**Poročilo druge laboratorijske vaje**

Informacije in Kodi

Študent: Luka Melinc



Mentorja: asist. Dr. Klemen Grm, izr. prof. dr. Simon Dobrišek

Datum: 1.4.2023

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

​​

Uvod

Pri drugi laboratorijski vaji smo se ukvarjali s kodirnimi tabelami, dekodiranjem posameznih znakov v različnih kodirnih tabel in dekodiranje ter pretvarjanjem med različnimi tipi zapisa znakov.

Na hitro o kodnih tabelah

V tej laboratorijski vaji smo se ukvarjali s kodirnimi tabelami IBM-852, ISO-8859-2, Windows-1250, MacCE, UTF-8, UTF-16LE in UTF-16BE.

Kodirna tabela IBM-852 (OEM 852) se uporablja v sistemih DOS za pisanje centralno-evropskih jezikov, ki uporabljajo latinico (srbohrvaščina, poljšina, češčina, madžarščina, romunščina in slovenščina). ISO-8859-2 kodira 191 znakov latinične abecede, kjer vsakemu znaku pripada osembitna koda. Uporablja se za kodiranje 11 centralno-evropskih jezikov. Windows-1250 se uporablja na napravah z WindowsOS za jezike srednje in vzhodne evrope, ki uporabljajo za zapis latinico. Večinoma ga je zamenjal standard UTF-8. MacCE ali MacCentralEurope se uporablja na računalnikih z MacOS za srednje-evropske in jugovzhodne evropske jezike. UTF-8 je univerzalni sistem kodiranja znakov, ki omogoča predstavitev vseh znakov iz Unikodovega nabora. Glavna značilnost UTF-8 je njegova spremenljiva dolžina bajtov, pri čemer lahko uporabi od enega do štirih bajtov za kodiranje posameznega Unikodovega znaka. UTF-8 je najbolj razširjen način kodiranja pletnih strani in datotek. Medtem je UTF-16 zelo podoben UTF-8, le da UTF-16 kodira znake z uporabo enega ali dveh bitnih enot (2 ali 4 bajtov).

Prva naloga

V prvi nalogi smo se najprej lotili izpisa kodnih zamenjav šumnikov slovenske abecede. Poiskali smo zamenjave tako za velike kot male črke šumnikov, vse skupaj šest znakov. To smo storili za sledeče kodirne tabele: IBM-852, ISO-8859-2, Windows-1250. MacCentralEurope ali MacCE, UTF-16LE in 16BE, UTF-8. Za vsako kodirno tabelo smo izpisali tudi vrednost znaka z decimalnim, heksadecimalnim in binarnim zapisom.

Najprej smo definirali array z vsemi sedmimi kodirnimi tabelami, ki smo jih uporabili ter array z znaki, ki smo jih kodirali (trije šumniki pisano z malimi in velikimi črkami – 6 znakov).

Nato smo definirali funkcijo za pretvarjanje iz desetiških vrednosti v heksadecimalne vrednosti.

A computer screen shot of a code

Description automatically generated

V tej funkciji smo najprej preverili, če je vrednost enaka 0, nato smo postavili nabor heksadecimalnih znakov, od 0 do F, s katerimi se lahko »gradi« heksadecimalna števila. V while zanki se računa vrednost ostanka pri deljenju s 16, ta ostanek ustreza indeksu iz niza heksadecimalnih znakov, ta znak pa se doda na začetek heksadecimalnega niza, ki ga gradimo. To se izvaja, dokler je vrednost večja od 0.

Nato smo napisali funkcijo, ki pretvarja iz decimalnega zapisa v binarni zapis števila.

A computer screen with green and blue text

Description automatically generated

Tudi v tej funckiji smo najprej preverili, ali je vrednost, ki jo pretvarjamo, enaka 0. Postavili smo spremenljivko bin\_vrednost, v katero smo vnašali enice in ničle binarnega zapisa. Vrednost smo vstrajno delili z dva ter spremljali, ali ob delitvi nastane ostanek, ali je delitev z dve brez ostanka. V primeru, da ostanek obstaja, smo v binarni zapis dodali 0, če pa ostanka ni bilo, pa smo dodali enico. Deljenje smo izvajali, dokler je bila vrednost večja kot 0.

V tretjem delu kode smo iz tabele prebrali znak, katerega zapis pretvarjamo in prevedli njegov zapis za vsako kodirno tabelo v desetiško, heksadecimalno in binarno zapis.



S for zanko smo iterirali najprej čez vse tipe kodiranja, znotraj vsakega tipa kodiranja pa smo iterirali še skozi vse znake v tabeli znakov. Za vsak znak smo s funckijo int.from\_bytes() sprejeli določeno zaporedje bajtov ter ga pretvorili v celo število. Prvi argument je zaporedje bajtov, ki je pridobljeno s kodiranjem znaka, drugi argument pa govori o temu, kateri bit je najbolj pomemben (most significant bit). Nato se za pretvorbo v heksadecimalni in binarni zapis uporabi še prej omenjeni funkciji ter izpiše rezultate v terminalu. Rezultati programa so zledeči:

A screen shot of a computer

Description automatically generated

Druga naloga

Pri drugi nalogi smo ustvarili program za branje Unicode kodnih točk, s katerim smo prebrali vhodno datoteko, dekodirali v Unicode znake ter jih shranili kot UTF-8 kodirano besedilo.

Pri drugi nalogi sem uporabil tudi funkcije iz prve naloge. UTF-8 kodiranje je neenakomeren kod, zato je potrebna struktura predpon, ki sem jo izvedel na sledeč način:

Tako smo posameznim bajtom dodali bite glede na tabelo, podano v navodilih.

A computer screen shot of a program code

Description automatically generated

Funkciji kot argument podamo vrednost, preverimo dolžino vrednosti in glede na število bajtvo, ki jih potrebujemo za zapis vhodne vrednosti posameznim bajtom dodajamo ustrezne predpone.

V glavnem delu kode smo odprli vhodno datoteko, jo prebrali, odstranili presledke ter besedilo razdelili na slovar znakovnih nizov. Iz prebrane datoteke smo pretovrili kodna mesta v številske podatke. Kodna mesta smo tako pretvorili v Unicode znake ter dekodirali vhodne številske znake ter jih zapisali v izhodno datoteko. Izkaže se, da je bilo vhodno besedilo stavek iz Wikipedije v štirih različnih pisavah o kodni tabeli UTF-8. V kodi izpišemo tudi tabelo unikatnih znakov in njene zamenjave, ki so shranjene v spremenljivki *unikatni\_znaki*. Dekodirano besedilo se shrani v datoteko *besedilo.txt*.

