Systemy wbudowane dla automatyki

Prowadzący: **dr inż. Krzysztof Urbański**

ARES00600 Systemy wbudowane dla automatyki
Automatyka i Robotyka (AiR)
Specjalność: Elektroniczne systemy automatyki (AEU)
Liczba godzin zajęć zorganizowanych w Uczelni (ZZU) 30W, 30L
Liczba godzin całkowitego nakładu pracy studenta (CNPS) **90W, 60L**
Forma zaliczenia W: Egzamin L: Zaliczenie na ocenę
Wymagania wstępne: Wiedza z zakresu kursu Mikroprocesory.

Zasady zaliczenia kursu

Kurs kończy się E: egzaminem, a więc w trakcie sesji będą 2 terminy: proponuję 4.VII oraz 11.VII. godz. 7:30, pok. 244 C4.
7-minutowe Q: quizy na początku każdego wykładu, lista kilku z zagadnień do opracowania będzie wcześniej znana. Mogą dotyczyć wcześniejszych, ale też bieżącego wykładu.
Ostatni wykład w semestrze K: kolokwium zaliczeniowe (termin „0”).

Zaliczenie wykładu można więc uzyskać na podstawie:

- Q: Quizy na wykładach
- K: Kolokwium zaliczeniowe
- E: Egzamin

Zaliczenie laboratorium: proporcjonalnie do liczby zrealizowanych zadań.
Warunkiem koniecznym zaliczenia bloku kursów jest pozytywna ocena z wykładu oraz z laboratorium.

Przedmiotowe Efekty Kształcenia

C1. Zdobycie wiedzy z zakresu **architektury mikrokontrolerów**
C2. Zdobycie podstawowej wiedzy na temat podstawowych **bloków peryferyjnych** implementowanych w układach mikrokontrolerowych
C3. Zdobycie umiejętności wykorzystania **interfejsów komunikacyjnych wykorzystywanych w automatyce**.
C4. Zdobycie podstawowej wiedzy na temat podstawowych bloków implementowanych w **strukturach układów programowalnych** z uwzględnieniem aplikacji przemysłowych.

Z zakresu wiedzy:

PEU W01 - zna zasady projektowania systemów mikroprocesorowych pod kątem wymaganego zastosowania oraz wymaganej wydajności

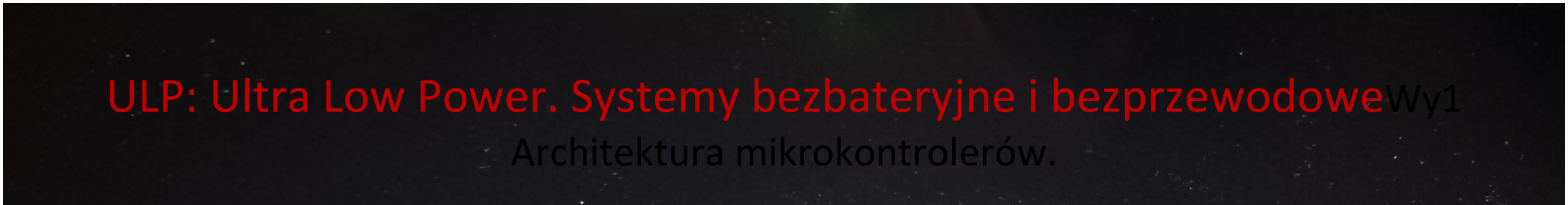
PEU W02 - posiada wiedzę na temat protokołów komunikacyjnych stosowanych w systemach automatyki takich jak ProfiBus, ProfiNet czy ModBus

PEU W03 - posiada wiedzę umożliwiającą dobór układu **FPGA** pod kątem wymaganej wydajności oraz oferowanych układów peryferyjnych do zadanej aplikacji

Z zakresu umiejętności:

PEU U01 - Umie przygotowywać, tworzyć, weryfikować i wdrażać oprogramowanie testujące i użytkowe mikrokontrolerów

PEU U02 - potrafi wykorzystać bloki składowe układów FPGA w zastosowaniach dla układów automatyki

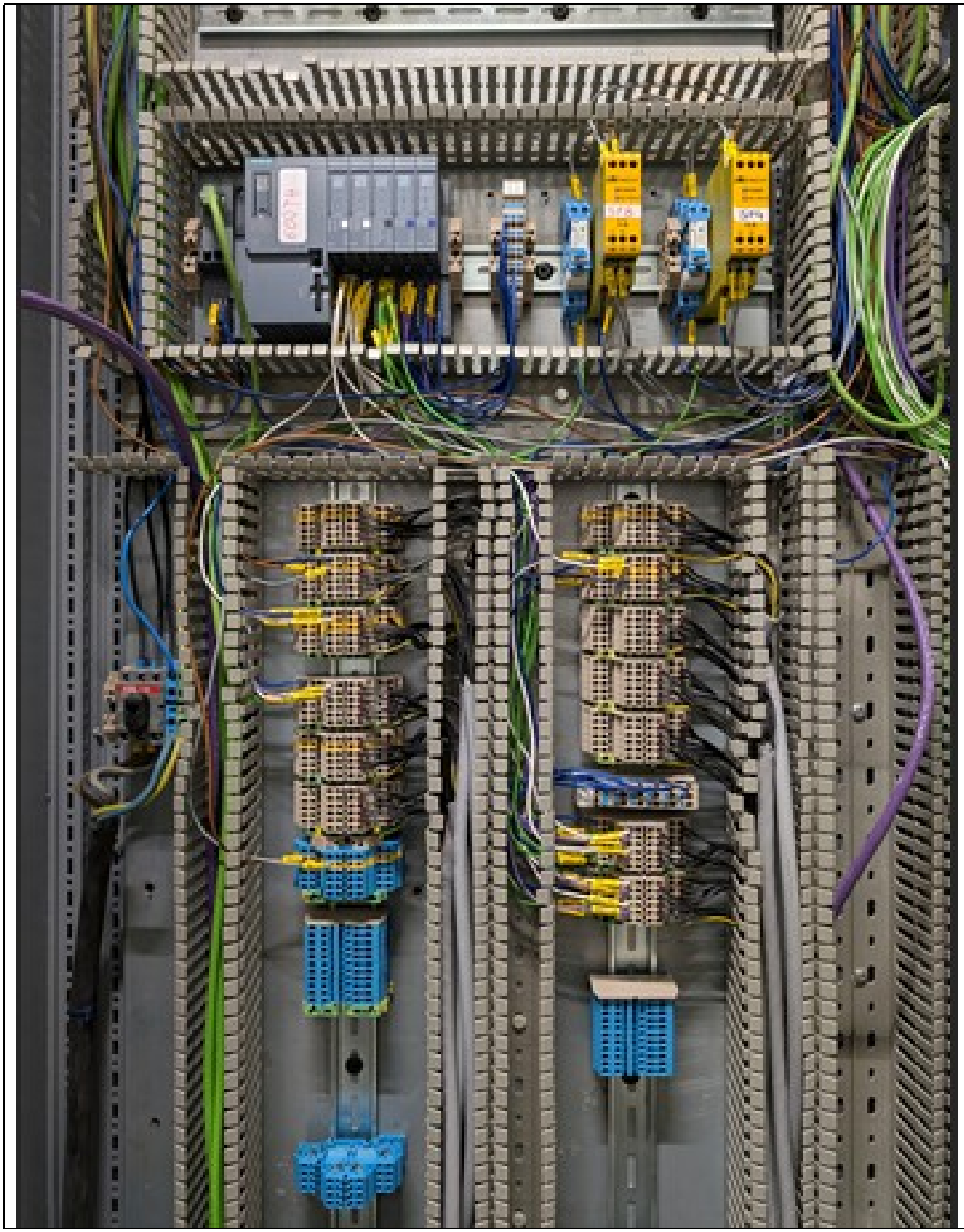


Co zaplanowałem w tym semestrze:

- datagramy UDP w sieci lokalnej - działanie modelu ISO/OSI, ale jeszcze bez stosu TCP. Przy tej okazji big-endian/little-endian oraz inne zagadnienia związane z reprezentacją danych w systemach mp. Kodowanie float, w tym w rejestrach PLC, pakowanie struktur danych w C i innych językach. Nieoptymalizowane DB w PLC do komunikacji z peryferiami lub SCADA.
- MCU jako master modbus RTU - komunikacja z modułem peryferyjnym, 8x Relay lub innym układem. Na wykładzie ProfiBus, ProfiNet, CAN, DMX512.
- MCU jako modbus-slave
- obsługa czujników 4-20mA - przez konwerter 2-20 na 0-3.3. Na wykładzie także HART i IO-Link. Konwersja RTD (Pt100) na 4-20 lub 0-10/0-3v3.
- metody HTTP w MCU - na gotowym stosie TCP, ale HTTP i JSON od podstaw. MCU jest wtedy inteligentnym czujnikiem działającym po WiFi. Raczej uPython na Pico2W, ale rozwiązania w C/C++ też są ok.

- Generyczna platforma: serwer Apache (VM lub SBC), PHP, okresowe wysyłanie danych pomiarowych, rejestracja czujnika WiFi w systemie, prezentacja danych, obsługa utraty połączenia - wszystko w wersji topornej, pokazującej od strony niskiego poziomu jak się to realizuje z pominięciem gotowych kombajnów typu Home Assistant. Tutaj więcej o mechanizmach bezpieczeństwa, kryptografii symetrycznej, asymetrycznej i funkcjach skrótu.
- Proste interfejsy graficzne w systemach o ograniczonych zasobach - OLED/IPS + biblioteka graficzna na mikrokontroler. Na wykładzie więcej - ePaper, OLED, IPS, TFT + kilka sposobów obsługi dotyku. Na wykładzie też obsługa HMI do PLC, czyli jak to naprawdę się robi w automatyce.
- Techniki ULP - praktyki projektowe i specyficzne rozwiązania w ultra-nisko-energetycznych MCU. Zawody - kto zaprogramuje MCU tak, aby jak najdłużej działał na jednym ładowaniu kondensatora wysyłając bezprzewodowo odczyt danych z czujnika co kilka sekund.

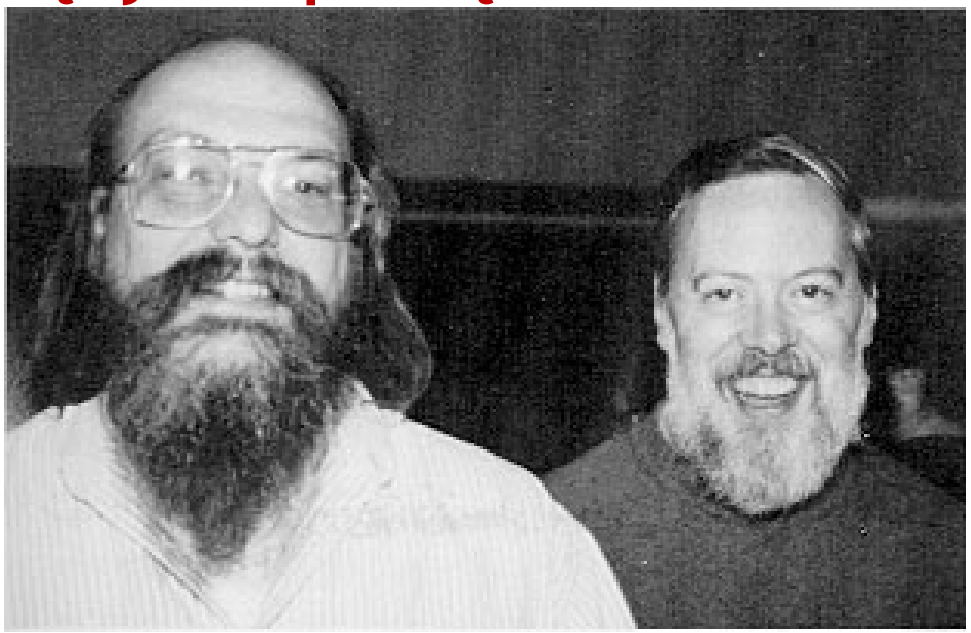
- Separacja galwaniczna interfejsów cyfrowych (ADuM, transformatory Ethernet, transoptory). Interfejsy iskrobezpieczne. Zabezpieczenia wejść i wyjść MCU - przeciwprzepięciowe, przeciwzwarciowe, …
- Czujniki bezprzewodowe z Zigbee, BLE, WiFi, LoRa - na wykładzie teoria, na laborce spróbujemy z NRF24L01 lub LoRa.
- MQTT broker na Linuksie: VM lub SBC, klienci na MCU+WiFi oraz Zigbee2MQTT
- HomeAssistant, użyjemy gotowych integracji (esphome, jest port na picoW, można użyć też samego ESP32). Dla serwera HA wystarczy maszyna wirtualna. Dodatkowo można pokazać realizację na Proxmox na cienkim kliencie oraz na Raspberry 4 lub 5.
- Dla HA istnieją sprawdzone i działające integracje z AI - np. analiza zapytań w języku naturalnym oraz analiza obrazów z kamer - to też da się uruchomić na zajęciach.
- Skrypt w pythonie robiący za SCADA i komunikujący się przez ProfiNet z PLC S7-1200 z wystawionym DB.



Po tych zajęciach:

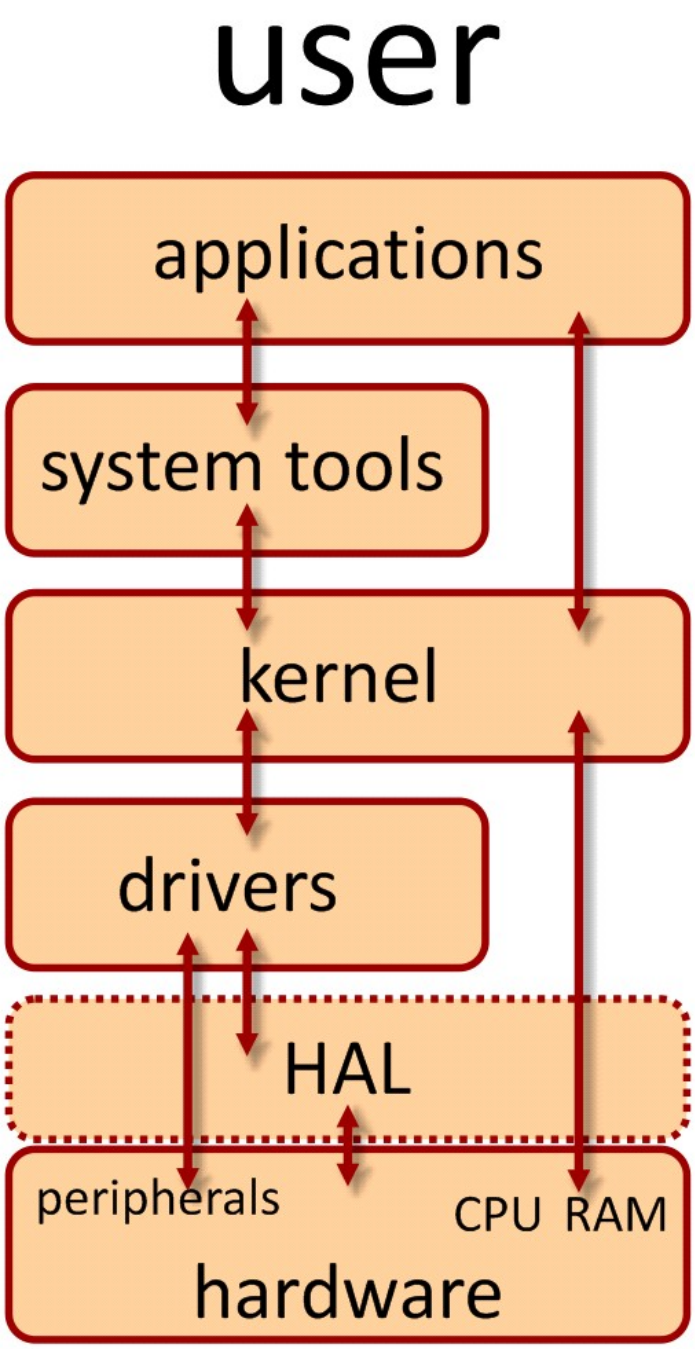
- Zbudujesz własny czujnik, który da się podłączyć do prawdziwego PLC (ModBus, 4-20mA).
- Napiszesz program, który w bazie danych SQL zapisuje parametry procesu pobrane z PLC (Profinet).
- Poznasz przeznaczenie i zasadę działania tych żółtych urządzeń.
- Uświadomisz sobie, że ChatGPT na razie nie potrafi wykonać pracy, której efekty widać obok.
- Jeśli Ty też (jeszcze) nie potrafisz, to jest plan aby to zmienić.

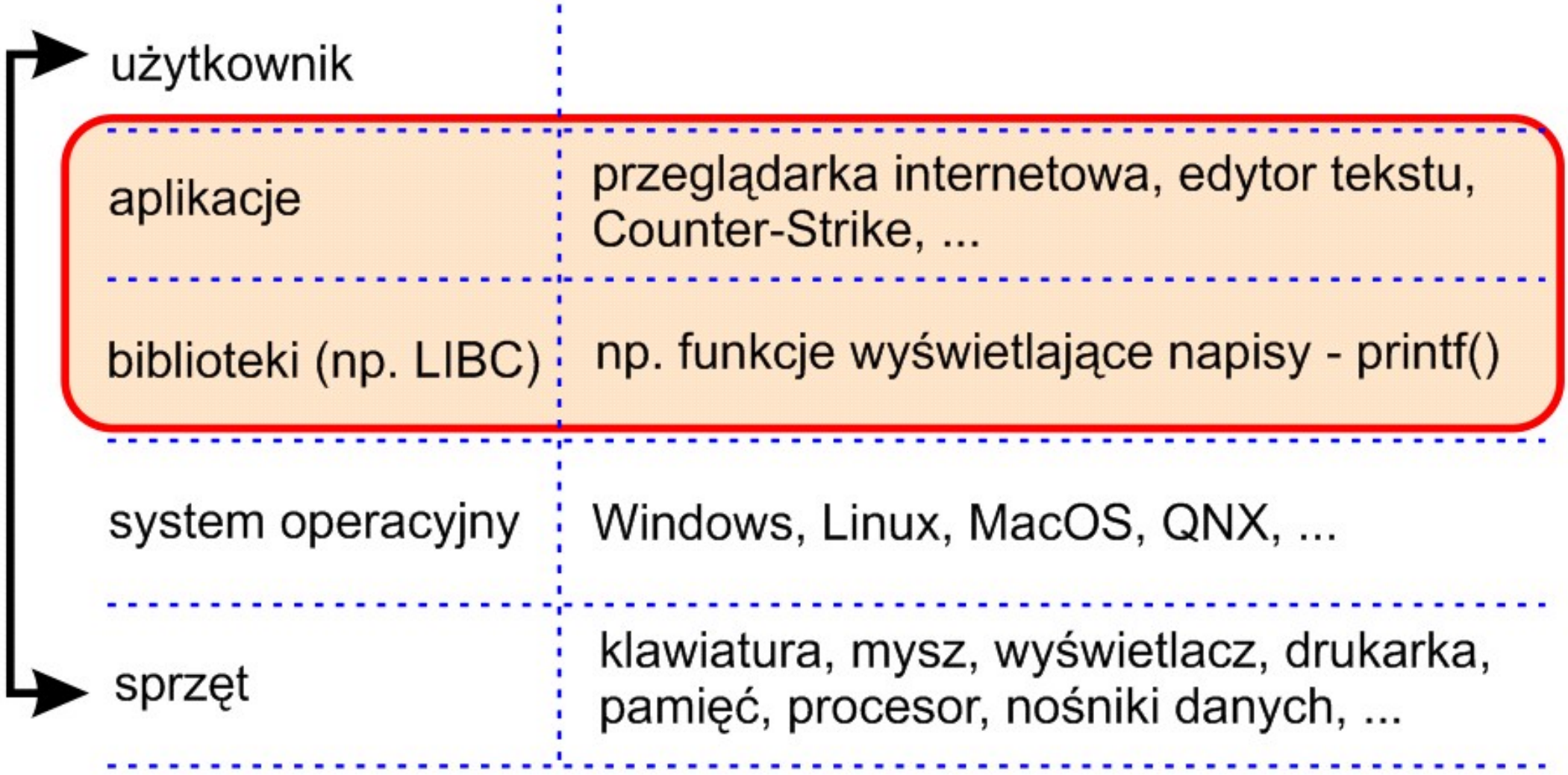
Język C: początki



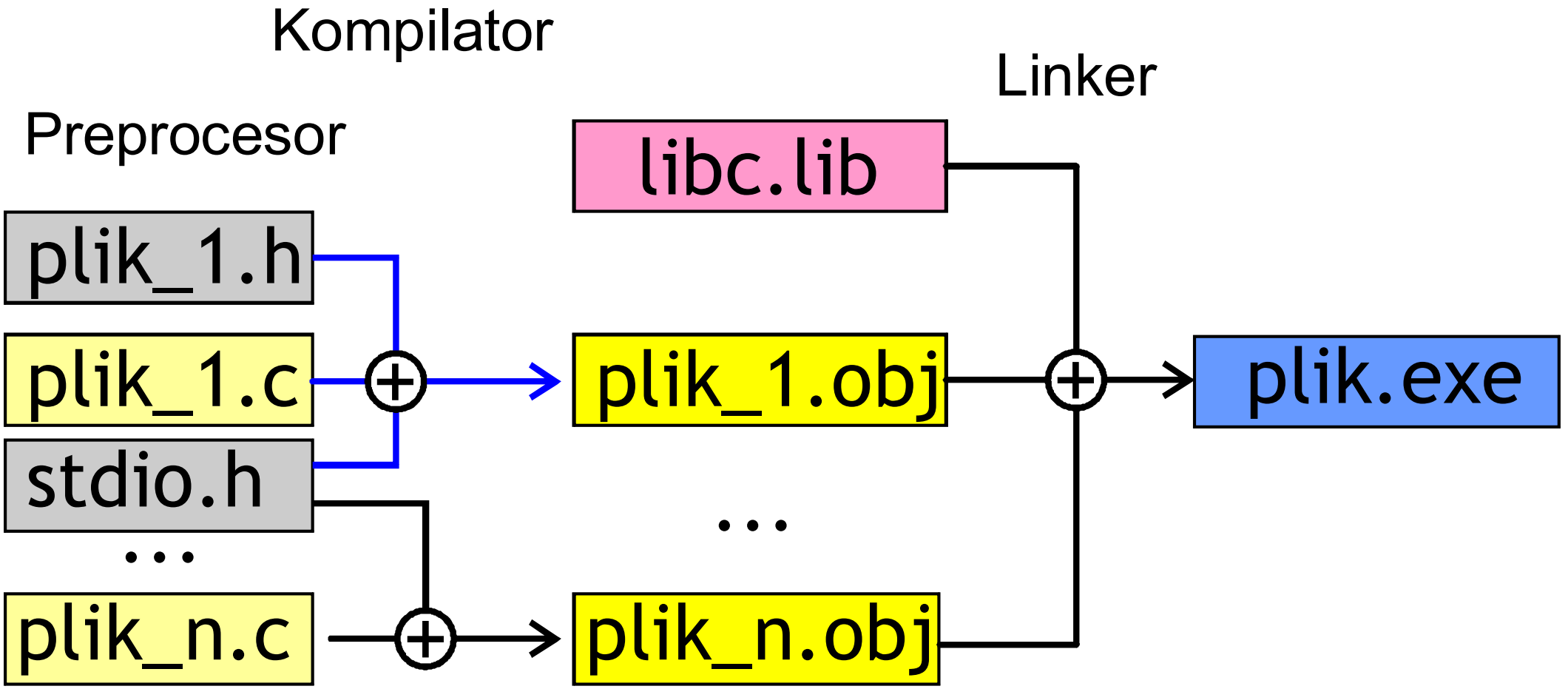
Ken Thompson i **Dennis Ritchie**

Umowna data powstania systemu UNIX (napisany w asemblerze):
1 stycznia 1970 00:00:00 UTC (*Unix Epoch)*, 1973 przepisany w C.
Początkowo język C powstał po to, aby ułatwić "portowanie" UNIXa.



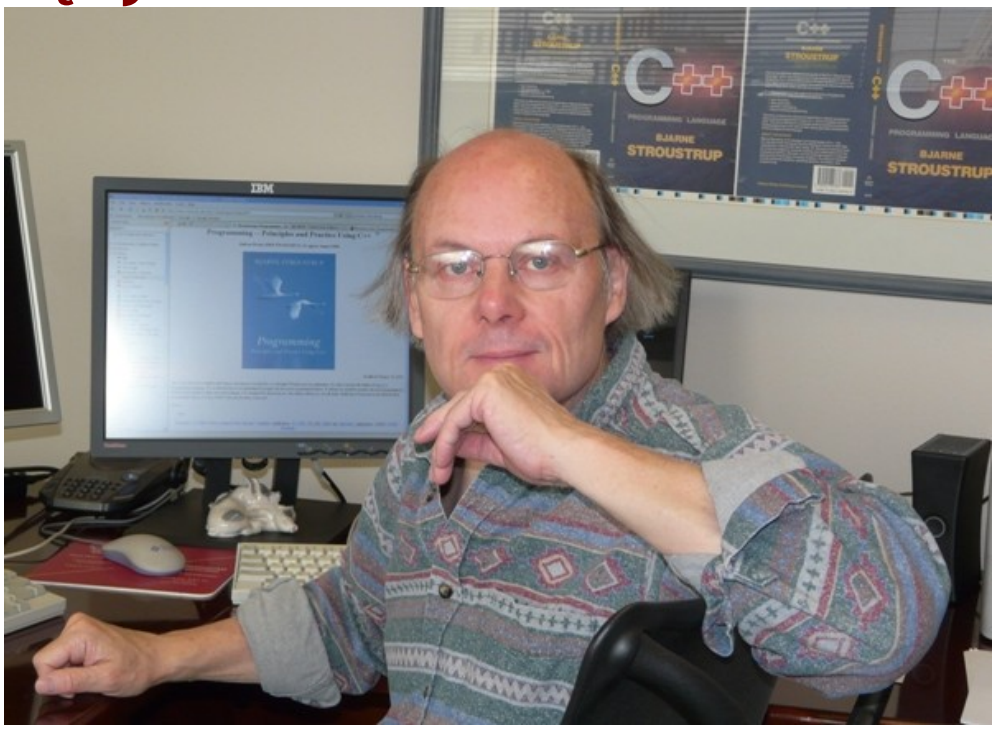


Warstwy abstrakcji - miejsce programu i bibliotek C/C++ między sprzętem a użytkownikiem.



Typowy przebieg kompilacji projektu C. Dla ułatwienia całością tego procesu może sterować IDE.
Rozszerzenia skompilowanych modułów .o, .obj czy pliku wynikowego .exe są specyficzne dla użytego kompilatora i platformy. Ważniejsze jest nazewnictwo plików źródowych: .c albo .cpp, co domyślnie powoduje wybór kompilatora C lub C++.

Język C++



Bjarne Stroustrup

1979 - początek prac nad językiem "C z klasami".

1983 - nazwa C++.
1985 - publikacja *The C++ Programming Language*.

1998 - pierwszy standard "C++98"
Aktualnie: C++23

Język C jest innym językiem niż C++, ale w większości sytuacji da się skompilować kod języka C kompilatorem C++. Ponieważ C++ oferuje znacznie większe możliwości (jest istotnym rozwinięciem C), to odwrotna operacja najczęściej zakończy się fiaskiem (kompilator C nie rozumie nowszych instrukcji C++).

W najbardziej podstawowych aspektach (na początku nauki) można te języki traktować niemalże jak zamienniki, pamiętając jednak, że C nie jest ścisłym podzbiorem C++.

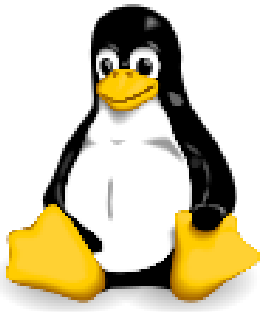
Wybrane paradygmaty programowania a C++

- p. deklaratywne
- p. funkcyjne
- p. logiczne
- **p. uogólnione (generyczne)** - częściowo
- **p. imperatywne ***
- **p. proceduralne ***
- **p. strukturalne ***
- **p. obiektowe ***
- **p. sterowane zdarzeniami** - w niektórych środowiskach
- inne

Wniosek: język C++ jest językiem wieloparadygmatowym (wspiera wiele filozofii realizacji programu).

Na tym kursie jednak nie skorzystamy z większości jego możliwości.

Linux czy UNIX?



GNU ::= GNU's Not UNIX definicja rekurencyjna ;)

Projekt **GNU** (1983): opracować system zgodny z UNIXem, ale darmowy (otwarty?). Rok 1991: nadal nic.

10.1991: Linus Torvalds *" Hello everybody out there using minix - I'm doing a (free) operating system (**just a hobby, won't be big and professional like gnu**) for 386(486) AT clones. This has been brewing since april, and is starting to get ready (...) I've currently ported bash(1.08) and gcc(1.40), and things seem to work."*

1992: pierwsze "dystrybucje" na licencji GNU.

‘How does Linux actually work? — Use the Source, Luke!’ (*open sourcers joke*)

Strumień cout a standardowe wyjście

Historycznie język C powstał w zamierzchłych czasach (lata 70-te ubiegłego wieku). Systemy komputerowe dysponowały wtedy ograniczonymi możliwościami jeśli chodzi o interakcję z użytkownikiem.

Powszechnie stosowanym interfejsem była klawiatura oraz konsola tekstowa, przy czym ta druga mogła być drukarką tekstową lub wyświetlaczem znakowym.

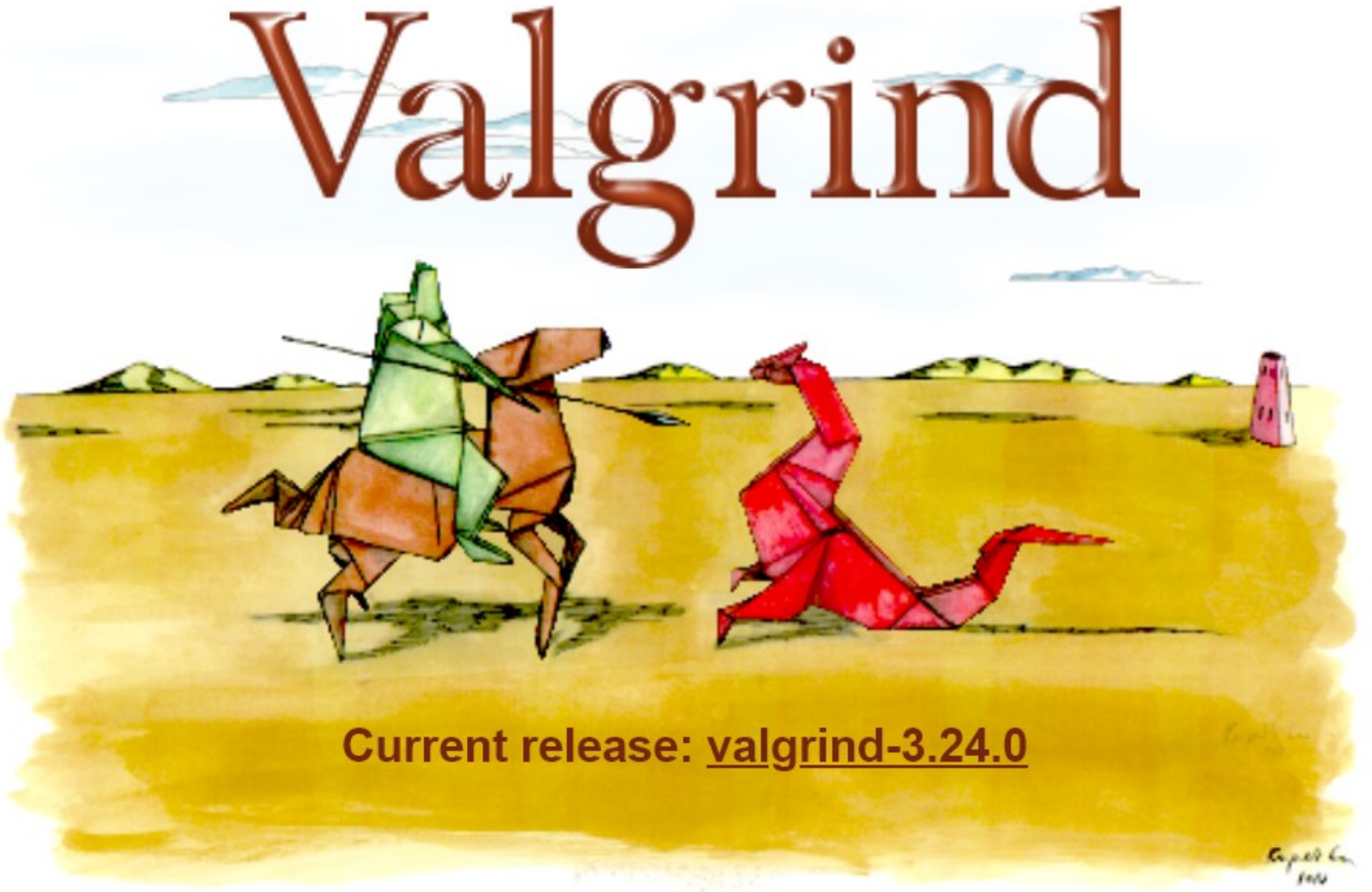
Pojęcie standardowego wejścia-wyjścia programu: stdio (ang. standard input-output) jest bezpośrednio związane z rozwiązaniami używanymi w systemach UNIX, a dzisiaj Linux. Tekstowy interpreter komend w Windows również używa podobnych mechanizmów.

Wyróżniamy 3 domyślne strumienie: stdin, stdout, stderr, do których ma dostęp program w C/C++. To ostatnie stanowi standardowe wyjście błędu, do którego powinny być kierowane komunikaty o błędach które pojawią się w trakcie działania programu.

Typowo stdin jest podłączony do klawiatury, a stdout oraz stderr do wyświetlacza tekstowego (konsoli).

Możliwe (i często praktykowane) jest **przekierowanie** tych strumieni do innych plików lub urządzeń. Można też wyniki działania (wyjście) jednego programu bezpośrednio "podłączyć" do wejścia innego programu, tworząc łańcuch powiązanych ze sobą aplikacji.

Standardowe wejście jest buforowane, blokujące, zorientowane na linie. Przez wysłaniem porcji danych i zatwierdzeniem przyciskiem Enter/Return można dane edytować. To zachowanie zwykle nie pasuje do tego, czego chcemy od MCU.



Zarządzanie pamięcią w C/C++ to ryzykowna gra, bardzo łatwo o błąd.

```
int t[10];
t[10] = 7;
cout << t[20] << endl;
```

//zagadka: co strasznego się wydarzy?

Kompilowanie kodu:

```
g++ --stack-check -W -Wall main.cpp -o xxx
```

Testy:

```
valgrind ./xxx
```

Może być konieczne zainstalowanie pakietu valgrind:

```
# apt-get install valgrind
```

Przykład działania tego narzędzia

```
#include <stdlib.h>
void f(void)
{
    int* x = malloc(10 * sizeof(int));
    x[10] = 0;        // problem 1: heap block overrun
}                     // problem 2: memory leak -- x not freed

int main(void)
{
    f();
    return 0;
}
```

```
int main()
{
    int* a = new int(33);
    *a = 7;
    return 0;
}
```

```
$ valgrind --leak-check=full ./Lab09a
==4060== HEAP SUMMARY:
==4060==     in use at exit: 4 bytes in 1 blocks
==4060==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==4060==
==4060== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==4060==    at 0x4C286E7: operator new(unsigned long) (vg_replace_malloc.c:287)
==4060==    by 0x400976: main (main.cpp:14)
==4060==
==4060== LEAK SUMMARY:
==4060==    definitely lost: 4 bytes in 1 blocks
==4060==    indirectly lost: 0 bytes in 0 blocks
==4060==      possibly lost: 0 bytes in 0 blocks
==4060==    still reachable: 0 bytes in 0 blocks
==4060==         suppressed: 0 bytes in 0 blocks
```

…I z tego wynikają główne problemy z C oraz C++ (tu w mniejszym stopniu).

➔ **MISRA** C:2023 (Motor Industry Software Reliability Association)

Zmienne

Podstawowe typy danych:

- nieokreślony: void
- wyliczeniowy: enum
- logiczny: bool (true/false albo TRUE/FALSE w C)
- całkowite: char, int
- zmiennoprzecinkowe: float, double
- wskaźnikowe: przez dodanie * (gwiazdki) po nazwie typu
- modyfikator: unsigned, signed, short, long,
- modyfikator: volatile

...lub lepiej: int32_t, uint8_t

Zadanie.

Janusz dostał 5 jabłek. 3 jabłka zjadł.
Pytanie: Ile jabłek zostało Januszowi?

Myślicie, że 2?
Guzik prawda! Nie wiadomo przecież, ile jabłek miał, zanim dostał 5 jabłek.
Morał: Zawsze zerujcie zmienne!

Nie do końca prawda – nie zawsze jest to potrzebne lub wskazane.

Zmienne w C/C++ ze względu na obszar pamięci i widoczność

- Globalne
  - Statyczne w module
  - Statyczne w funkcji
- Lokalne (w tym argumenty funkcji)
- Alokowane dynamicznie
- Rejestrowe
- Adresowane jawnie
- ~~Zmienne~~ const

W MCU możliwe (i częste) są odstępstwa od standardu ANSI C.

```
#ifdef __cplusplus
extern "C" {
#endif


//…kod pliku H


#ifdef __cplusplus
}     // extern "C"
#endif
```

→demonstracja: ELF map

```
#ifndef NANOMODBUS_H
#define NANOMODBUS_H


//…kod pliku H


#endif      //NANOMODBUS_H
```

→demonstracja: #include loop

Lub inaczej:

```
#pragma once
```

Kilka sposobów na stałe w C/C++

```c
typedef enum nmbs_error {
    // Library errors
    NMBS_ERROR_INVALID_REQUEST = -8,
    NMBS_ERROR_INVALID_UNIT_ID = -7,
} nmbs_error;
```

```c
const int NMBS_ERROR_INVALID_REQUEST = -8;
const int NMBS_ERROR_INVALID_UNIT_ID = -7;
```

```c
#define NMBS_ERROR_INVALID_REQUEST -8
#define NMBS_ERROR_INVALID_UNIT_ID -7
```

```
extern int zmienna;
```

```
int zmienna;
```

→deklaracja a definicja – demonstracja na żywo