# Tema 8

## Bitwise i jednostruko spregnuta lista

# Operacije nad bitovima

- Grupa operatora čije je delovanje definisano nad bitovima
- Operandi mogu isključivo biti celobrojni tipovi
  - `int` (i sa varijacijama `long` i `short`) i `char` tipovi, bilo u `signed` ili `unsigned` obliku

Bitwise operatori

| Operator | Značenje |
|---|---|
| & | logičko "I" bit po bit |
| \| | logičko "ILI" bit po bit |
| ^ | ekskluzivno "ILI" bit po bit |
| << | levi pomak (left shift) |
| >> | desni pomak (right shift) |
| ~ | prvi komplement (negacija bit po bit) |

- Varijante sa bočnim efektom: &=, |=, ^=, <<=, >>= i ~=

# Brojevni sistemi

- Bitwise operacije rade na nivou bita
- Binarni brojni sistem, heksadecimalni zarad kraćeg (i čitljivijeg) zapisa

Vrednosti u brojnim sistemima

| Decimalni | Heksadecimalni | Binarni |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |

| | | |
|---|---|---|
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| 10 | A | 1010 |
| 11 | B | 1011 |
| 12 | C | 1100 |
| 13 | D | 1101 |
| 14 | E | 1110 |
| 15 | F | 1111 |

- Konvertovati sledeće vrednosti iz binarnog u heksadecimalni brojni sistem:
  - `0100 0111 0101 0111`
  - `1101 0100 1010 1001`
- Konvertovati sledeće vrednosti iz heksadecimalnog u binarni brojni sistem:
  - `0x8000`
  - `0x1234`

# Primer 1

Ispis broja u binarnom obliku

```c
#include <stdio.h>

#define MASK 0x8000

void print_binary(short);

int main()
{
    short dec;

    printf("Unesite broj: ");
    scanf("%hd", &dec);

    print_binary(dec);

    return 0;
}
```

```
void print_binary(short dec)
{
    int i;

    for(i = 1;i <= sizeof(short) * 8;i++)
    {
        printf("%d", (dec & MASK) != 0);
        dec <<= 1;

        if(i % 4 == 0)
        {
            printf(" ");
        }
    }

    printf("\n");
}
```

# Primer 2

Učitati podatke o zaposlenima u preduzeću iz ulaznog fajla u sledeći dinamički alocirani niz struktura, gde se jedan element sastoji od sledećih polja:

- Ime (jedna reč, maksimalno 20 karaktera)
- Prezime (jedna reč, maksimalno 20 karaktera)
- Dozvoljeni sektori pristupa (jedan bajt, tip `unsigned char`)

Na osnovu zadatog argumenta komandne linije (karakter, A, B ili C), u izlaznu datoteku `dozvoljen_pristup.txt`, ispisati zaposlene kojima je pristup dozvoljen u zadatom sektoru. Sektori su obeleženi na sledeći način:

```
0x5 -> 0 1 0 1
       C B A
```

U ovom primeru, zaposleni ima pristup sektorima `A` i `C` (vrednost bita na pozicijama je 1).

Primer ulazne datoteke:

```
Milan      Sekulic    0x1
Aleksandar Stosic     0x3
Milica     Strajnic   0x6
Goran      Maletic    0x5
Jelena     Stanivukov 0x7
```

Očekivana izlazna datoteka za poziv: `./pristup zaposleni.txt A`

```
Milan Sekulic
Aleksandar Stosic
Goran Maletic
Jelena Stanivukov
```

- U slučaju da nije moguće otvoriti ulaznu datoteku, izaći sa kodom greške 2
- U slučaju da nije moguće otvoriti izlaznu datoteku, izaći sa kodom greške 3

```c
#include <stdio.h>
#include <stdlib.h>

#define ARRAY_SIZE 5
#define MAX_STRING 21

typedef struct zaposleni_st
{
    char ime[MAX_STRING];
    char prezime[MAX_STRING];
    unsigned char pristup_sektorima;
} ZAPOSLENI;

FILE *safe_fopen(char *, char *, int);
void ucitaj(FILE *, ZAPOSLENI *, int *);
void ispisi_sa_pristupom(FILE *, ZAPOSLENI *, unsigned char, int);
```

```c
int main(int argc, char **argv)
{
    ZAPOSLENI *zaposleni;
    int n;

    if(argc != 3)
    {
        printf("Primer poziva: %s zaposleni.txt A\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    zaposleni = (ZAPOSLENI *)malloc(ARRAY_SIZE * sizeof(ZAPOSLENI));

    if(zaposleni == NULL)
    {
        printf("Nije moguce zauzeti memoriju!\n");
        exit(EXIT_FAILURE);
    }

    FILE *ulazna = safe_fopen(argv[1], "r", 2);
    ucitaj(ulazna, zaposleni, &n);
    fclose(ulazna);
```

```
    char *sektor_str = argv[2];
    unsigned char sektor;

    switch(sektor_str[0])
    {
    case 'A': sektor = 0x1; break;
    case 'B': sektor = 0x2; break;
    case 'C': sektor = 0x4; break;
    default:
        printf("Ne postoji zadati sektor!\n");
        exit(EXIT_FAILURE);
    }


    FILE *izlazna = safe_fopen("dozvoljen_pristup.txt", "w", 3);
    ispisi_sa_pristupom(izlazna, zaposleni, sektor, n);
    fclose(izlazna);

    free(zaposleni);

    return EXIT_SUCCESS;
}
```

```c
FILE *safe_fopen(char *ime, char *rezim, int kod_greske)
{
    FILE *fp = fopen(ime, rezim);

    if(fp == NULL)
    {
        printf("Nije moguce otvoriti datoteku %s!\n", ime);
        exit(kod_greske);
    }

    return fp;
}
```

```c
void ucitaj(FILE *ulazna, ZAPOSLENI *zaposleni, int *pn)
{
    int i = 0;

    // hh - jednobajtna, x - heksadecimalna vrednost
    while(fscanf(ulazna, "%s %s %hhx",
                 zaposleni[i].ime,
                 zaposleni[i].prezime,
                 &zaposleni[i].pristup_sektorima
          ) != EOF)
    {
        i++;
    }

    *pn = i;
}
```

```c
void ispisi_sa_pristupom(FILE *izlazna, ZAPOSLENI *zaposleni,
                         unsigned char sektor, int n)
{
    int i;

    for(i = 0;i < n;i++)
    {
        // Ukoliko zaposleni ima odgovarajuci bit
        // u pristupu_sektorima postavljen na 1,
        // rezultat operacije bice razlicit od 0,
        // sto je logicki tacna vrednost
        if(zaposleni[i].pristup_sektorima & sektor)
        {
            fprintf(izlazna, "%s %s\n",
                    zaposleni[i].ime, zaposleni[i].prezime);
        }
    }
}
```

# Zadatak 1

Doraditi `Primer 2`:

a. Da se niz proširi za veličinu direktive `ARRAY_SIZE` svaki put kada se dostigne njegova popunjenost
   - Niz treba proširiti kada `i` dostigne vrednost koja je deljiva sa `ARRAY_SIZE`
   - Za proširenje niza, koristiti funkciju `realloc`

b. Još jednim sektorom `D`, kome će biti namenjena binarna cifra na prvoj višoj poziciji pored sektora `C`. U ulaznu datoteku dodati zaposlenog koji ima pristup sektoru i ispitati rešenje.

Dva načina da se reši `Zadatak 1` pod `a.`:

1. Proširiti funkciju `ucitaj` tako da se učitavanje podataka iz datoteke prekine kad se dostigne kapacitet niza

   - Funkcija `ucitaj` vraća vrednost 1, ukoliko je kapacitet niza dostignut, 0 ukoliko su svi podaci učitani
   - Na osnovu povratne vrednosti funkcije `ucitaj`, proširiti kapacitet niza i pozvati ponovo funkciju `ucitaj` da nastavi sa učitavanjem podataka

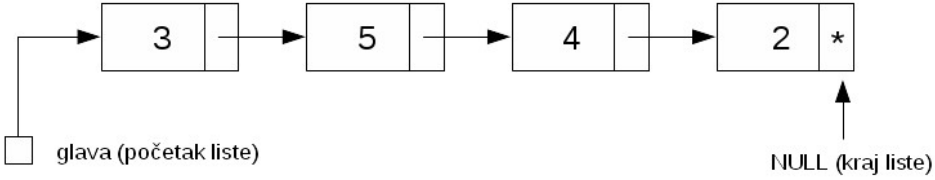
2. Proširiti funkciju `ucitaj` tako da se kapacitet niza proširi tokom učitavanja podataka iz datoteke

   - Pošto funkcija `realloc` menja adresu zauzetog memorijskog prostora, u slučaju poziva unutar funkcije `ucitaj`, potrebno je preneti pokazivač na niz po referenci

```
void ucitaj(FILE *pin, ZAPOSLENI **pzaposleni, int *pn);
```

# Jednostruko spregnuta lista

- Dinamička struktura podataka
- Element liste (čvor) se sastoji iz dva polja
  - Podatak (informacioni deo čvora liste, celobrojna vrednost, karakter, struktura itd.)
  - Pokazivač na sledeći element liste (sadrži adresu sledećeg elementa liste)
- Pristup elementima vrši se preko pokazivača na prvi element liste - glave liste



# Operacije nad listom

1. Inicijalizacija liste
2. Unos novog elementa na kraj liste
3. Listanje liste (prikaz svih elemenata iz liste)
4. Traženje čvora sa zadatom vrednošću elementa u listi
5. Brisanje čvora sa zadatom vrednošću elementa iz liste
6. Brisanje cele liste (oslobađanje memorije)

# Element (čvor) liste

```
typedef struct cvor_st
{
    int el;
    struct cvor_st *sledeci;
} CVOR;
```

- Podatak (informacioni deo čvora) je celobrojna vrednost `el`
- `sledeci` sadrži adresu sledećeg čvora
    - Ukoliko je vrednost `NULL`, u pitanju je poslednji čvor u listi

# Inicijalizacija liste

```
void inicijalizacija(CVOR **pglava)
{
    *pglava = NULL;
}
```

- Glava liste prenosi se po referenci
  - Pošto će se adresa pokazivača `glava` iz main funkcije menjati u funkciji `inicijalizacija`, funkciji je potrebno proslediti adresu pokazivačke promenljive `glava`, otud dve zvezdice (**)
- Kreiranje prazne liste bez elemenata
  - Lista je prazna ukoliko je vrednost glave `NULL`
- glava liste dobija vrednost `NULL`

# Pravljenje novog čvora

```
CVOR *napravi_cvor(int el)
{
    CVOR *novi = (CVOR *)malloc(sizeof(CVOR));

    if (novi == NULL)
    {
        printf("Nije moguce zauzeti memoriju!\n");
        exit(EXIT_FAILURE);
    }

    novi->el = el;
    novi->sledeci = NULL;

    return novi;
}
```

- Dinamička alokacija memorije za novi čvor pomoću funkcije `malloc`
- Popunjavanje informacionog dela čvora
- Postavljanje vrednosti `sledeci` na `NULL`

# Unos novog elementa na kraj liste

```
void dodaj_na_kraj(CVOR **pglava, CVOR *novi)
{
    if (*pglava == NULL)
    {
        *pglava = novi;
    }
    else
    {
        CVOR *tekuci = *pglava;
        while (tekuci->sledeci != NULL)
        {
            tekuci = tekuci->sledeci;
        }
        tekuci->sledeci = novi;
    }
}
```

# Unos novog elementa na kraj liste

- glava liste se prenosi po referenci (**), zbog slučaja kad je lista prazna
- Pomoću `while` petlje ide se od glave liste do zadnjeg čvora
- Za vrednost `sledeci` trenutno aktuelnog zadnjeg čvora postaviti adresu novog čvora umesto `NULL`

# Listanje (ispis) liste

```
void ispisi_listu(CVOR *glava)
{
    CVOR *tekuci = glava;

    printf("[");
    while (tekuci != NULL)
    {
        printf("%d", tekuci->el);
        if (tekuci->sledeci != NULL)
        {
            printf(", ");
        }

        tekuci = tekuci->sledeci;
    }
    printf("]\n");
}
```

- Pomoću `while` petlje prolazi se kroz listu od prvog do zadnjeg elementa i ispisuje se njihova vrednost

# Traženje čvora

```
CVOR *nadji_cvor(CVOR *glava, int el)
{
    CVOR *tekuci = glava, *nadjen = NULL;

    while (tekuci != NULL)
    {
        if (tekuci->el == el)
        {
            nadjen = tekuci;
            break;
        }
        tekuci = tekuci->sledeci;
    }
    return nadjen;
}
```

# Traženje čvora

- Pomoću `while` petlje prolazi se kroz listu i traži se čvor koji sadrži prosleđenu vrednost
- Ukoliko je pronađen element, traženje će stati (naredba `break`) i funkcija će vratiti pokazivač na nađeni element
- Funkcija vraća `NULL` pokazivač ako nema traženog elementa

# Brisanje liste

```
void obrisi_listu(CVOR **pglava)
{
    CVOR *tmp;

    while (*pglava != NULL)
    {
        tmp = *pglava;

        *pglava = (*pglava)->sledeci;
        tmp->sledeci = NULL;
        free(tmp);
    }
}
```

# Brisanje liste

- Glava liste prenosi se po referenci (biće menjana)
- Pokazivač ka trenutnoj glavi liste čuva se u pokazivaču `tmp`
- Glava liste postaje sledeći element u listi
- Čvor čija adresa je u `tmp` raskida vezu sa ostatkom liste (sledeći se postavlja na `NULL`) i oslobađa se njegova dinamički alocirana memorija pomoću funkcije `free`

# Test program

```
int main()
{
    CVOR *glava;

    inicijalizacija(&glava);

    dodaj_na_kraj(&glava, napravi_cvor(1));
    dodaj_na_kraj(&glava, napravi_cvor(2));
    dodaj_na_kraj(&glava, napravi_cvor(3));
    dodaj_na_kraj(&glava, napravi_cvor(4));

    ispisi_listu(glava);
```

```
    obrisi_listu(&glava);

    ispisi_listu(glava);

    return EXIT_SUCCESS;
}
```

# Provera postojanja curenja memorije

- Uz pomoć programa Valgrind
- Pokretanje: `valgrind <ime-programa>`
  - Primer: `valgrind ./a.out`

Testirati prethodni primer jednostruko spregnute liste pomoću programa `Valgrind`. Porediti ispise programa pre i posle zakomentarisanog poziva funkcije `obrisi_listu`.

- Napomena: ukoliko nemate `Valgrind` na vašem sistemu, instalira se na sledeći način: `sudo apt-get -y install valgrind`

# Zadatak 1

U jednostruko spregnutoj listi nalaze se ocene koje je student dobio tokom godine. Izračunati prosek i rezultat ispisati na standardnom izlazu (ekranu).

```
int main()
{
    char broj_indeksa[] = "ee300-2020";
    char ime_studenta[] = "Jovan Jovanovic";
    CVOR *glava;

    inicijalizacija(&glava);

    dodaj_na_kraj(&glava, napravi_cvor(8));
    dodaj_na_kraj(&glava, napravi_cvor(7));
    dodaj_na_kraj(&glava, napravi_cvor(9));
    dodaj_na_kraj(&glava, napravi_cvor(10));
    dodaj_na_kraj(&glava, napravi_cvor(6));
    dodaj_na_kraj(&glava, napravi_cvor(8));
    dodaj_na_kraj(&glava, napravi_cvor(9));
    dodaj_na_kraj(&glava, napravi_cvor(8));
```

```
    printf("Student %s, sa brojem indeksa %s, ima prosek %.2lf\n",
           ime_studenta, broj_indeksa, prosek_studenta(glava));

    obrisi_listu(&glava);

    return EXIT_SUCCESS;
}
```

Očekivani ispis programa:

```
Student Jovan Jovanovic, sa brojem indeksa ee300-2020, ima prosek 8.12
```

# Zadatak 2

Napisati jednostruko spregnutu listu, koja sadrži stringove maksimalne dužine 30 karaktera. Popuniti listu sa određenim stringovima (može biti zadato unapred, bez potrebe za unosom), zatim putem tastature uneti karakter. Izračunati srednju vrednost udela zadatog karaktera na nivou cele liste. Ispisati rezultat zaokružen na dve decimale.

Primer:

Zadato slovo a i reči u listi:

Pera -> 0.25

Mika -> 0.25

Laza -> 0.5

Srednja vrednost udela: 0.33 (zaokruženo na dve decimale)