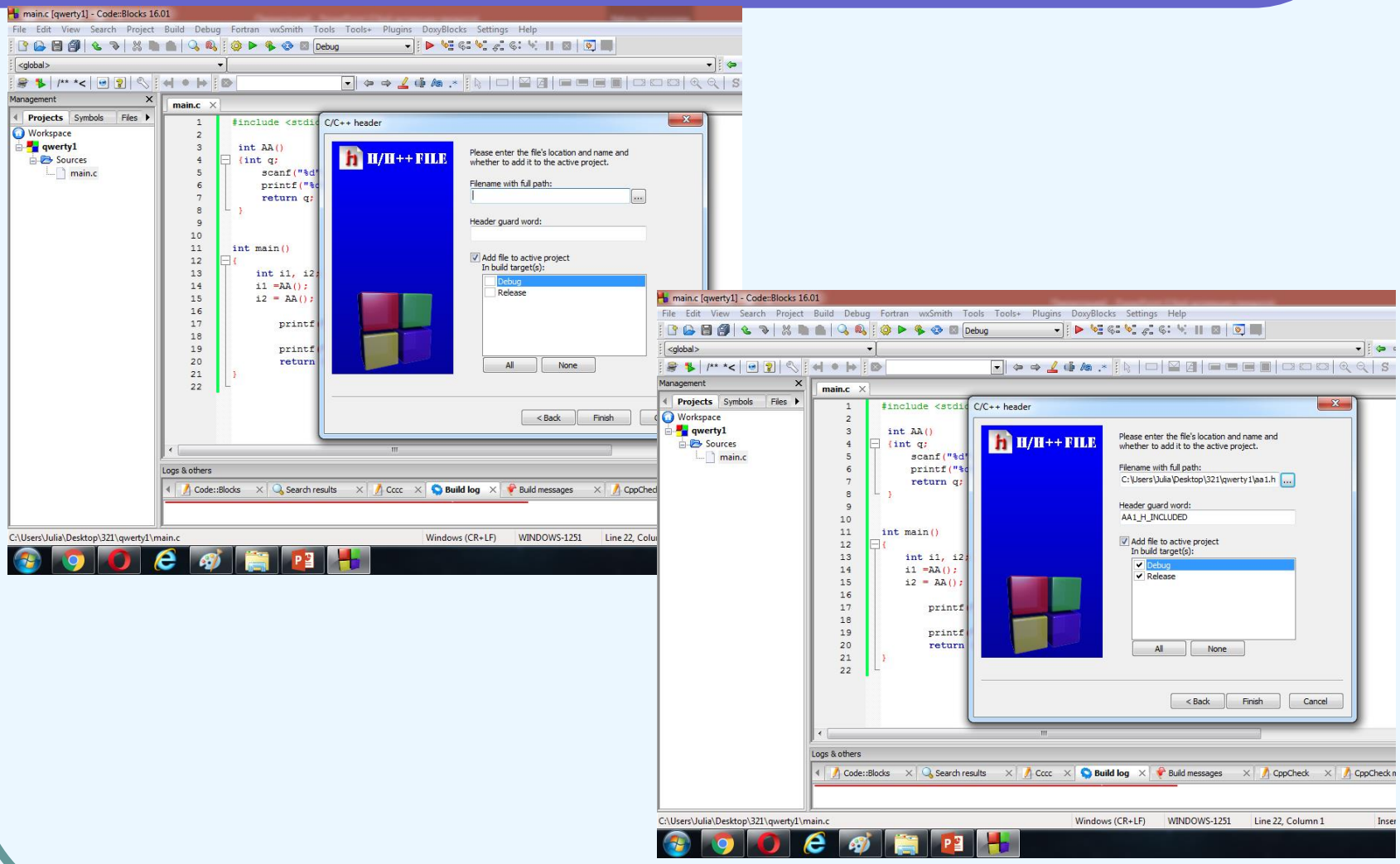
Щоб створити **власну бібліотеку** у мові програмування C/C++, потрібно виконати кілька кроків.

1. Створіть заголовочний файл (header file)
2. Створіть файл реалізації (source file)
3. Створіть головний файл (main file)

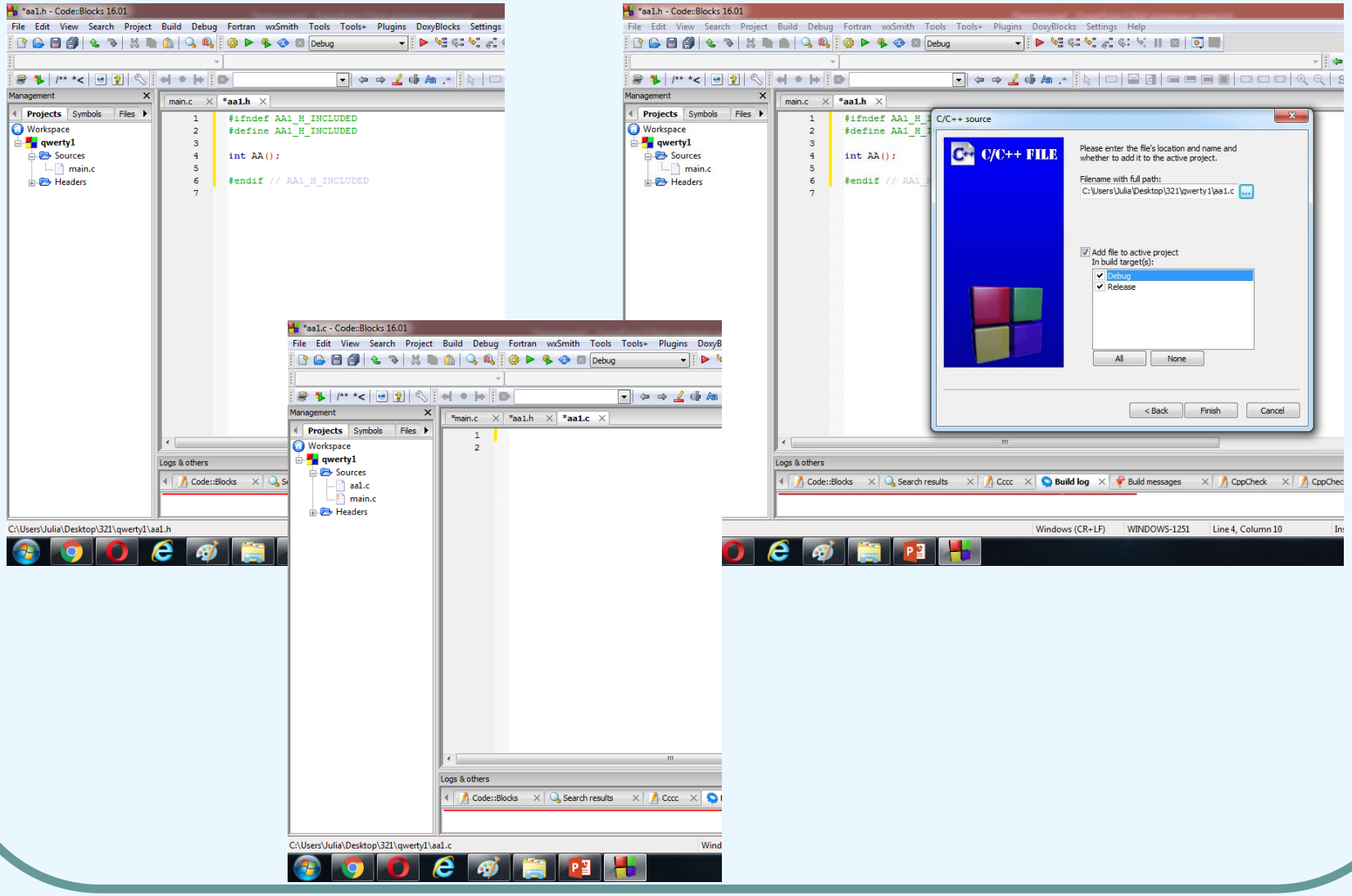
Code::Blocks


main.c [qwerty1] - Code::Blocks 16.01
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
New
Open... Ctrl-O
Open with hex editor
Open default workspace
Recent projects
Recent files
Import project
Save file Ctrl-S
Save file as...
Save all files Ctrl-Shift-S
Save project
Save project as...
Save project as template...
Save all projects
Save workspace
Save workspace as...
Save everything Alt-Shift-S
Close file Ctrl-W
Close all files Ctrl-Shift-W
Close project
Close all projects
Close workspace
Print... Ctrl-P
Export
Properties...
Quit Ctrl-Q
Empty file Ctrl-Shift-N
Class...
Project...
Build target...
File...
Custom...
From template...
Nassi Shneiderman diagram
int main()
{
int i1, i2;
i1 =AA();
i2 = AA();
printf("i1 = %d\n",i1);
printf("i2 = %d\n",i2);
return 0;
}
Create a new file
New from template
Projects
Build targets
Files
Custom
User templates
Category: <All categories>
C/C++ header
C/C++ source
Empty file
Fortran source
Go
Cancel
TIP: Try right-clicking an item
1. Select a wizard type first on the left
2. Select a specific wizard from the main window (filter by categories if needed)
3. Press Go
Workspace
qwerty1
Sources
main.c
Windows (CR+LF) WINDOWS-1251 Line 22, Column 1 Insert

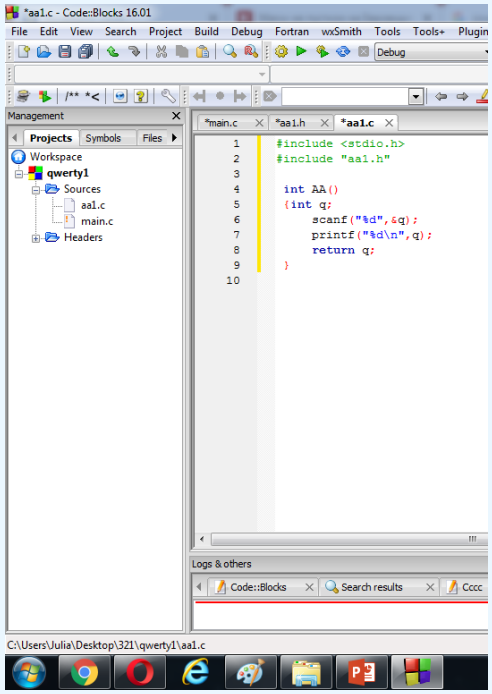
Code::Blocks

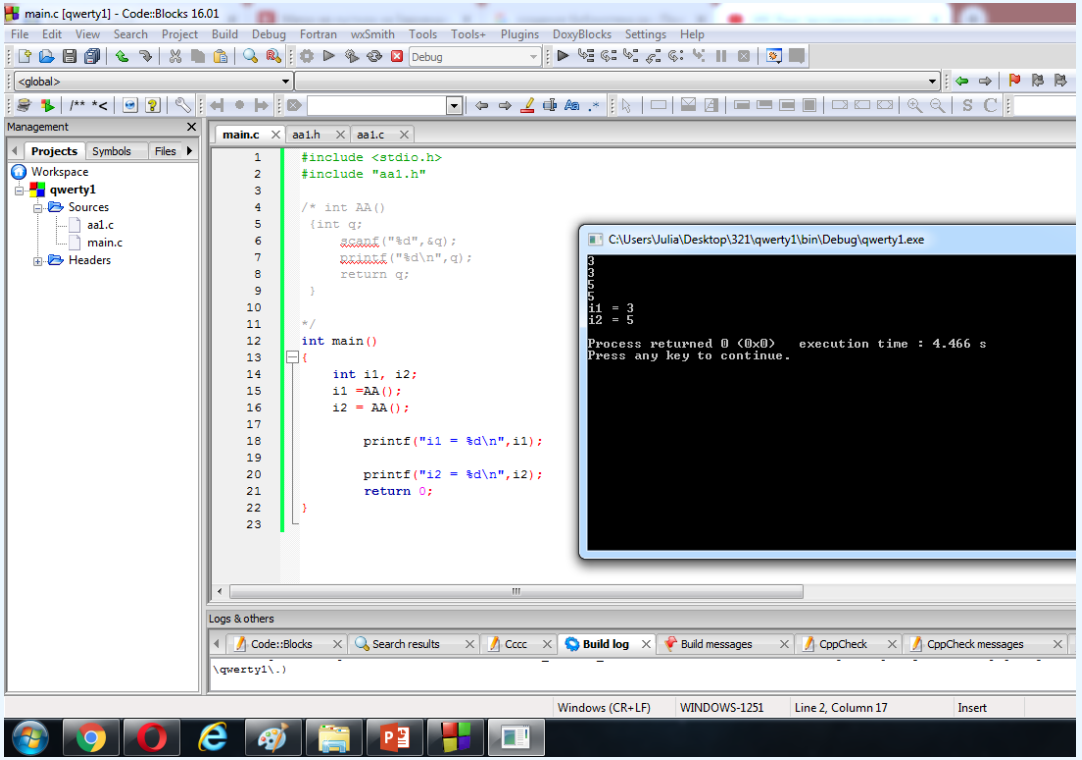

main.c [qwerty1] - Code::Blocks 16.01
C/C++ header
H/H++ FILE
Please enter the file's location and name and whether to add it to the active project.
Filename with full path:
Header guard word:
Add file to active project
In build target(s):
Debug
Release
All
None
< Back
Finish
Cancel
Logs & others
Code::Blocks
Search results
Cccc
Build log
Build messages
CppCheck
Windows (CR+LF)
WINDOWS-1251
Line 22, Column 1

Code::Blocks


*aa1.h - Code::Blocks 16.01
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings
Management
Projects Symbols Files
Workspace
qwerty1
Sources
main.c
Headers
main.c *aa1.h
#ifndef AA1_H_INCLUDED
#define AA1_H_INCLUDED
int AA();
#endif // AA1_H_INCLUDED
Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck
C:\Users\Julia\Desktop\321\qwerty1\aa1.h
Windows (CR+LF) WINDOWS-1251 Line 4, Column 10
*aa1.c - Code::Blocks 16.01
*main.c *aa1.h *aa1.c
aa1.c
C/C++ source
C/C++ FILE
Please enter the file's location and name and whether to add it to the active project.
Filename with full path:
C:\Users\Julia\Desktop\321\qwerty1\aa1.c
Add file to active project
In build target(s):
Debug
Release
All
None
< Back
Finish
Cancel

Code::Blocks


*aa1.c - Code::Blocks 16.01
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugin
Management
Projects Symbols Files
Workspace
qwerty1
Sources
aa1.c
main.c
Headers
*main.c *aa1.h *aa1.c
1 #include <stdio.h>
2 #include "aa1.h"
3
4 int AA()
5 {int q;
6 scanf("%d",&q);
7 printf("%d\n",q);
8 return q;
9 }
10
Logs & others
Code::Blocks Search results Cccc
C:\Users\Julia\Desktop\321\qwerty1\aa1.c


main.c [qwerty1] - Code::Blocks 16.01
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects Symbols Files
Workspace
qwerty1
Sources
aa1.c
main.c
Headers
main.c aa1.h aa1.c
1 #include <stdio.h>
2 #include "aa1.h"
3
4 /* int AA()
5 {int q;
6 scanf("%d",&q);
7 printf("%d\n",q);
8 return q;
9 }
10
11 */
12 int main()
13 {
14 int i1, i2;
15 i1 =AA();
16 i2 = AA();
17
18 printf("i1 = %d\n",i1);
19
20 printf("i2 = %d\n",i2);
21 return 0;
22 }
23
C:\Users\Julia\Desktop\321\qwerty1\bin\Debug\qwerty1.exe
3
3
5
5
i1 = 3
i2 = 5
Process returned 0 (0x0) execution time : 4.466 s
Press any key to continue.
Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck CppCheck messages
\qwerty1\.)
Windows (CR+LF) WINDOWS-1251 Line 2, Column 17 Insert

Дякую за увагу!