Podstawy programownia (w języku C++)

Wstęp do programowania

Marek Marecki

Polsko-Japońska Akademia Technik Komputerowych

4 października 2021

Overview

Środek historii

...czyli 150 lat do CPU

1822 Difference Engine[1] – Charles Babbage

1837 Analytical Engine – Charles Babbage hardware[2] [3], a Ada Lovelace software

1941 Konrad Zuse – Z3, pierwszy programowalny komputer[4]

1944 Harvard Mark I – drugi programowalny komputer

1971 Intel 4004 – 4-bitowy mikroprocesor

W 2016 UK pozwoliło sprzedać ARM – czyli po raz drugi wypuścili z rąk ważny kawałek technologii.

[1] skonstruowany w londyńskim Science Museum w 1991

[2] zbudował prototyp CPU w 1871

[3] w 1906 jego syn, Henry Babbage, zbuduje kompletene CPU

[4] zniszczony podczas bombardowania Berlina przez Aliantów

Architektury CPU

1837 Analytical Engine

1978 x86

1985 ARM, MIPS

1991 PowerPC

2001 Itanium (VLIW; *failed*)

2003 Mill (VLIW; *in development*)

2010 RISC-V

Overview

von Neumann

1. CPU
2. RAM
3. pamięć masowa
4. I/O

Overview

A co to?

Sposób na wyrażenie swoich żądań względem maszyny.

Kontrakt z demonem – spełnia rozkazy *dokładnie tak jak są wypowiedziane*, bez oglądania się na *intencje* programisty.

A po co to komu?

`mov eax, 0x2a` (x86)
vs
`auto x = int{42};` (C++)

Dużo prostsze wydawanie maszynie skomplikowanych rozkazów, i łatwość zrozumienia znaczenia programu.
Automatyczna alokacja rejestrów i pamięci; automatyczne skoki; ergonomiczna semantyka.

Przenośność (ang. *portability*) programów między platformami.

Umowny podział

1. compiled *vs* interpreted (JIT?)
2. typing: static *vs* dynamic, strong *vs* weak
3. paradigm[5]: functional *vs* object-oriented *vs* structural *vs* prototype-based *vs* ...
4. rodziny: C-like (pochodne po języku ALGOL), ML-like, Lisp-like
5. "toy" *vs* "real"

C	compiled, static-weak typing, structural
C++	compiled, static-strong typing, multiparadigm
Smalltalk	interpreted, dynamic-strong, object-oriented
OCaml	compiled, static-strong, functional
Perl	interpreted, dynamic-weak, multiparadigm

[5] Seven Languages in Seven Weeks; Bruce A. Tate; ISBN-13: 978-1-934356-59-3

Overview

Co jest potrzebne w języku?

? (pytanie do sali)

Z punktu widzenia prostego człowieka

1. *control flow* – mechanizmy przepływu kontroli, czyli sterowania programem
2. *data structures* – reprezentacja struktur danych
3. *I/O* – zapis i odczyt danych, czyli sposób na interakcję ze światem zewnętrznym

Jackson Structured Programming

Control flow

Michael Jackson, 1975; Principles of Program Design

1. *sequence* – sekwencjonowanie, czyli ustalenie kolejności wykonywania operacji
2. *selection* (*alternative*) – wybór (alternatywa), czyli decyzja o podjęciu jednej z kilku różnych ścieżek wykonania
3. *iteration* – iteracja, czyli powtarzanie tych samych kroków n razy

Nadaje się do opisu algorytmów, ale nie za bardzo do czegoś więcej. Często tak jest z różnymi modelami – są wygodne w teorii, ale niezbyt praktyczne.

Warnier/Orr

Control flow

Jean-Dominique Warnier, 1976; Logical construction of programs
Kenneth Orr, 1977; Structured systems development

1. *recursion* – rekurencja, czyli sposób na zagnieżdżone wykonywanie operacji
2. *concurrency* – współbieżność, czyli sposób na wykonywanie kilku operacji "w tym samym czasie" (naprzemiennie na jednym procesorze, lub równolegle[6] na wielu)

Rekurencja i współbieżność są nieodłącznymi elementami programów, które działają w "prawdziwym świecie". Bez nich niemożliwe byłoby interaktywne używanie komputerów.

[6] ten wariant nazywa się *parallelism*, i czasem jest podawany obok współbieżności jako coś innego

Michael Scott

Control flow

Michael Lee Scott, 2000; Programming language pragmatics[7]

1. *procedural abstraction* – zbiór operacji opakowany w sposób umożliwiający ich wspólne wywołanie, w skrócie: funkcja
2. *nondeterminacy* – niedeterminizm, czyli sposób na zapewnienie losowości przy wyborze ściezki wykonania
3. *exceptions** – wyjątki, pozwalające na ”skok” kontroli w przypadku wystąpienia błędu

Funkcje i niedeterminizm zamykają bazowe mechanizmy, które służą kontroli przepływu.

[7] ISBN 1-55860-442-1

Podsumowanie

Control flow

1. *sequence*
2. *selection*
3. *iteration*
4. *recursion*
5. *concurrency*
6. *procedural abstraction*
7. *nondeterminism*
8. *exceptions**

Egzotyczne metody kontroli przepływu – *continuations*, *coroutines*.

Bit, nibble[8], byte, word, half-word, double-word, quad-word...

Data structures

Na początku było słowo

J 1,1-3

Bit - czyli wartość mogąca przechowywać 0 lub 1.
Podstawową jednostką danych obsługiwanych przez CPU jest "słowo" - sekwencja bitów o pojedynczego długości rejestru. Dla architektury x86-64 długość słowa to 64 bity.
Zapis i odczyt słowa w pamięci zazwyczaj są *operacjami atomowymi* co ma znaczenie dla programowania współbieżnego.

[8] połowa bajtu, czyli 4 bity

Liczby całkowite, ułamki, wartości logiczne

Data structures

Liczby całkowite - ze znakiem (`signed`), bez znaku (`unsigned`).
Liczby zmiennoprzecinkowe - pojedynczej precyzji, podwójnej precyzji.
Wartości logiczne - prawda, fałsz.

Znaki, napisy

Data structures

Znaki[9] - reprezentujące pojedynczy glif (literę, znak interpunkcyjny, itd.) lub symbol kontrolny.
Napisy - reprezentujące sekwencje znaków (np. ''`Hello, World!`'').

[9] kiedyś były najczęśniej szerokości 1 bajty (ASCII), ale obecnie, od upowszechnienia się standardu Unicode, są zazwyczaj zmiennej długości (UTF-8)

Lista, kolejka, stos, zbiór, drzewo, krotka...

Data structures

1. *list* – lista, czyli poszeregowana sekwencja wartości typu T, do których daje dostęp w dowolnej kolejności (ang. *random access*)
2. *queue* – kolejka, czyli poszeregowana sekwencja wartości typu T, do których daje dostęp na zasadzie *FIFO*
3. *stack* – stos, czyli poszeregowana sekwencja wartości typu T, do których daje dostęp na zasadzie *LIFO*
4. *set* – zbiór, nieposzeregowana kolekcja wartości typu T
5. *tree* – drzewo, często wykorzystywane do budowania "map" czyli struktur asocjacyjnych pozwalających na przechowanie wartości typu T pod kluczem typu K
6. *tuple* – krotka, czyli struktura danych zawierająca n pól typów T_0, T_1, ..., T_n

Podstawową złożoną strukturą danych jest *tablica*, czyli sekwencja n elementów typu T. Bazując na tablicach bajtów (czyli de facto surowych, wydzielonych obszarach pamięci) można zaimplementować wszystkie powyższe struktury danych.

Typy użytkownika

Data structures

enum wyliczenie, czyli zdefiniowany zbiór wartości, które dany typ może przechować (vide *sum type*)

struct struktura, czyli typ złożony z kilku pól różnych typów – może przechowywać wszystkie kombinacje wartości pól (vide *product type*)

Języki programowania często zapewniają programistom możliwość tworzenia własnych typów danych.

Wskaźniki i dynamiczna alokacja pamięci

Data structures

Implementacja wielu struktur danych (np. list lub napisów o zmiennej długości) byłaby niemożliwa bez wskaźników i dynamicznej alokacji pamięci.

Podsumowanie

Data structures

1. słowo (*word*), bajt (*byte*), bit
2. tablica (*array*)
3. typy użytkownika (*user-defined type – enum*, *struct*)
4. wskaźnik (*pointer*)
5. dynamiczna alokacja pamięci (*memory allocation*)
6. typy proste *vs* typy złożone

Operacje wejścia-wyjścia

I/O

Sposób na interakcję i wymianę danych ze "światem zewnętrznym", czyli wszystkim tym co dzieje się poza CPU i pamięcią operacyjną (RAM).

1. *I/O port*, *MMU* (ang. *memory management unit*), *memory-mapped I/O*
2. *file-descriptor*, *socket* (ten sam interfejs dla plików i połączeń w sieci; POSIX)
3. *memory-mapped file*

Zapisując i odczytując bajty da się obsłużyć każdy rodzaj urządzenia - monitor, klawiaturę, dysk, ramię robota, silnik, itd.