# Strukture podataka i algoritmi 1

## I test – max 13 poena

11.04.2024.

| Ime i prezime | Broj indeksa | Broj poena |
|---|---|---|
| | | |

**Napomena:** jedino u prvom zadatku može doći do kompajlerske greške.

1. Napisati rezultate sledećih kodova (2 poena, svaki primer po 0.5 poena).

a)

```
int a = 10, b = 20;
int *pa = &a;
*pa = 30;
 a++;
pa = &b;
printf("%d %d %d\n", a, *pa, b);
```

31 20 20

b)

```
putchar(putchar('r'));
```

rr

c)

```
#define IZRAZ (x) (x – 5 + x / 2)

int main() {
   int x = 10;
   printf("%d\n", IZRAZ(4 + x));
   return 0;
}
```

18

d)

```
int a = 10, b = 20;
int *pa = &a * 20, *pb = &a;
*pa = a + *pb;
*pb = b;
*pb = *pa + 3;
printf("%d %d %d %d\n", a, b, *pa,
*pb);
```

Compile error

2. a) Šta je pokazivač i kolika je veličina jednog pokazivača u bajtovima na 64-bit sistemu? (0.5 poena)

Pokazivač je promenljiva čija je vrednost adresa neke druge promenljive. 8 bajtova.

b) Razlika između malloc i calloc. Do koje greške dolazi ako proces pristupa delu memorije koji nije njegov? (0.5 poena)

Malloc prima broj bajtova, dok calloc prima broj elemenata i veličinu jednog elementa i inicijalizuje vrednosti na 0. Segmentation fault.

3. Šta je rezultat sledećeg koda? (1 poen)

```
#include <stdio.h>
int f(char *str, int i, int l) {
   if (i >= l / 2)
      return 0;
   return ((str[i] == str[l-i-1]) ? 1 : 0) + f(str, i+1, l);
}
int main() {
   char *str = "abcgtcna";
   printf("%d\n", f(str, 0, 8));
   return 0;
}
```

2

4. Šta je rezultat sledećeg koda? (1.5 poen)

```
int main() {
   int a = 10, b = 3, c = 12;
   float d = 2;

   printf("%d\n", (int)(b/d) | a);
   printf("%f\n", (float)((b+c)/a));
   printf("%d\n", b | c);
   printf("%d\n", b ^ c);
   printf("%d\n", (b+c) & ~(-3));
   return 0;
}
```

```
11
1.000
15
15
2
```

5. Šta je rezultat sledećeg koda? (1.5 poen)

```
#include <stdio.h>
int f(int a) {
   static int b = 10;
   b--;
   int c = (a-- > --b)? a : ((b++ < a++)? b : 100);
   b += 5;
   return c;
}
int main() {
   printf("%d\n", f(10));
   printf("%d\n", f(15));
```

```
    printf("%d\n", f(20));
    return 0;
}
```

```
9
14
19
```

6. Napisati program koji računa zbir dva binarna broja. Brojevi su zapisani kao nizovi iste dužine. Zabranjeno je korišćenje aritmetičkih operatora kao što su +, - , *, /, %, ... Obavezno je korišćenje bitovskih operatora &, |, ^, ~, ... (2 poena)

Primer: [1,0,0,1] + [0,1,0,1] = [0,1,1,1,0]

```
#include <stdio.h>
// Funkcija za dodavanje dva binarna broja predstavljena kao nizovi
void add_binary_numbers(int bin1[], int bin2[], int size) {
    int carry = 0;
    int result[size + 1]; // Da sačuvamo rezultat (jedan dodatni digit za prenos)

    for (int i = size - 1; i >= 0; i--) {
        // Izračunajte zbir trenutnih bitova i prenosa
        int sum = bin1[i] ^ bin2[i] ^ carry;

        // Izračunajte prenos za narednu iteraciju
        carry = (bin1[i] & bin2[i]) | (carry & (bin1[i] ^ bin2[i]));

        // Sačuvajte zbir u rezultat
        result[i + 1] = sum;
```

```
    }

    // Ako je ostao prenos, dodajte ga na najznačajniji bit
    result[0] = carry;

    // Ispis rezultata
    printf("Zbir binarnih brojeva: ");
    for (int i = 0; i < size + 1; i++) {
        printf("%d", result[i]);
    }
    printf("\n");
}

int main() {
    int bin1[] = {1, 0, 0, 1};
    int bin2[] = {0, 1, 0, 1};
    int size = sizeof(bin1) / sizeof(bin1[0]);
    add_binary_numbers(bin1, bin2, size);
    return 0;
}
```

7. Napisati program kojim se štampaju svi četvorocifreni brojevi koji su "dobri". Broj je dobar ako je cifra jedinica paran broj, cifra desetica neparan broj, cifra stotina paran broj, ... Npr. brojevi 1234 i 7836 su dobri, dok su brojevi 1359 i 2143 loši. Napomena: Provera da li je neki broj stepen broja "dobar" vrši funkcija **proveri,** koja ima sledeći potpis **int proveri(int n, int k)**, koja **mora** biti **rekurzivna** i vraća 1 ukoliko je broj "dobar" ili 0 ukoliko nije. Funkcija mora da radi za bilo koji broj, a ne samo za četvorocifrene brojeve. (2 poena).

```c
#include <stdio.h>

// Funkcija za proveru da li je broj "dobar"
int proveri(int n, int k) {

    // Base case: ako je nula, broj je "dobar"
    if (n == 0) {
        return 1;
    }

    // Izdvojimo poslednju cifru
    int poslednja_cifra = n % 10;

    // Proverimo da li poslednja cifra zadovoljava uslov
    if ((k % 2 == 0 && poslednja_cifra % 2 == 0) || (k % 2 != 0 && poslednja_cifra % 2 != 0)) {
        // Rekurzivno proverimo preostale cifre
        return proveri(n / 10, k + 1);
    } else {
        return 0;
    }
}

int main() {

    // Ispis svih "dobrih" četvorocifrenih brojeva
    for (int num = 1000; num < 10000; num++) {
        if (proveri(num, 0)) {
            printf("%d\n", num);
        }
    }

    return 0;
}
```

8. Napisati program koji dodaje brojeve na stek i nakon dodavanja ih skida sa steka i ispisuje. Program **mora** da sadrži dve funkcije za dodavanje na stek i skidanje sa steka. Prilikom skidanja sa steka se vraća vrednost koja je skinuta. Za implementaciju steka je potrebno koristiti dinamički alocirani niz čija je početna veličina 10 int-a. Ukoliko se pokuša dodavanje na pun stek, potrebno je proširiti memoriju steka za 10. Program implementirati bez korišćenja operatora [] prilikom pristupa članovima niza. Potpisi funkcija: void push(int **stack, int stack_size, int stack_capacity), int pop (int *stack, int *stack_size). stack_size je trenutan broj elemenata na steku, a stack_capacity je maksimalna veličina steka. (2 poena)

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

```
// Funkcija za inicijalizaciju steka
void initialize_stack(int** stack_data, int* stack_size, int* stack_capacity) {
   *stack_data = (int*)malloc(10 * sizeof(int)); // Početna veličina steka je 10
   if (*stack_data == NULL) {
      printf("Greška pri alokaciji memorije za stek.\n");
      exit(1);
   }
   *stack_size = 0;
   *stack_capacity = 10;
}

// Funkcija za dodavanje elementa na stek
void push(int** stack_data, int* stack_size, int* stack_capacity, int value) {
   if (*stack_size == *stack_capacity) {
      // Ako je stek pun, proširimo memoriju za još 10 elemenata
      *stack_capacity += 10;
      *stack_data = (int*)realloc(*stack_data, *stack_capacity * sizeof(int));
      if (*stack_data == NULL) {
         printf("Greška pri realokaciji memorije za niz steka.\n");
         exit(1);
      }
   }
   *(*stack_data + *stack_size) = value;
   (*stack_size)++;
}

// Funkcija za skidanje elementa sa steka
int pop(int* stack_data, int* stack_size) {
   if (*stack_size == 0) {
      printf("Stek je prazan.\n");
      exit(1);
   }
   (*stack_size)--;
   return *(stack_data + *stack_size);
}

// Funkcija za oslobađanje memorije steka
void destroy_stack(int* stack_data) {
   free(stack_data);
}

int main() {
   int* stack_data;
   int stack_size;
   int stack_capacity;
```

```
    initialize_stack(&stack_data, &stack_size, &stack_capacity);

    // Dodajemo brojeve na stek
    push(&stack_data, &stack_size, &stack_capacity, 10);
    push(&stack_data, &stack_size, &stack_capacity, 20);
    push(&stack_data, &stack_size, &stack_capacity, 30);

    // Skidamo brojeve sa steka i ispisujemo ih
    printf("Skidani brojevi sa steka:\n");
    while (stack_size > 0) {
        printf("%d\n", pop(stack_data, &stack_size));
    }

    // Oslobađamo memoriju steka
    destroy_stack(stack_data);

    return 0;
}
```