

TiDA 04: kompilacja z linii komend + Python

Cel zajęć

Na tych zajęciach:

- Nauczysz się jak zbudować i uruchomić program C++ z linii komend
- Zrobisz prostą automatyzację w Pythonie zapisującą wynik do pliku .log.

Pracujemy na Windows + kompilator **MSVC (c1.exe)**. Najprościej używamy **Developer Command Prompt for VS 2022 (x64)**.

TEORIA & CZĘŚĆ PRAKTYCZNA

Przygotowanie folderu

Utwórz folder projektu, np. C:\projekty\TiDA_04_podst.

W środku będą dwa pliki:

```
TiDA_04_podst/  
├─ main.cpp  
└─ build_and_run.py
```

Prosty program C++ (main.cpp)

Skopiuj poniższy kod do main.cpp:

```
#include <iostream>  
using namespace std;  
  
int main() {  
    cout << "Hello from MSVC!" << endl;  
    return 0;  
}
```

Kompilacja z linii komend (bez Pythona)

1. Otwórz **Start** → **Developer Command Prompt for VS 2022** → wybierz **x64**.
2. Przejdź do folderu projektu:

```
cd C:\projekty\blok04_prosty
```

3. Skompiluj i uruchom:

```
cl main.cpp /O2 /MD /Feprogram.exe  
program.exe
```

Powinieneś zobaczyć: Hello from MSVC!

◆ Co znaczą podstawowe przełączniki?

- **/O2** – szybki kod (optymalizacja, używamy w Release).
- **/Od** – brak optymalizacji (łatwe debugowanie, wolniej).
- **/MD** – biblioteka runtime w DLL (mniejszy EXE).
- **/MDd** – jak wyżej, ale do Debug.

Na tym etapie **wystarczy znać**: /O2 /MD (szybko) oraz /Od /MDd (debugowo).

◆ Prosty skrypt Python do kompilacji i logowania (build_and_run.py)

Ten skrypt:

- kompiluje main.cpp,
- uruchamia program.exe,
- zapisuje wynik do pliku output_YYYYMMDD_HHMMSS.log.

Uwaga: Uruchamiaj go z **Developer Command Prompt** (wtedy środowisko dla cl jest gotowe).

```
# build_and_run.py (prosta wersja)
```

```
import subprocess, datetime
```

```
source = "main.cpp"
```

```
exe = "program.exe"
```

```
# 1) kompilacja
```

```
subprocess.check_call(["cl", source, "/O2", "/MD", f"/Fe{exe}"])
```

```
# 2) uruchomienie i zebranie wyjścia
```

```
result = subprocess.run([exe], capture_output=True, text=True)
```

```
# 3) zapis do pliku .log
```

```
stamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
```

```
log_name = f"output_{stamp}.log"
```

```
with open(log_name, "w", encoding="utf-8") as f:  
    f.write(result.stdout)
```

```
print("Zapisano:", log_name)
```

Uruchomienie (z Developer Command Prompt):

```
python build_and_run.py
```

◆ (Opcjonalnie) Uruchamianie bez Developer Prompt

Jeśli chcesz, aby skrypt sam ustawiał środowisko, podaj ścieżkę do `vcvars64.bat` i użyj wersji minimalnej z łańcuchem `&&`:

```
# build_and_run_vcvars.py (opcjonalnie)
import subprocess, datetime

VCVARS = r"C:\Program Files\Microsoft Visual
Studio\2022\Community\VC\Auxiliary\Build\vcvars64.bat"
CMD = f'"{VCVARS}" && cl main.cpp /O2 /MD /Feprogram.exe'

# 1) kompilacja w środowisku VS
subprocess.check_call(CMD, shell=True)

# 2) uruchomienie i log
out = subprocess.run(["program.exe"], capture_output=True, text=True)
name = "output_" + datetime.datetime.now().strftime("%Y%m%d_%H%M%S") + ".log"
open(name, "w", encoding="utf-8").write(out.stdout)
print("Zapisano:", name)
```

Szybkie FAQ

- **Nie działa `cl`** → Uruchom **Developer Command Prompt for VS 2022 (x64)**.
 - **Gdzie jest mój EXE?** → Tam, gdzie wywołujesz `cl` (lub jak podasz `\Feścieżka.exe`).
 - **Po co log?** → Żeby móc porównać wyniki automatycznie i dołączyć je później do raportu.
-

Krótkie wprowadzenie: testy automatyczne i CI

- **Test automatyczny** — skrypt sprawdza, czy program działa (np. czy wypisał "Hello").
- **Continuous Integration (CI)** — każde wgranie kodu do repozytorium (np. GitHub) automatycznie robi: **build → test → raport**.

Na tym poziomie wystarczy rozumieć: nasz `build_and_run.py` to **pierwszy krok** do testów automatycznych i CI.

ZADANIA

1. Zmień optymalizację na Debug i porównaj:

```
cl main.cpp /Od /MDd /Feprogram_debug.exe
program_debug.exe
```

Czy wynik jest taki sam?

2. Zapisz do logu wynik działania wersji Release i Debug.
W logach dopisz ręcznie, której komendy użyłeś.
 3. Zmień tekst w `main.cpp` tak, aby wypisywał Twoje imię.
 4. Zmodyfikuj `build_and_run.py`, aby do pliku `.log` dopisywał na końcu linię **KONIEC LOGU**.
-