Izrada interaktivnih zadataka iz programiranja koji provjeravaju znanje o naredbama grananja i petlji

Bivol, Antonia

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Antonia Bivol

IZRADA INTERAKTIVNIH ZADATAKA IZ PROGRAMIRANJA KOJI PROVJERAVAJU ZNANJE O NAREDBAMA GRANANJA I PETLJI

Diplomski rad

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO-MATEMATIČKI FAKULTET

FIZIČKI ODSJEK

INTEGRIRANI PREDDIPLOMSKI I DIPLOMSKI SVEUČILIŠNI STUDIJ
FIZIKA I INFORMATIKA; SMJER: NASTAVNIČKI

Antonia Bivol

Diplomski rad

Izrada interaktivnih zadataka iz programiranja koji provjeravaju znanje o naredbama grananja i petlji

Voditelj diplomskog rada: izv.prof.dr.sc. Maro Cvitan

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____

Zagreb, 2024.

Zahvaljujem svom mentoru, izv. prof. dr. sc. Maru Cvitanu na savjetima i strpljenju tijekom pisanja ovog rada.

Također, želim zahvaliti svojoj obitelji i prijateljima na podršci i motivaciji tijekom cijelog obrazovanja.

Na kraju zahvaljujem Molly i Rivi što su uvijek bile uz mene.

„Long story short, I survived.“ – Taylor Swift

Sažetak

Cilj rada je unaprijediti metode podučavanja i provjere znanja određenih aspekata programskih jezika. Rad se fokusira na petlje i grananja u programskom jeziku C. Dan pregled sintakse petlji i grananja u programskom jeziku C kao i opis metoda podučavanja koje uključuju programiranje u paru i programiranje uživo. Opisana je specifična metoda provjere znanja petlji i grananja u kojoj je potrebno reproducirati uzorak u zadanoj veličini. Opisan je sustavan način rješavanja takvih interaktivnih zadataka. Analizirane su tipične pogreške studenata. Opisan je sustavni pristup izradi uzoraka prihvatljivih za provjere znanja takvim zadacima. Predloženo je nekoliko karakteristika uzoraka koje omogućuju procjenu težine takvog zadataka. Osmišljeno je i priloženo više od 40 zadataka ispisa uzoraka prikladnih za vježbu i provjere znanja.

Ključne riječi: programski jezik C, grananje, for petlja, interaktivni zadaci, programiranje uživo, zadaci ispisa uzoraka, procjena težine zadataka

Development of interactive programming exercises testing branching and looping statement knowledge

Abstract

The purpose of the thesis is to improve the methods of teaching and testing knowledge of certain aspects of programming languages. The paper focuses on loops and branching in the C programming language. An overview of the syntax of loops and branching in the C programming language is given, as well as a description of teaching methods that include pair programming and live coding. A specific method of checking the knowledge of loops and branching is described, in which it is necessary to reproduce the letter-like figure with a specified dimension. A systematic way of solving such interactive coding tasks is described. Typical student errors were analyzed. A systematic approach to creating figures acceptable for testing knowledge of such tasks is described. Several characteristics of the figures have been proposed that allow the assessment of the difficulty of such coding tasks. More than 40 figures suitable for practice and testing have been designed and attached.

Keywords: programming language C, branching, for loops, interactive tasks, live coding, sample printing tasks, task difficulty evaluation

Sadržaj

1. Uvod

Kod studenata u današnjem sustavu visokog obrazovanja u području programiranja mogu se javiti izazovi u razumijevanju i korištenju nekih osnovnih građevnih jedinica programa u programskom jeziku C što može otežati daljnje savladavanje materije u idućim fazama stjecanja znanja iz programiranja. Stoga je namjera ovog rada prikazati interaktivan način na koji studenti mogu razvijati svoje znanje i unaprijediti razumijevanje odabranih osnovnih koncepata - operatora, naredbi grananja i naredbi ponavljanja kroz zadatke ispisa uzoraka. To će u ovom radu biti izvedeno kroz osmišljavanje i analiziranje velikog broja interaktivnih zadataka ispisa uzoraka, a istovremeno će rad ponuditi novu klasifikaciju zadataka prema težini koja je nastala na temelju dubinske evaluacije spomenutih zadataka i subjektivne procjene njihovih težina. Rad je strukturiran u pet poglavlja i započinje pregledom programskog jezika C, njegove svrhovitosti u učenju te osnovnih koncepata koji se koriste pri kreiranju i pokretanju programa.

Treće poglavlje pruža pogled u dosadašnja istraživanja o poteškoćama u razumijevanju for petlja i grananja, što se može prevladati raznim metodama podučavanja kao što je programiranje uživo.

U idućem poglavlju pobliže se razmatra korištenje petlji i grananja za ispis uzoraka. Uz to, naglasak je stavljen na kreiranje i analizu zadataka, konkretno, opisan je proces osmišljavanja zadataka ispisa uzoraka te su prikazane analize primjera različitih težina i čestih pogrešaka.

Fokus petog poglavlja je na klasifikaciji zadataka za ispis prema težini na temelju subjektivne procjene.

Konačno, zaključno poglavlje sumira doprinos ovog rada kroz analizu čestih pogrešaka u studentskim rješenjima, koja pruža uvid u dijelove gdje studenti nailaze na poteškoće u razumijevanju osnovnih koncepata. Prema tim uvidima mogu se modelirati smjernice za buduća predavanja i ispite znanja.

2. Pregled programskog jezika C

2.1 O svrhovitosti učenja programskog jezika C

Programski jezik C dizajniran je za stvaranje malih, brzih programa. Njegova razina [1] je niža od većine drugih programskih jezika, što znači da njime stvaramo kod koji je puno bliži onome što računala stvarno razumiju. Smisleno je pitati se je li učenje programskog jezika C vrijedno truda. Naime, stjecanje znanja u C-u omogućava dublje razumijevanje značajki jezika kao što su C++, Java, C# i drugih jezika temeljenih na C-u. Studenti koji prvo nauče jedan od ovih jezika često ne ovladaju osnovnim značajkama koje potječu iz C-a [2]. Učenje C-a pomaže razumjeti mnoge temeljne aspekte arhitekture operativnog sustava, poput pokazivača i rada s memorijskim lokacijama.

2.2 Osnovne varijable i njihovi tipovi

U programu se primarno koriste varijable i konstante kao osnovni oblici podataka. Varijable su memorijske lokacije koje sadrže promjenjive vrijednosti i one se mogu mijenjati tijekom izvršavanja programa, dok konstante imaju fiksne vrijednosti koje se ne mijenjaju [2]. Korištenje varijabli i konstanti omogućuje efikasno upravljanje podacima i olakšava čitljivost koda. Na početku programa deklariramo varijable i svaka varijabla mora imati definirani tip [2] koji specificira koju vrstu podataka će sadržavati.

U programskom jeziku C postoji nekoliko osnovnih tipova [2] varijabli:

- *char* – jedan znak iz skupa znakova.
- *int* – predstavlja cjelobrojnu varijablu (cijeli broj), a ona može biti pozitivna ili negativna, najčešće u rasponu od -2147483648 do 2147483647.
- *float* – realni broj jednostruke preciznosti (32-bitni tip podataka), koji može pohraniti decimalne brojeve. Na primjer, `float broj = 10.54f;` ovdje varijabla `broj` predstavlja decimalni broj s jednostrukom preciznošću, dok sufiks `f` označava da se radi o tipu float.
- *double* – realni broj dvostruke preciznosti (64-bitni tip podataka), koji može pohraniti veći raspon decimalnih brojeva s većom preciznosti u usporedbi s float tipom. Na primjer, `double broj=10.54;` ovdje varijabla broj sadrži decimalni broj s dvostrukom preciznosti.

Uz osnovne tipove, postoje i složeni tipovi varijabli. U deklaraciji imena varijabla odvajaju se zarezima i naredba se završava znakom *točka-zarez*:

```
int prvi_broj, drugi_broj;
```

Nazivi varijabli koje se deklariraju su `prvi_broj` i `drugi_broj`, dok je njihov tip `int`.

2.2.1 Pridruživanje

U programskom jeziku C koristi se znak jednakosti = kao operator pridruživanja [2]. Tako vrijednost varijable s desne strane pridružujemo varijabli na lijevoj strani.

```
prvi_broj = 10;
```

2.3 Funkcije za ulaz i izlaz

Jedne od najčešće korištenih funkcija su ulaz i izlaz funkcije koje su ključne za interakciju programa s korisnikom. Prije korištenja funkcija, potrebno je uključiti zaglavlje *stdio.h*.

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

Funkcija `printf()` čita podatke iz programa i isporučuje te podatke na zaslon (ili općenitije u standardni output), odnosno omogućuje ispis rezultata i poruka, što pomaže u prikazu informacija i ispravci grešaka. S druge strane, funkcija `scanf()` čita podatke s tipkovnice (ili općenitije iz standardnog inputa) i isporučuje te podatke programu, čime omogućuje unos podataka od korisnika, što je nužno za dinamičko upravljanje informacijama unutar programa. Zajedno, ove dvije funkcije omogućuju da uspostavimo dvosmjernu komunikaciju s programom [3].

2.3.1 printf()

Funkcija `printf` je namijenjena za prikaz *stringova*. Opći format funkcije glasi:

```
printf(kontrolni_string,arg_1,arg_2,...,arg_n);
```

`kontrolni_string` predstavlja konstantni znakovni niz koji uključuje informaciju o formatiranju ispisa [4].

Kontrolni znakovni niz (*string*) predstavlja konstantan niz znakova koji se sastoji od različitih skupina znakova konverzije, a pritom je svakom argumentu pridružena po jedna grupa [2]. Svaka skupina znakova konverzije počinje sa znakom *postotka* (%) koji je praćen

oznakom konverzije koja označava tip podatka koji se obrađuje, kao što su *%c* za znak, *%d* za cijeli broj, *%f* za broj s pokretnim zarezom [2].

```
int a;
printf("a = %d", a);
```

2.3.2 scanf()

Funkcija *scanf* omogućava formatirani unos podataka sa standardnog ulaza, a opći format funkcije glasi :

```
scanf(kontrolni_string,arg_1,arg_2,...,arg_n);
```

gdje `kontrolni_string` predstavlja konstantni znakovni niz u kojem su informacije o vrijednostima koje se učitavaju u argumente `arg_1,…,arg_n` [4].

Kontrolni znakovni niz ima sličan format i funkciju kao gore opisana funkcija `printf`. Grupe kontrolnih znakova se unutar kontrolnog niza znakova mogu nastavljati bez razmaka ili pak mogu biti odvojene prazninama koje se prilikom učitavanja podataka ignoriraju. Za argumente funkcije `scanf` potrebno je napomenuti kako prilikom poziva treba koristiti adresni operator `&` kako bi se proslijedila adresa varijable u koju će se učitati podatak [2].

```
int broj;
scanf("%d", &broj);
```

2.4 Operatori

Operatori su osnovni alati za izgradnju izraza, a jezik C nudi uobičajeni skup tih operatora koji se nalaze u većini programskih jezika. To su simboli koji obavljaju različite operacije na operandima, poput aritmetičkih, logičkih ili relacijskih operacija. Oni omogućuju izvođenje osnovnih matematičkih izračuna, usporedbe vrijednosti i manipulaciju podacima unutar izraza [2].

2.4.1 Aritmetički operatori

Aritmetičke operatore primjenjujemo na numeričkim operandima, odnosno cjelobrojnim, realnim i znakovnim operandima [5]. Operacija dijeljenja cjelobrojnih operanada jednaka je cjelobrojnom dijeljenju, a rezultatu se odsijecaju decimalne znamenke za cjelobrojni rezultat. Nazivnik u operaciji mora biti različit od nule. Ako barem jedan od operanda nije

cijeli broj, tada će rezultat dijeljenja biti realan broj. Operacija *modulo* kao rezultat prikazuje ostatak od dijeljenja dva cjelobrojna operanda [5].

Primjer za vrijednosti A = 3, B = 2 je prikazan u *Tablica 2.1*.

OPERATOR	ZNAČENJE	PRIMJER
`+`	zbrajanje	`A + B = 5`
`-`	oduzimanje	`A - B = 1`
`*`	množenje	`A * B = 6`
`/`	dijeljenje	`A / B = 1`
`%`	modulo	`B%A = 2`

Tablica 2.1 Aritmetički operatori.

Operatori `*`, `/`, `%` imaju viši prioritet od + i − [5]. Bitno je napomenuti kada se `int` i `float` operandi miješaju, rezultat je tipa `float`. Za primjer 10 + 2.5f = 11.5 i 8.7f / 2 = 3.35. Operator `/` može dati neočekivane rezultate. Kada su oba njegova operanda cijeli brojevi, operator `/`, 'skraćuje' rezultat ispuštanjem decimalnog ostatka. Stoga je vrijednost od 1/2 je jednaka 0, a ne 0.5.

2.4.2 Unarni operatori

Unarni operatori prikazani u *Tablica 2.2* u programskom jeziku C djeluju na jedan operand i izvršavaju operacije poput negacije koristeći unarni minus, inkrementiranja ili dekrementiranja. Svi unarni operatori su u istoj prioritetnoj grupi i imaju viši prioritet od aritmetičkih operatora. Inkrementacija se radi operatorom `++` što povećava varijablu za jedan, dok se dekrementacija radi operatorom `--` što smanjuje varijablu za jedan [5].

OPERATOR	ZNAČENJE
!	Logički operator negacije, operand mora imati aritmetički tip, a rezultat je 1 ako je vrijednost njegovog operanda u usporedbi jednaka 0, odnosno 0 u suprotnom. Tip rezultata je `int`.

++	Operator inkrementiranja povećava vrijednost varijable za 1,ekvivalentno `x = x+1`. Preinkrement (`++x`): Varijabla se prvo poveća, pa koristi. `int x = 5;` `int y = ++x; // x = 6, y = 6` Postinkrement (`x++`): Varijabla se prvo koristi, pa povećava. `int x = 5;` `int y = x++; // x = 6, y = 5`
--	Operator dekrementiranja smanjuje vrijednost varijable za 1, ekvivalentno `x = x-1`. Predekrement (`--x`): Varijabla se prvo smanji, pa koristi. `int x = 5;` `int y = --x; // x = 4, y = 4` Postdekrement (`x--`): Varijabla se prvo koristi, pa smanji. `int x = 5;` `int y = --x; // x = 4, y = 4`
-	Unarni minus, operator mora imati aritmetički tip, a rezultat je negativ njegovog operanda.
&	Adresni operator, uzima adresu svog operanda.

Tablica 2.2 Unarni operatori.

2.4.3 Relacijski operatori

Relacijski operatori prikazani u *Tablica 2.3* imaju isti prioritet, ali niži od aritmetičkih operatora [5].

OPERATOR	ZNAČENJE	PRIMJER
<	Strogo manje	`A < B`
<=	Manje ili jednako	`A <= B`
>	Strogo veće	`A > B`
>=	Veće ili jednako	`A >= B`
==	Jednako	`A == B`
!=	Različito	`A != B`

Tablica 2.3 Relacijski operatori.

2.4.4 Operatori pridruživanja

Nakon što vrijednost izraza bude izračunata, često će se morati pohraniti u varijablu za kasniju upotrebu. Osnovni operator pridruživanja [5] je =, pa naredba pridruživanja glasi:

```
varijabla = izracunati_izraz;
```

Izrazi pridruživanja su prikazani u *Tablica 2.4*.

IZRAZ	EKVIVALENTNI IZRAZ
`i += 1`	`i = i + 1`
`i -= j`	`i = i-j`
`i *= j+2`	`i = i*(j+2)`
`i /= 2`	`i = i/2`
`i %= 2`	`i = i%2`

Tablica 2.4 Izrazi pridruživanja.

2.4.5 Logički operatori

Operandi logičkih operatora su logičke vrijednosti, odnosno logički izrazi gdje se cjelobrojne vrijednosti različite od nule interpretiraju kao istina, dok se nula interpretira kao laž. Vrijednost složenog logičkog izraza može biti 0 (laž) ili 1 (istina) [4].

OPERATOR	ZNAČENJE
&&	logičko I
||	logičko ILI
!	Logička negacija (unarno)

Tablica 2.5 Logički operatori.

Logički operatori prikazani u *Tablica 2.5* imaju niži prioritet od prioriteta relacijskih operatora i operatora jednakosti. Logički izrazi sastavljeni su od individualnih logičkih izraza povezanih operatorima && i || koji idu s lijeva na desno.

2.4.6 Prioriteti operacija

KATEGORIJA	OPERATORI	ASOCIJATIVNOST
Unarni operatori	!, ++, --, -, &	D → L
Aritmetički o.	*, /, %	L → D
Aritmetički o.	+, -	L → D
Relacijski o.	<, <=, >, >=	L → D
O. jednakosti	= =, !=	L → D
Logičko I	&&	L → D
Logičko ILI	||	L → D
O. pridruživanja	=, +=, -=, *=, /=, %=	D → L

Tablica 2.6 Prioriteti operacija (tablica preuzeta iz [4]).

2.5 Grananje

Kao što u životu ljudi donose odluke, tako i programi, da bi donijeli odluku, važu svoje mogućnosti. Naredba grananja omogućuje programu odabir između dvije alternative ispitivanjem vrijednosti uvjeta. Tako je omogućeno izvršavanje manjih grupa naredbi ovisno o tome je li ispunjen uvjet [2].

2.5.1 Naredba `if`

Naredba `if` omogućuje da provjerimo je li uvjet istinit ili lažan i program izvršava određeni kod prema rezultatu. Osnovni oblik grananja je:

```
if (uvjet)
    naredba;
```

Uvjet se obavezno postavlja u zagrade i dio je `if` grananja, nije dio naredbe. Kada se `if` naredba izvrši, uvjet u zagradama se izvrijedni; ako je vrijednost uvjeta različita od nule - što C interpretira kao istinito - izvršava se naredba iza zagrada [2].

Primjer, ako je vrijednost varijable A veća od 5, ispisuje se poruka na zaslonu.

```
if ( A > 5)
    printf("Vrijednost A je veća od 5");
```

Nakon naredbe `if` koja provjerava je li jedan uvjet istinit, ali što ako je cilj provjeriti nekoliko uvjeta ili provjeriti nije li ijedan uvjet istinit? Često se naredba `if` [1] koristi za provjeru nalazi li se varijabla unutar raspona vrijednosti, na primjer nalazi li se A između 0 i `N` , pišemo

```
if (0 <= A && A <= N)
    printf("Vrijednost A se nalazi između 0 i %d", N);
```

Operator `and` (`&&`) daje vrijednost *istine* samo ako su oba dana uvjeta istinita te je vrlo učinkovit: ako je prvi uvjet lažan, računalo se neće zamarati procjenom drugog uvjeta. Ono zna da ako je prvi uvjet lažan, onda cijeli uvjet mora biti lažan. Uz operator `and`, često se kao izraz `if` grananja koristi i operator `or` (`||`) koji ima vrijednost istine ako je bilo koji dani uvjet istinit [1].

```
if (A == 1 || A == 5)
    printf("A ima vrijednost 1 ili 5");
```

Sljedeći operator, `!` *operator logičke negacije* je koristan u `if` grananju jer omogućava provjeru uvjeta koji trebaju biti lažni za izvršavanje određenog bloka koda. Tako jednostavno možemo obrnuti logički izraz i kontrolirati tok programa u situacijama kada želimo izvršiti određenu akciju, samo ako uvjet nije ispunjen.

```
int A=0;
if(!A)
    printf("Nije A”);
```

Što ako želimo da naredba `if` kontrolira dvije ili više naredbi? `if` umjesto jedne naredbe može sadržavati blok naredbi koje su omeđene vitičastim zagradama i to se naziva *složenom naredbom* [2].

```
if (uvjet) {
    Naredba_1;
    //.....
    Naredba_N;
}
```

Ovisno o tome imali li uvjet istinitu ili lažnu vrijednost, čitav blok naredbi se izvršava ili ne izvršava.

2.5.2 Naredba `if-else`

Naredba `if` može imati i `else` dio [2]. Naredba `if-else` omogućuje kontroliranje toka programa na temelju toga je li uvjet istinit ili lažan, izvršavajući jedan od dva moguća bloka koda.

Osnovni oblik `if-else` naredbe je:

```
if (uvjet)
    naredba_1;
else
    naredba_2;
```

Uvjet kao i kod `if` naredbe stavlja se u zagradu. Ako je izraz `uvjet` istinit (vrijednost različita od nule), izvršava se prva naredba `naredba_1` ili blok naredbi iza `if`. U suprotnom, ako je uvjet lažan (vrijednost nula), izvršava se naredba ili blok naredbi iza else dijela [4]. Takva struktura omogućuje donošenje odluka u programu kada postoji potreba za alternativnim izvršavanjem blokova koda na osnovu ispunjenosti uvjeta.

Primjer, ako je vrijednost varijable `A` veća od 5, ispisuje se poruka na zaslonu. U protivnom se ispisuje alternativna poruka.

```
if (A > 5)
    printf("Vrijednost A je veća od 5");
else
    printf("Vrijednost A je manja ili jednaka 5");
```

Slično kao kod osnovnog `if` grananja, unutar `if-else` grananja mogu se nalaziti i složene naredbe, blokovi omeđeni vitičastim zagradama. Tako `if-else` naredba može kontrolirati više naredbi u oba slučaja.

```
if (uvjet) {
    naredba_1;
    naredba_2;
    //…
} else {
    naredba_3;
    naredba_4;
    //…
}
```

Više `if-else` naredbi može se nadovezati jedna na drugu i time se dobiva složena naredba:

```
if (uvjet_1)
    naredba_1;
else if(uvjet_2)
    naredba_2;
else
    naredba_3;
naredba_4;
```

Tako u gornjem primjeru složene naredbe [4], prvo se provjerava `uvjet_1`; ako je istinit, izvršiti će se `naredba_1`, a potom se prelazi na posljednju naredbu `naredbu_4` izvan grananja. Ako `uvjet_1` nije istinit, provjerava se `uvjet_2`, nakon čega se, ovisno o njegovoj istinitosti, izvršava `naredba_2` ili `naredba_3`. Program potom nastavlja s izvršavanjem naredbe `naredba_4`.

2.6 Petlje

Petlja je naredba koja omogućuje višestruko izvršavanje određene naredbe. U programskom jeziku C, svaka petlja sadrži kontrolni izraz. Prilikom svake iteracije, kontrolni izraz se procjenjuje; ako je izraz istinit – ima vrijednost koja nije nula – petlja se nastavlja izvršavati. C nudi tri vrste naredbi za ponavljanje: `for`, `while` i `do-while` petlje [2]. U ovom radu

fokus će biti stavljen isključivo na `for` petlju, budući da je razumijevanje `for` petlje ključno za sljedeća poglavlja i zadatke koje ćemo analizirati i rješavati.

2.6.1 `for` petlja

Petlja `for` prikladna je za petlje koje povećavaju ili smanjuju varijablu brojanja [2]. Najčešće se koristi kada se blokovi naredbi unaprijed ponavljaju određen broj puta.

Oblika je:

```
for (izraz_1; izraz_2; izraz_3) naredba;
```

Najčešće, prvi izraz, `izraz_1`, postavlja početne vrijednosti za parametre koji kontroliraju petlju. Zatim, drugi izraz, `izraz_2`, određuje uvjete za nastavak izvršavanja petlje, a treći izraz, `izraz_3`, mijenja vrijednosti parametara inicijaliziranih u prvom izrazu. Petlja `for` počinje inicijalizacijom kontrolnih parametara, pri čemu se uvjet iz drugog izraza procjenjuje na početku svakog prolaza kroz petlju, dok se treći izraz izvršava na kraju svakog prolaza [2].

Primjer u nastavku ispisat će brojeve od 0 do 5,

```
int i;
for (i = 0; i < 6; i++) {
    printf("%d\n", i);
}
```

gdje prvi izraz postavlja varijablu prije nego se petlja pokrene (`int i=0`), dok drugi izraz definira uvjet za pokretanjem petlje koji mora biti manji od 6. Ako je drugi izraz istinit, petlja počinje iznova, a ako je izraz lažan, petlja završava. Treći izraz povećava vrijednost varijable i svaki put kada je blok koda u petlji izvršen [2][6].

Nijedan od tri izraza u `for` petlji nije obavezan. Ako srednji izraz izostane, pretpostavlja se da je njegova vrijednost 1, što znači da će petlja trajati beskonačno. Na primjer, `for(;;);` predstavlja *beskonačnu petlju* koja ne izvršava nikakav kod [4].

Također je moguće postaviti jednu petlju unutar druge, što se naziva *ugniježđenom for petljom* [7]. Ugniježđene petlje su važan koncept u programiranju u C-u jer omogućuju ponavljanje određenih zadataka s različitim ulaznim vrijednostima više puta. Stoga se

ugniježđena petlja može definirati kao petlja unutar petlje, gdje unutarnja petlja dovršava sve svoje iteracije prije nego što vanjska petlja nastavi (s iteracijom).

```
for(inicijalizacija varijable; uvjet; inkrement){
    for(inicijalizacija varijable; uvjet; inkrement){
        //naredbe unutarnje petlje
    }
    //naredbe vanjske petlje
}
```

2.7 Funkcije

Sav kod u programskom jeziku C izvršava se unutar funkcija i mogu se smatrati građevnim blokovima C programa [2]. Najvažnija funkcija u bilo kojem C programu je `main()` funkcija koja je polazište za sav kod u programima [1]. Ako se u programu ne nalazi funkcija `main()`, program se neće moći pokrenuti. Pomoću funkcija program se može podijeliti na manje dijelove koje je lakše razumjeti i modificirati, ali i smanjiti ponavljanje koda u različitim dijelovima programa [2].

2.7.1 Definiranje funkcija

Kada se pokrene program, potrebna je provjera je li program uspješno izvršen ili ne. Funkcija `main()` ima povratni tip `int`, a provjera se radi analizom povratne vrijednosti funkcije `main`. Ako u funkciji `main` definiramo vraćanje nule, to znači da je program bio uspješan, dok vraćanje bilo koje druge vrijednosti ukazuje na problem u funkciji ili programu [1]. Naziv funkcije piše se iza povratnog tipa, iza čega slijede parametri funkcije u zagradama i naredbe funkcije omeđene vitičastim zagradama. Svaka (nova) funkcija je zapravo mali program sa svojim deklaracijama i naredbama.

Osnovni oblik definicije funkcije `main` je:

```
int main(){
    int A = 20;
    printf("Varijabla A ima vrijednost %d", A);
    // ostale naredbe….
    return 0;
}
```

Odnosno, definiciju funkcije možemo pisati u obliku:

```
povratni_tip naziv_funkcije (tip_1 arg_1,…, tip_n arg_n){
    //tijelo funkcije
    return vrijednost;
}
```

gdje nakon naziva funkcije dolazi popis argumenata u zagradi odijeljenih zarezima i svakom argumentu je nužno specificirati tip [2]. U vitičastim zagradama nalazi se tijelo funkcije koje uključuje deklaracije varijabli i naredbe [5]. Bitno je istaknuti da varijable definirane unutar funkcije ekskluzivno pripadaju toj funkciji i ne mogu biti pozvane i mijenjane od strane drugih funkcija [2].

Nakon što su uvedene varijable, `for` petlje i funkcije, može se napisati primjer korištenja ugniježđenih `for` petlji. Kroz jednostavan primjer koristeći `printf()` funkciju možemo razumjeti što se događa sa varijablama u svakoj iteraciji.

```
#include <stdio.h>
int main() {
    int i, j;
    for (i = 1; i <= 3; i++) {
        for (j = 1; j <= 2; j++) {
            printf("Ovo je iteracija i=%d, j=%d\n", i, j);
        }
    }
    return 0;
}
```

Vanjsku petlju postavimo da se pokreće tri puta, za `i` od 1 do 3, dok se unutarnja `for` petlja pokreće dva puta, za `j` od 1 do 2, za svaku iteraciju vanjske petlje. Funkcija `prinf` ispisat će trenutne vrijednosti varijabla `i` i `j` za svaku kombinaciju.

U ispisu programa dobiva se sljedeće:

```
Ovo je iteracija i=1, j=1
Ovo je iteracija i=1, j=2
Ovo je iteracija i=2, j=1
Ovo je iteracija i=2, j=2
Ovo je iteracija i=3, j=1
Ovo je iteracija i=3, j=2
```

Gotovo svi zadaci koji će biti obrađeni u kasnijim poglavljima bit će gore navedenog oblika.

2.7.2 Deklaracija funkcija

Prije poziva svake funkcije u programu, funkciju je potrebno *deklarirati* čime informiramo prevoditelja kakvu funkciju očekivati. Deklaracije se možemo promatrati kao potpis funkcije, odnosno informaciju o nazivu funkcije, tipove argumenata koje funkcija očekuje i

koji će tip podataka vratiti [1]. Obično se funkcije definiraju iznad ili ispod `main` funkcije, a ako se definira ispod, deklaracija je nužna.

Osnovni oblik deklaracije glasi:

```
tip_podatka naziv_funkcije(tip_1 arg_1,..., tip_n arg_n);
```

gdje se može primijetiti da deklaracija nema tijelo funkcije i završava točka-zarezom [1].

Unutar `main` funkcije, pozivi se rade tako da se navedu ime funkcije i vrijednosti argumenata u zagradama kako su navedeni u deklaraciji funkcije.

2.8 Pretprocesorske naredbe

Već je ranije spomenuta važnost uključivanja zaglavlja kao što je `<stdio.h>`, što omogućuje ispravno povezivanje programa s biblitekom i time korištenje funkcija i tipova podataka iz biblioteka koje su definirane u zaglavlju. Osim te biblioteke, prilikom pisanja programa u programskom jeziku C korisna su i druga zaglavlja. Naime, biblioteka je skup programskih komponenti koje se mogu ponovno koristiti u mnogim programima [8]. Prikazano je kako se funkcija treba deklarirati prije nego što se upotrijebi u programu, a najlakši način za to je uključivanje zaglavlja koje sadrži sve potrebne deklaracije funkcija. Svaka linija koda koja započinje sa znakom '#' razumije se kao pretprocesorska naredba koja omogućuje uključivanje zaglavlja ili drugih uputa pretprocesoru.

2.8.1 `#include` naredba

Naredba `#include` koristi se kako bi se zaglavlje programa napravilo dijelom prijevodne jedinice koja omogućuje obradu vanjskih datoteka zaglavlja od strane prevoditelja [8]. Naredba include piše se na vrh programa kao:

```
#include <datoteka_zaglavlja>
```

gdje pretprocesor uklanja liniju s naredbom `#include` i zamjenjuje je sadržajem datoteke koja je navedena.

Zaglavlje koje se koristi u ovom radu je `<stdio.h>`, koja pruža osnovne funkcije za ulaz i izlaz kao što su funkcije `scanf()` i `printf()`.

2.9 Kreiranje i pokretanje programa

Kako su pobliže objašnjeni svi osnovni dijelovi koji su potrebni u programu u programskom jeziku C za rješavanje zadataka koji slijede, u idućim poglavljima naglasak će biti stavljen na praktični aspekt rada s programima u C-u. Kreiranje programa u C-u započinje pisanjem izvornog koda u tekstualnom editoru koristeći naučene standardne C sintakse. Program je zatim potrebno spremiti u datoteku s ekstenzijom `.c`, što označava da je riječ o C izvornoj datoteci. Nazovimo datoteku `zadatak.c`. Da bi se program pokrenuo, potrebno ga je prevesti.

Na Linuxu, za prevođenje koda koristi se *C prevoditelj*, poput `gcc`. Prevođenje se izvodi naredbom `gcc zadatak.c -o zadatak.out`, koja pretvara izvorni kod i kreira izvršni program `zadatak.out`. Nakon uspješnog prevođenja, program se izvršava unosom naredbe `./zadatak.out` u terminalu, čime se dobiva povratni odgovor ili ishod iz programa [2].

Na Windowsu, isti postupak može se provesti pomoću *MinGW* (Minimalist GNU for Windows), koji omogućuje korištenje `gcc` prevoditelja u Windows okruženju. Nakon instalacije MinGW-a, program `zadatak.c` može se prevesti koristeći naredbu `gcc zadatak.c -o zadatak.exe`. Umjesto `.out`, u Windowsu se kreira izvršna datoteka s ekstenzijom `.exe`. Program se zatim izvršava dvostrukim klikom na `zadatak.exe` ili iz terminala unosom `zadatak.exe` [9]. Osim toga postoji i WSL (Windows Subsystem for Linux) koji omogućuje pokretanje Linux distribucija izravno unutar Windows okruženja. Moguće je instalirati Linux terminal i koristiti iste naredbe za kompajliranje kao na Linux sustavima, poput `gcc zadatak.c -o zadatak.out` [10].

3. Razvoj logičkog razmišljanja u programiranju: Uloga for petlji, if grananja i live codinga za studente

Studenti početnici u programiranju često teško razumiju specifične programske konstrukcije kao što su grananja i petlje koji predstavljaju temeljne strukture koje omogućuju kontrolu toka izvršavanja programa. Ove osnovne strukture obično se obrađuju samo na početku kolegija iz programiranja, ali njihov značaj ostaje ključan ne samo za daljnje učenje, već i za sve napredne korake u programiranju. Razumijevanje ovih koncepata kasnije se pokazuje neophodnim za rješavanje zahtjevnih zadataka.

3.1 Poteškoće s petljama i grananjem

Kroz `for` petlje, studenti uče kako upravljati ponavljanjem određenog zadataka, što je ključno za rješavanje problema koji zahtijevaju iterativni pristup. S druge strane `if` grananje pomaže studentima u razvijanju logičkog razmišljanja i razumijevanju uvjetnog izvršavanja koda [11]. Jedan od načina rješavanja izazova razumijevanja koncepata je pružiti studentima probleme u praksi u ovim temama koje se smatraju teškima za razumijevanje - kao što su *ugniježđene petlje* [11]. Osim ugniježđenih petlji teška su i *grananja sa više uvjeta*, a kroz interaktivne zadatke se omogućuje studentima da u stvarnom vremenu prate proces pokušaja i pogrešaka, gdje mogu testirati kod, uočiti njegove nedostatke, ispraviti ih i vidjeti kako njihov kod postaje funkcionalan. Poteškoće se mogu pojaviti na različitim razinama, od sintaktičkih pogrešaka do zabluda vezanih za pravilno izvršavanje petlje. Takve zablude često proizlaze iz nedostatka konceptualnog razumijevanja konstrukta petlje. Iako poteškoće nisu uvijek jasno prepoznate, one mogu uzrokovati dodatne pogreške tijekom programiranja, kao i daljnjeg učenja [12]. Konstrukcija petlje može zbuniti studente koji često imaju problema s razumijevanjem opsega petlje, koje će se linije naredbi ponavljati, koliko će se puta kod unutar petlje izvršiti i tako dalje [12].

Kako zablude ometaju učenje i često su prisutne među studentima, mnoga su se istraživanja usredotočila na korištenje različitih pristupa podučavanju kako bi se pomoglo u rješavanju pogrešnih predodžbi studenata i drugih poteškoća te tako poboljšala izvedba studenata u učenju programiranja. Jedan pristup je korištenje dobro odabranih primjera

programa jer studenti početnici u programiranju često iz njih konstruiraju svoje znanje. Primjeri programa trebaju biti jasni i osmišljeni kako bi pomogli studentima da steknu točno razumijevanje programiranja i poboljšaju prijenos znanja [12]. Sljedeća metoda koja može pomoći studentima da prebrode ove poteškoće je *pair programming* ili programiranje u paru. Ova praksa omogućuje da dva studenta dijele jedno računalo, pri čemu jedan piše kod (tzv. vozač), dok drugi daje komentare i prijedloge (tzv. navigator). Tijekom nastave uloge se izmjenjuju nekoliko puta, čime oba sudionika aktivno sudjeluju u rješavanju zadatka. Ova metoda ne samo da je korisna u praktičnom programiranju, već pomaže studentima da međusobno riješe nedoumice i zajednički dođu do riješenja [13]. Obje strane uče tijekom procesa – 'slabiji' student dobiva individualnu pomoć, dok 'jači' student uči kroz objašnjavanje i ponavljanje koncepta. Važno je osigurati da svi studenti sudjeluju u radu u parovima, kako nitko ne bi bio izdvojen, a uloge treba mijenjati više puta tijekom riješavanja kako bi se spriječilo da jedna osoba dominira zadatkom [13]. Zahtijevati od studenata da čitaju, prate i objašnjavaju primjere programa također je korisna strategija [12]. Osim dobrih primjera, rješavanje problema je efektnije i praktično podijeliti u *četiri koraka* [14]:

1. *Shvatiti problem*
2. *Odrediti kako riješiti problem*
    a. *U nekom obliku*
    b. *U obliku koji je kompatibilan s računalom*
3. *Prevesti rješenje u programski jezik*
4. *Testirati i ispraviti program ukoliko ima grešku*

Za rješavanje problema najvažniji korak je razumijevanje procesa stvaranja rješenja i koda u stvarnom vremenu. Jedna od metoda koja pomaže u boljem shvaćanju ovog procesa je *live coding*, koja studentima pruža mogućnost da prate kako se teorija pretvara u praksu. U nastavku će detaljnije biti prikazano kako *live coding* doprinosi učenju i razvoju programerskih vještina.

3.2 Live coding

Izraz *live coding*, koji se ponekad prevodi kao *kodiranje uživo* ili *programiranje uživo*, odnosi se na metodu podučavanja programiranja u kojoj profesor piše kod u stvarnom vremenu pred studentima, omogućujući im da prate proces stvaranja i ispravljanja koda dok se odvija [15]. Dvije najčešće korištene tehnike su primjeri statičnog koda i programiranje

uživo. Statički primjeri koda su programi (isječci koda) koje je napisao profesor prije nastave i koriste se za objašnjenje kako isječak koda radi [15]. Kada se studentima prezentira unaprijed napisani kod, njihovo kognitivno opterećenje može biti visoko jer možda nisu sigurni na što se usredotočiti u kodu, pa se mogu osjećati preopterećeno [15]. S druge strane, može se pretpostaviti, ako se kod piše redak po redak kako se formira u nastavnom satu koristeći *live coding*, kognitivno opterećenje studenata moglo bi biti manje jer oni točno znaju na što se trebaju usredotočiti što može povećati njihovo razumijevanje osnovnih koncepata.

Prednosti korištenja *live coding*-a kao metodologije aktivnog učenja za podučavanje programiranja, prema relevantnoj literaturi uključuju [16]:

1. *Demonstracija u stvarnom vremenu* – Programiranje uživo omogućuje nastavnicima da u stvarnom vremenu pišu i pokreću kod, demonstrirajući korak po korak ključne koncepte i tehnike. Osim toga, brzina pisanja koda uživo usporava tempo nastave, što osigurava da studenti imaju dovoljno vremena za praćenje i razumijevanje. Studenti tako mogu promatrati način razmišljanja, strategije rješavanja problema i metode otklanjanja pogrešaka koje profesor koristi, čime se poboljšava njihovo razumijevanje.
2. *Trenutačne povratne informacije* – Dok se kod piše i izvodi uživo, studenti odmah vide rezultate i ispravnost koda. To omogućuje trenutnu povratnu informaciju, što pomaže studentima da prepoznaju i isprave pogreške u stvarnom vremenu. Ova neposredna povratna informacija ključna je za aktivno učenje i dublje razumijevanje programskih koncepata.
3. *Interaktivnost i angažman – Live coding* potiče aktivno sudjelovanje studenata, što im omogućuje postavljanje pitanja, predlaganje izmjena i razmjenu informacija tijekom predavanja. Profesori mogu odgovoriti na "što ako" pitanja i prilagoditi lekciju u stvarnom vremenu ovisno o interesima i pitanjima studenata. Tako se lekcija može odvijati u smjeru koji je najkorisniji za studente, što doprinosi njihovom aktivnom sudjelovanju i angažmanu [13]. Dodatno istraživanja [13] su pokazala da samo promatranje demonstracija pisanja programa nije nužno korisno za studente i da oni često zaboravljaju ili čak krivo pamte ishod demonstracije. Kako bi se demonstracije pisanja programa učinile učinkovitijima, važno je potaknuti

studente da unaprijed predviđaju ishod programa prije nego što ga profesor pokrene. Ova predviđanja mogu biti izražena na različite načine, primjerice dizanjem ruku, podizanjem kartica s oznakama A, B, C, D ili razgovorom s kolegom. Zapisivanje i iznošenje predviđanja pomaže studentima da više obrate pažnju i reflektiraju o onome što uče. Bez obzira na ishod, profesori bi trebali koristiti netočna predviđanja kao priliku za daljnju raspravu i istraživanje, bez kritiziranja studenata [13].

4. *Pogreške i otklanjanje pogrešaka* – Programiranje uživo omogućuje nastavnicima da pokažu kako se nositi s pogreškama i njihovim otklanjanjem te ključnim aspektima programiranja. Promatranjem profesorovog procesa u stvarnom vremenu, studenti stječu uvid u učinkovite strategije rješavanja problema i rukovanje pogreškama, što ih priprema za pisanju vlastitog rješenja programa.
5. *Vizualno i auditivno učenje* – Programiranje uživo kombinira vizualne i auditivne modalitete učenja, prilagođavajući se različitim stilovima učenja. Studenti istovremeno promatraju kod na ekranu i slušaju objašnjenja profesora, što poboljšava razumijevanje i pamćenje koncepata.
6. *Prilagodljivost* – Nastava koja koristi *live coding* može se prilagoditi prema napretku studenata i njihovim povratnim informacijama. Nastavnici mogu odmah mijenjati primjere koda kako bi odgovorili na specifična pitanja ili detaljnije objasnili određene koncepte. Ova fleksibilnost omogućuje personalizirano učenje, osiguravajući da sadržaj bude relevantan i prilagođen potrebama studenata.

Live coding donosi brojne prednosti u nastavi programiranja, no ima svojih nedostataka koji se mogu izbjeći ili umanjiti praksom. Nastavnici ponekad mogu ići presporo zbog raznih razloga [13].

Također, previše vremena može se potrošiti na pisanje dijela programa koji nije ključan za lekciju (npr. unos zaglavlja), što može omesti studente i odvratiti im pažnju od glavne teme. Ovaj problem može se izbjeći unaprijed pripremljenim kosturom programa koji već sadrži potrebne dijelove ili tako da se relevantni dijelovi programa kopiraju i zalijepe tijekom predavanja. Važno je napomenuti da live coding ne mora uvijek početi s praznim

ekranom. Profesori mogu studentima dati početni kod temeljen na prethodno naučenim konceptima, a zatim ga proširiti ili modificirati uživo [13].

Iako neka istraživanja [15], [17] ne pronalaze značajne razlike u uspjehu studenata na ispitima između programiranja uživo i tradicionalnih metoda, ona otkrivaju određene prednosti, poput smanjenja vanjskog kognitivnog opterećenja studenata, zahvaljujući usporenom tempu predavanja tijekom programiranja uživo [15].

Konačno, *live coding* kao metoda aktivnog učenja u podučavanju programiranja pruža dinamično i angažirajuće iskustvo koje ostavlja snažan dojam na studente. Ova metoda potiče studente na aktivno sudjelovanje, eksperimentiranje i međusobnu suradnju, što doprinosi dubljem razumijevanju programskih koncepata i poboljšanju njihovih vještina u rješavanju problema [16] što koriste u daljnjem obrazovanju.

4. Interaktivni zadaci ispisa uzoraka koristeći for petlje i grananja: primjeri zadataka i njihovih rješenja

Namjera ovog rada je izraditi više različitih zadataka sličnih težina za vježbu i provjeru znanja koristeći `for` petlje i `if` grananja u programskom jeziku C. U zadacima je dan jedan uzorak ispisan posebnim znakom `#`, a zadatak studenta je napraviti program u kojem koristeći funkciju `printf`, razne uvjete, `for` petlje i `if` grananje ispisuje zadani uzorak. Uzorak se može definirati kao slovo ili znak, a zadan je izgled uzorka u nekoliko različitih veličina. Od studenta se traži izrada računalnog programa kojem je ulazna vrijednost veličina uzorka, a izlaz je ispis tog uzorka u zadanom obliku i veličini. Ovakvi zadaci omogućuju studentima da praktično primijene osnovne strukture programiranja, istovremeno razvijajući logičko razmišljanje i sposobnost rješavanja problema.

4.1 *Korištenje petlji i uvjetnog grananja za ispis uzoraka*

Tehnika ispisivanja uzoraka odnosi se na korištenje petlji i uvjetnog grananja za generiranje različitih vizualnih uzoraka pomoću znakova kao što su `#`, `$`, `*` i slični, u proizvoljnoj veličini. Ova tehnika omogućuje studentima da uvježbaju osnovne koncepte programiranja i da bolje razumiju kontrolu toka programa, budući da ona zahtijeva precizno upravljanje petljama odnosno ponavljanjima i uvjetima u kodu. Jedna od karakteristika ove tehnike ispisivanja uzoraka je sposobnost da poveća angažman studenata kroz vizualne rezultate. Studenti brže uče kada vide neposredan rezultat svog rada, što im omogućuje da lakše prepoznaju i isprave pogreške pripremajući se za slične provjere znanja [17]. Uz spomenutu karakteristiku, ističu se još druge dvije karakeristike. To su strojno ocjenjivanje rješenja zadataka koje je brzo i precizno no pod cijenu da jako penalizira manje pogreške, te karakteristika da provjerava znanje ugnježđenih petlji kombiniranih s grananjima.

Ovaj tip zadataka idealan je za korištenje na vježbama i provjerama znanja jer pruža brzo procjenjivanje studentskog razumijevanja osnovnih struktura kontrole toka u programiranju koji su ključni za daljnje učenje. Primjer jednog uzorka dan je na *Slika 4.1*.

Slika 4.1 Primjer uzorka.

4.2 Kako osmisliti zadatak ispisa uzorka

Proces osmišljavanja zadataka može se podijeliti na nekoliko koraka, uključujući istraživanje, skiciranje, testiranje i analizu uzorka. Svaki korak igra ključnu ulogu u osiguravanju da zadaci budu izazovni, ali i rješivi.

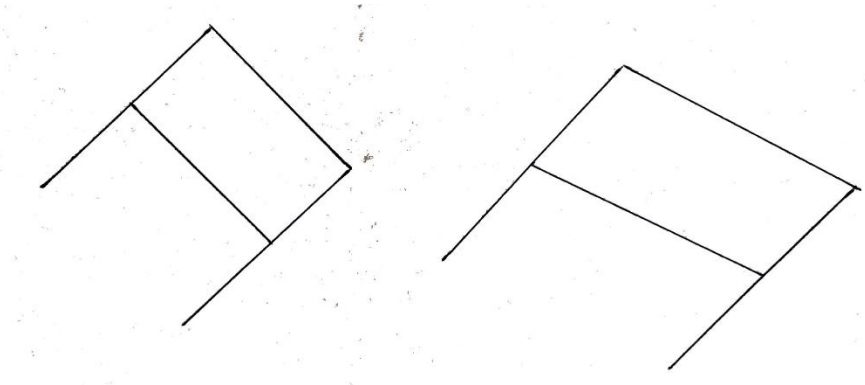
Korak 1: Istraživanje i odabir uzorka

Proces osmišljavanja zadataka započeo je istraživanjem različitih oblika slova abecede, brojeva i drugih sličnih uzoraka. Cilj je osmisliti zadatke koji su dovoljno izazovni, a odabir zanimljivog uzorka pokazao se posebno zahtjevnim budući da je bilo važno osigurati da uzorak bude dovoljno jasan i logički povezan kako bi studenti mogli razumjeti i riješiti određeni zadatak.

Korak 2: Skiciranje i testiranje uzorka

Nakon odabira potencijalnih uzoraka, uzorci su prvo skicirani olovkom na papiru radi boljeg razumijevanja njihove strukture i izgleda. Skica uzorka je prikazana na *Slika 4.2*. Skiciranje na papiru omogućilo je procjenu vizualne privlačnosti i jasnoće uzorka, dok je kasnija izrada u tekstualnom editoru prikazana na *Slika 4.3* omogućila preciznije određivanje

optimalne veličine uzorka. Veličina uzorka je ključna za jasnoću prikaza, posebno za uočavanje načina na koji su sve linije uzorka povezane i kako se međusobno nadovezuju.



Slika 4.2 Prva skica uzorka na papiru.

```
-------#-------
------#-#------
-----#---#-----
----#-----#----
---#-#-----#---
--#---#-----#--
-#-----#-----#-
#-------#-----#
----------#---#-
-----------#-#--
------------#---
-----------#----
----------#-----
---------#------
--------#-------
```

Slika 4.3 Skica uzorka u tekstualnom editoru.

Korak 3: Odabir početne veličine uzorka

Veličina početnog uzorka temeljena je na brojevima 8 i 15, pri čemu je širina ili visina uzorka odgovarala tim vrijednostima, uz pretpostavku da je početni indeks uzorka 1. Ovaj pristup omogućio je stvaranje uzorka idealne veličine koji je jasan i praktičan za rješavanje.

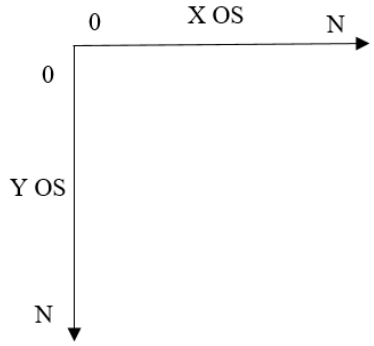
Korak 4: Rad s razmacima i analiza linija

Kako bi se osigurala pravilna raspodjela prostora unutar uzorka, u početnim fazama dizajna korišten je znak minusa − umjesto bjeline. To je omogućilo bolji pregled rasporeda

razmaka između znakova. Znakovi minusa su kasnije zamijenjeni bjelinama. Nakon oblikovanja uzorka u tekstualnom editoru, uzorak je podijeljen na linije, pri čemu je svaka linija analizirana zasebno kako bi se utvrdila logika iza svakog segmenta uzorka.

Korak 5: Matematička analiza linija

Uzorak koji se sastoji od crtica i znaka ljestva može se promatrati kao koordinatni sustav (*Slika 4.4*), pri čemu je vodoravna os X, a okomita os Y. Svaka linija unutar uzorka može se definirati pomoću koordinata početka i kraja, što se može zapisati kao par `[x1,y1]` za početak i `[x2,y2]` za kraj linije. Ovaj pristup, koji omogućuje precizno definiranje uvjeta unutar `for` petlji i `if` grananja kako bi se osiguralo da se svaki dio uzorka ispravno ispiše, podsjeća na oblik jednadžbe pravca, gdje se jednadžba pravca može koristiti za određivanje uvjeta koji definiraju svaku liniju.



Slika 4.4 Izgled zamišljenog koordinatnog sustava uzorka.

4.3 Primjer i analiza jednostavnih uzoraka

Kada se vježba ispis uzoraka, ključno je započeti s jednostavnim primjerima kroz koje studenti postupno razvijaju temeljna znanja koja su potrebna za uspješno rješavanje kompliciranijih zadataka u budućnosti i na ispitima. Prilikom rješavanja jednostavnog uzorka, cilj je koristiti osnovne elemente, `for` petlje i `if` grananja, kako bi se jasno prikazao postupak ispisa uzorka.

4.3.1 Primjer 1

Prikazani uzorak u *Tablica 4.1* sastoji se od gornje i bočne linije ispunjene znakom `#`, dok je središnji dio uzorka ispunjen bjelinama.

N = 8	N = 9	N = 10	N = 11	N = 12
				

Tablica 4.1 Jednostavni primjer 1: Uzorak za veličine od N = 8 do N = 12.

Za zadanu veličinu uzorka, gdje je `N=8`, prvo su određene koordinate početka i kraja linija koje služe za precizno opisivanje uzorka:

- Za gornju liniju, koordinate su `[x1,y1,x2,y2] = [1,1,8,1]`, što u općenitom obliku postaje `[1,1,N,1]`, gdje `N` predstavlja veličinu uzorka.
- Za lijevu liniju, koordinate su `[x1,y1,x2,y2] = [1,1,1,8]`, što u općenitom obliku postaje `[1,1,1,N]`.

Iz opisa linije mogu se odrediti granice `for` petlji koje će kontrolirati ispisivanje uzorka. Ako se uzme da su `y` koordinate vezane za vanjsku petlju označenu sa `i`, a da su `x` koordinate vezane uz unutarnju petlju odnosno za `j`, iz koordinata svake linije određujemo gornje granice `for` petlja. Tada se iz koordinata linije `[1,1,1,8]` može zaključiti da vanjska `for` petlja (`i`) treba iterirati od 1 do `N`, jer je maksimalna `y` koordinata 8, što odgovara vrijednosti `N`. Slično tome, iz koordinata linije `[1,1,8,1]` proizlazi da unutarnja `for` petlja (`j`) treba iterirati od 1 do `N`, jer je maksimalna `x` koordinata također 8, odnosno `N`.

Prilikom pisanja programa potrebno je uključiti standardnu C biblioteku pomoću `#include <stdio.h>`, što omogućava korištenje funkcija kao što su `printf` i `scanf` [8]. Svaki C program mora sadržavati glavnu funkciju `main()`, koja služi kao početna točka izvršavanja programa. Struktura ove funkcije, kao što je ranije objašnjeno, započinje deklaracijom varijabli, a završava s naredbom `return 0;`, što označava

uspješan završetak programa [2]. `For` petlje koje su definirane unutar `main` funkcije koriste se za iteraciju kroz redove i stupce, gdje vanjska petlja kontrolira redove (`i`), a unutarnja petlja stupce (`j`) što vidimo u *Ispis programa 4.1*. Sljedeći korak u izradi programa je kreiranje uvjeta unutar unutarnje `for` petlje koji će odrediti kada se ispisuje znak `#`, čime se oblikuje željeni uzorak.

```c
#include <stdio.h>
int main(){
    int i,j,N;
    scanf("%d",&N);
    for( i = 1; i <= N; i++){
        for(j = 1; j <= N; j++){
        }
    }
    return 0;
}
```

Ispis programa 4.1 Jednostavni primjer 1: ugniježđene for petlje.

Nakon što su definirane granice petlji, slijedi analiza koordinata linija kako bi se odredili uvjeti za ispisivanje znaka `#`. Svaka linija opisana je svojim koordinatama u formatu `[x1,y1,x2,y2]`, pri čemu se analiziraju općeniti obrasci poput `[1,1,n,1]`. Cilj je pronaći povezanost između veličina `i`, `j` (indeksi unutar petlji) i `N` (veličina uzorka) kako bi se kreirao općeniti uvjet koji kontrolira ispis određene linije. U ovom primjeru, prvo se kreće od analize okomite linije i zatim se prelazi na analizu vodoravne linije.

Proučavanje `x` koordinata koje predstavljaju unutarnju petlju `j` (prvi i treći element u koordinatama) pokazuje da vrijednost `j` ide od 1 do N. Stoga, uvjet za ispis može biti definiran kao `j>=1 && j<=N`, što omogućava ispis svih znakova u tom rasponu. S druge strane, analiza `y` koordinata, koje odgovaraju vanjskoj petlji `i` (drugi i četvrti element u koordinatama), pokazuje da je vrijednost `y` konstantna, što se može izraziti uvjetom `i==1` za određeni redak. Logički operator `&&` (AND) se koristi jer svi navedeni uvjeti moraju biti ispunjeni kako bi se ispisao znak `#`. U ovom slučaju:

- `i==1`: Provjerava je li trenutna `y` koordinata (vanjska petlja) u prvom retku.
- `j>=1 && j<=N`: Provjerava je li trenutna `x` koordinata (unutarnja petlja) unutar raspona od 1 do `N`.

Logički operator `&&` koristi se kako bi se osiguralo da se znak `#` ispisuje samo kada su oba uvjeta istovremeno ispunjena – kada se nalazimo u prvom retku (`i==1`) i u ispravnom rasponu stupaca (`i` od 1 do `N`). Ako bilo koji od ovih uvjeta nije ispunjen, znak `#` se neće ispisati. Konačni uvjet za prvu liniju je: `j>=1 && j<=N && i==1.`

Isti postupak se ponavlja i za sljedeću liniju, čije su koordinate `[1,1,1,N]`. U ovom slučaju, `x` koordinata (`j`) je stalno jednaka 1, dok `y` koordinata (`i`) ide od 1 do `N`. Dakle, ova linija se proteže u prvoj koloni (`j==1`), a ispisuje znak `#` za sve vrijednosti i od 1 do `N`.

- `j==1`: Provjerava je li trenutna `y` koordinata (vanjska petlja) u prvom retku.
- `i>=1 && i<=N`: Provjerava je li trenutna `x` koordinata (unutarnja petlja) unutar raspona od 1 do `N`.

Ponovno koristimo logični operator `&&` (AND) iz istog razloga pa konačno uvjet unutar `if` strukture glasi: `j==1 && i>=1 && i<=N`. Ovaj uvjet osigurava ispisivanje znaka `#` na prvoj koloni za sve retke, čime se formira lijeva vertikalna linija u uzorku. Kombiniranjem ovih uvjeta pomoću logičkih i relacijskih operatora, kreira se jedinstveni uvjet unutar `if` strukture koji precizno kontrolira kada se ispisuje znak `#`. Potrebno je obraditi i sve ostale pozicije unutar petlji što se postiže `else` dijelom, koji ispisuje bjelinu (' ') za sve pozicije koje ne zadovoljavaju uvjete za ispis `#`. Tako se osigurava da se unutar uzorka pravilno oblikuje prazni prostor koji dodatno naglašava oblik željenog uzorka.

U rješavanju ovog zadatka moguće je koristiti različite pristupe za provjeru uvjeta. Može se koristiti klasična struktura s više uvjeta pomoću `if, else if, else`, gdje svaki uvjet zasebno provjerava različite linije. Alternativno, svi uvjeti mogu se kombinirati unutar jednog `if` izraza koristeći logičke operatore, kao što je: `if ((j<=N && i==1) || (j==1 && i<=N))`. Logički operator `||` (OR) koristi se kako bi se provjerilo zadovoljava li trenutna pozicija barem jedan od uvjeta. U ovom primjeru, ispis će se dogoditi ako je zadovoljen prvi uvjet (horizontalna linija) *ili* drugi uvjet (vertikalna linija). Na kraju svake iteracije vanjske `for` petlje, ali izvan unutarnje `for` petlje, koristi se funkcija `printf("\n");` kako bi se nastavilo ispisivanje u novom retku, čime se pravilno oblikuje struktura ispisa uzorka. U *Ispis programa 4.2* prikazan je konačan oblik koda u kojem su definirane sve petlje i uvjeti za ispis uzorka. Nakon što se kod pokrene,

korisnik unosi željenu veličinu uzorka N, primjerice 8, a rezultat se prikazuje kao ispis uzorka koji se može vidjeti u *Tablica 4.2*, gdje su ispravno prikazane linije i praznine za različite vrijednosti N prema zadanim pravilima.

```c
#include <stdio.h>
int main(){
    int i,j,N;
    scanf("%d",&N);
    for( i = 1;i<=N; i++){
        for(j = 1; j<=N; j++){
            if ((j >= 1 && j <= N && i == 1)
            || (j == 1 && i >= 1 && i<= N)){
                printf("#");
            }
            else{
                printf(" ");
            }
        }
        printf("\n");
    }
    return 0;
}
```

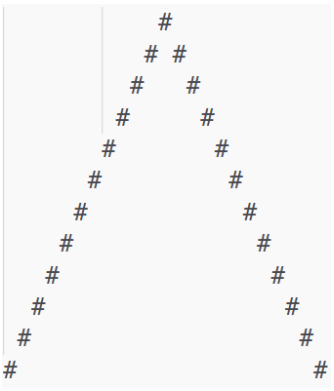
Ispis programa 4.2 Jednostavni primjer 1: program za ispis uzorka.

N = 8	N = 9	N = 10
8 ######## # # # # # # # [1] + Done	9 ######### # # # # # # # # [1] + Done	10 ########## # # # # # # # # # [1] + Done

Tablica 4.2 Jednostavni primjer 1: ispis uzorka za N=8, N=9 i N=10.

4.3.2 Primjer 2

U ovom primjeru je fokus na riješavanju uzorka s linijama pod kutom. Prikazani uzorak sastoji se od simetričnih linija sastavljene od znaka # koje tvore oblik sličnom strelici prema gore (*Tablica 4.3*).

N = 8	N = 9	N = 10
		
N = 11	**N = 12**	
		

Tablica 4.3 Jednostavni primjer 2: Uzorak za veličine od N = 8 do N = 12.

Analizom uzorka za zadanu veličinu `N=8` koordinate linija definirane su na sljedeći način:

- Za lijevu liniju koordinate su `[1,8,8,1]`, što u općenitom obliku linije postaje `[1,N,N,1]`
- Za desna liniju koordinate su `[8,1,15,8]`, što u općenitom obliku postaje `[N,1,2*N-1,8]`

Na temelju koordinata linija, određene su granice `for` petlji. Vanjska petlja `i` ide od 1 do 8 iz razloga što uzorak ima visinu od 8 redaka. Unutarnja petlja `j` iterira od 1 do `2*N - 1`, što znači da uvjet za unutarnju petlju možemo pisati kao `j<=2*N - 1`.

Da bi se došlo do uvjeta za crtanje linije koja prolazi kroz zadane koordinate `[1,N,N,1]`, potrebno je pronaći način kombiniranja `x` i `y` koordinata (tj. `i` i `j`) svake točke

kako bi se dobio zajednički obrazac. Jedan od prvih pristupa može biti oduzimanje y koordinata od x koordinata za obje točke. Međutim, pri oduzimanju koordinata prve točke (N,1) dobiva se razlika od N-1 (primjerice, za N=8, rezultat je 8 - 1 = 7), dok za drugu točku (1,N) je razlika 1-N (primjerice, 1 - 8 = -7), što daje suprotan rezultat (negativnu vrijednost). Budući da rezultati oduzimanja nisu jednaki (7 i -7), ovaj pristup nije prikladan za definiranje uvjeta.

Umjesto toga, zbrajanjem koordinata obje točke može se uočiti obrazac. Zbrajanjem koordinata prve točke (x1+y1 = 1+N, što za N=8 daje 1 + 8 = 9) i druge točke (x2+y2 = N+1, što za N=8 daje 8 + 1 = 9) dobiva se isti rezultat – 9. Ovaj rezultat povezan je s veličinom uzorka N, jer je 9 zapravo N+1. Stoga se uvjet za crtanje ove linije može općenito zapisati kao i+j == N+1.

- Lijeva linija ima pripadajući uvjet: i+j == N+1
- Desna linija ima pripadajući uvjet: j-i == N-1

Osim ove metode, kako bi se lakše razumjeli uvjeti za crtanje desne i lijeve linije u ovom uzorku, dobro je prisjetiti se jednadžbe pravca i identificirati varijable i i j. Općeniti oblik jednadžbe pravca je y = ax + b, gdje je a nagib pravca, a b presjek pravca s osi y. Ako pretpostavimo da se koriste varijable i za y-koordinatu (y → i) i j za x-koordinatu (x → j), tada se jednadžba pravca y = ax + b može preformulirati kao: i = - j+(N+1). U ovom slučaju, a=-1, b=N+1. Ova jednadžba opisuje nagib pravca i presjek s osi y odnosno i. Iako bi na prvi pogled moglo izgledati da je nagib pravca pozitivan, jer linija ide prema gore u uzorku, važno je razumjeti kako rastu vrijednosti za i i j u ovom specifičnom koordinatnom sustavu. Naime, osi ne idu standardno - gore za y-os i desno za x-os - već i vrijednosti rastu prema dolje, a j vrijednosti prema desno, kako je prikazano na *Slika 4.4*. Na taj način možemo ispravno postaviti uvjete za crtanje linija u uzorku, pri čemu se za lijevu liniju koristi uvjet i+j == N+1 i za desnu liniju i = j - (N-1) odnosno uvjet je j-i == N-1.

Jednim od ova dva postupka dolazi se do uvjeta koji se koristi u if strukturi kako bi se ispisao znak # na odgovarajućim lokacijama. Budući da se oba uvjeta moraju provjeriti za različite dijelove uzorka, logički operator || (OR) idealan je za kombiniranje tih uvjeta unutar jedne if strukture. Korištenjem uvjeta (i+j == N+1) || (j-i == N-1),

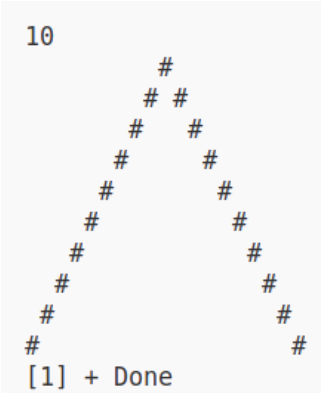
postiže se da se znak # ispisuje kada je zadovoljen barem jedan od uvjeta – bilo za lijevu ili desnu liniju. Na taj način, kod ostaje pregledan dok osiguravamo ispis uzorka.

U *Ispis programa 4.3* prikazan je konačan program za ovaj primjer.

```c
#include <stdio.h>
int main(){
    int i,j,N;
    scanf("%d",&N);
    for( i = 1;i<=N; i++){
        for(j = 1;j<=2*N-1; j++){
            if ((i+j == N +1) || (j-i == N-1)){
                printf("#");
            }
            else{
                printf(" ");
            }
        }
        printf("\n");
    }
    return 0;
}
```

Ispis programa 4.3 Jednostavni primjer 2: program za ispis uzorka.

Nakon pokretanja programa, korisnik unosi željenu veličinu uzorka `N`, a ispis rezultata može se vidjeti u *Tablica 4.4*.

N = 8	N =9	N=10
		

Tablica 4.4 Jednostavni primjer 2: ispis uzorka za N=8, N=9 i N=10.

Prikazani primjeri jednostavnih uzoraka demonstriraju osnovne tehnike korištenja `for` petlji i `if` grananja za ispisivanje uzoraka. U prvom primjeru, fokus je bio na ravnim linijama, dok je u drugom primjeru uvedena logika koja uključuje linije pod kutom. Ovi zadaci jasno pokazuju kako se uz pomoć jednostavnih uvjeta i pravilnog razumijevanja

koordinata mogu generirati uzorci. Razumijevanje ovih osnovnih principa ključan je korak prema težim primjerima.

4.4 *Primjer i analiza uzoraka umjerene težine*

Nakon što su obrađeni osnovni primjeri, prelazi se na uzorke umjerene težine koji donose nove izazove. Ovi zadaci su teži jer uključuju veći broj linija sastavljenih od jedne vrste znaka, što zahtijeva pažljivije određivanje uvjeta i dulje vrijeme rješavanja. Uz to, obradit će se više primjera nego u prethodnom poglavlju, budući da je zadatak ovog rada osmisliti veliki broj takvih zadataka umjerene težine koji su idealni za provjeru znanja. Iako su ovi zadaci zahtjevniji, i dalje su dovoljno intuitivni što ih čini prikladnim za vježbu i provjeru razumijevanja programskih struktura. Kroz sljedeće primjere, analizirat ćemo kako povećani broj faktora uzorka utječe na kreiranje uvjeta i pravilno oblikovanje uzoraka.

4.4.1 Primjer 1 umjerene težine

Ovaj primjer uzorka kombinira dvije dijagonalne linije, jednu vertikalnu i jednu horizontalnu liniju koje se međusobno presijecaju na 3 mjesta unutar uzorka. Takva struktura ponovno zahtijeva pažljivo definiranje uvjeta za svaku liniju, kako bi se svi dijelovi uzorka pravilno ispisali.

N = 8	N = 9	N = 10
		
N = 11	**N = 12**	
		

Tablica 4.5 Primjer 1 umjerene težine: Uzorak za veličine od N = 8 do N = 12.

U ovom primjeru analizira se uzorak (*Tablica 4.5*) za vrijednosti N = 8 sa *Slika 4.5* i postupno se dolazi do općenitih oblika linija za proizvoljnu vrijednost `N` gdje su koordinate zahtjevnije nego u prethodnim primjerima.

Slika 4.5 Primjer 1 umjerene težine: oznake linija.

Za `N = 8`, koordinate linija definirane su na sljedeći način:

- `[1,8,8,1]`
- `[8,1,8,8]`
- `[8,8,15,8]`
- `[8,15,15,8]`

Prva linija ima koordinate `[1,8,8,1]`, što označava da linija ide od točke (1, 8) do točke (8, 1). Kada se uzme vrijednost `N=8`, ove koordinate mogu se općenito izraziti kao `[1,N,N,1]`. Ovaj opći oblik linije ostaje isti bez obzira na vrijednost `N`.

Druga linija ima koordinate `[8,1,8,8]`, što znači da linija ide od točke (8,1) do točke (8, 8). U općenitom obliku, za `N=8`, ove koordinate postaju `[N,1,N,N]`.

Treća linija ima koordinate `[8,8,15,8]`, što znači da linija ide od točke (8, 8) do točke (15, 8). Kada se generalizira za proizvoljnu vrijednost `N`, koordinate postaju

`[N,N,2*N-1,N]`. U ovom slučaju, izraz `2*N-1` predstavlja najdalju točku na desnoj strani uzorka.

Četvrta linija ima koordinate `[8,15,15,8]`, što znači da linija ide od točke (8, 15) do točke (15, 8). U općenitom obliku, koordinate postaju `[N,2*N-1,2*N-1,N]`, gdje `2*N - 1` ponovno predstavlja krajnju točku na desnoj strani uzorka.

Općeniti oblici svake linije su:

- Za liniju `[1,8,8,1]`, općeniti oblik je `[1,N,N,1]`
- Za liniju `[8,1,8,8]`, općeniti oblik je `[N,1,N,N]`.
- Za liniju `[8,8,15,8]`, općeniti oblik je `[N,N,2*N-1,N]`
- Za liniju `[8,15,15,8]`, općeniti oblik je `[N,2*N-1,2*N-1,N]`

Na temelju analize koordinata, određuju se granice `for` petlji za ispis ovog uzorka. Vanjska petlja, označena s `i`, iterira od `1` do `2*N`, budući da uzorak ima visinu od `2*N` redaka. Unutarnja petlja, označena s `j`, iterira od `1` do `2*N-1`, jer uzorak ima širinu od `2*N` stupaca. Spajanjem tih dviju petlji formira se ugniježđena petlja koja omogućuje crtanje svih linija unutar uzorka, uz pravilno praćenje odnosa između i i `j` kako bi se ispravno odredili uvjeti za ispisivanje znaka '#' na odgovarajućim pozicijama. Da bi se došlo do uvjeta za crtanje linija koje prolaze kroz zadane koordinate, potrebno je kombinirati `x` i `y` koordinate (tj. `i` i `j`) svake točke kako bi se dobio zajednički obrazac.

Pri analizi prve linije općenitog oblika `[1,N,N,1]` primjećuje se da zbrajanjem koordinata prve točke (1, N) i druge točke (N, 1) rezultat uvijek iznosi `N+1`. Dakle, uvjet za crtanje ove linije može se zapisati kao `i+j == N+1`.

Kod druge linije općenitog oblika `[N,1,N,N]` je `x` koordinata (`j`) konstantna i jednaka `N`, dok `y` koordinata (`i`) varira od 1 do `N`. Stoga se uvjet za crtanje ove linije može izraziti kao `j==N && i<=N`.

Treća linija, `[N,N,2*N-1,N]` ima `y` koordinatu (`i`) je stalna i jednaka `N`, dok `x` koordinata (`j`) varira od `N` do `2*N-1`. Što se može izraziti uvjetom `j>=N && j<2*N && i==N`.

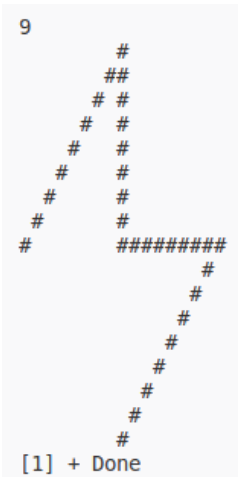
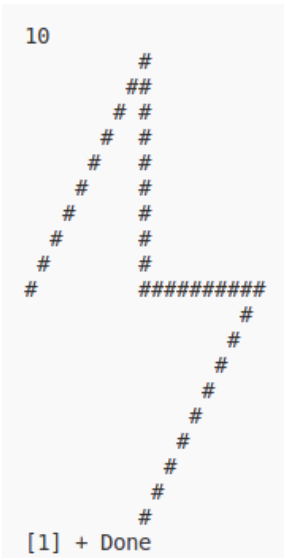
Četvrta linija općenitog oblika `[N,2*N-1,2*N-1,N]` se primjećuje da zbrajanjem koordinata prve točke (N, 2*N - 1) i druge točke (2*N - 1, N) se dobiva rezultat `3*N-1`. Stoga se uvjet za crtanje ove linije može općenito zapisati kao `i+j == 3*N-1`.

Kombiniranjem svih analiziranih uvjeta za linije u uzorku, konačni uvjet unutar `if` strukture može se izraziti pomoću logičkih operatora. Objedinjeni uvjet glasi: `(i+j == N+1) || (j==N && i<=N) || (j>=N && j<2*N && i==N) || (i+j == 3*N-1)`. Prvi uvjet kontrolira crtanje lijeve dijagonale, dok drugi uvjet osigurava crtanje vertikalne linije na sredini uzorka. Treći uvjet omogućuje crtanje horizontalne linije, a posljednji uvjet se brine za crtanje desne dijagonale. U *Ispis programa 4.4* prikazan je konačan oblik koda za ovaj primjer:

```
1  #include <stdio.h>
2  int main(){
3      int i,j, N;
4      scanf("%d",&N);
5      for( i = 1; i<2*N; i++){
6          for(j = 1; j<2*N; j++){
7              if ((j +i == N+1) || (j == N && i<=N)
8              || (j>=N && j<2*N && i==N )
9              || ( j + i == 3*N-1)){
10                 printf("#");
11             }
12             else{
13                 printf(" ");
14             }
15         }
16         printf("\n");
17     }
18     return 0;
19 }
```

Ispis programa 4.4 Primjer 1 umjerene težine: program za ispis uzorka.

Ispis uzorka je dan u *Tablica 4.6*.

N = 8	N =9	N=10
		

Tablica 4.6 Primjer 1 umjerene težine: ispis uzorka za N=8, N=9 i N=10.

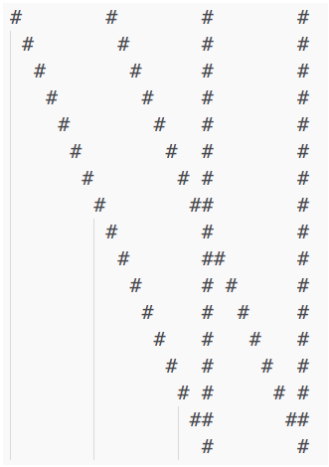
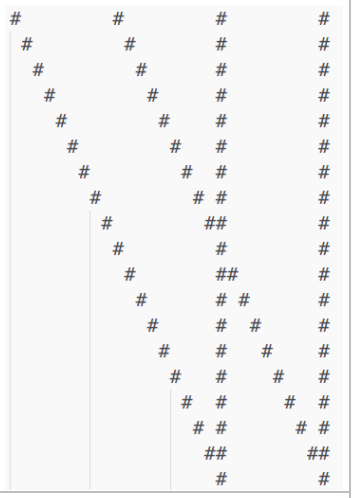
Rotiranjem uzorka može se proširiti broj varijacija zadatka slične težine. Svaki rotirani oblik zadržava osnovne značajke uzorka. Na primjer, početni uzorak može se rotirati za 90, 180 i 270 stupnjeva, čime se dobivaju tri nova zadatka s istim značajkama. Rotirani uzorci prikazani su u *Tablica 4.7*.

Tablica 4.7 Primjer 1 umjerene težine: varijacije rotiranog uzorka.

4.4.2 Primjer 2 umjerene težine

U ovom primjeru (*Tablica 4.8*) analiziraju se dani uzorci za `N` od 8 do 12, a kreće se s analizom uzorka za `N=8`. Uzorak se sastoji od četiri linije, od kojih svaka zahtijeva posebnu analizu kako bi se došlo do općenitih oblika za proizvoljnu vrijednost `N`.

N = 8	N = 9	N = 10
		
N = 11	**N = 12**	
		

Tablica 4.8 Primjer 2 umjerene težine: uzorak za veličine od N = 8 do N = 12.

Za `N=8`, koordinate linija definirane su na sljedeći način:

- [1, 1, 15, 15]
- [8, 1, 22, 15]
- [15, 1, 15, 15]

- [22, 1, 22, 15]

Prva linija ima koordinate `[1,1,15,15]`, što označava da linija ide od točke (1, 1) do točke (15, 15). Kada se uzme vrijednost `N=8`, ove koordinate mogu se općenito izraziti kao `[1,1,2*N-1,2*N-1]`. U ovom slučaju, izraz `2*N-1` predstavlja krajnju točku na desnoj strani uzorka, jer kada je `N=8`, rezultat je `2*8-1=15`. Ovaj opći oblik linije ostaje isti bez obzira na vrijednost `N`.

Druga linija ima koordinate `[8,1,22,15]`, što znači da linija ide od točke (8, 1) do točke (22, 15). U općenitom obliku, za `N=8`, ove koordinate postaju `[N,1,3*N-2,2*N-1]`. Prvo, `x` koordinata početne točke je `N`, a krajnja `x` koordinata iznosi `3*N-2`, što se računa kao 3*8 - 2 = 22. `Y` koordinata krajnje točke iznosi `2*N-1`, što je 2*8 - 1 = 15. Ovaj obrazac se koristi bez obzira na vrijednost `N`.

Treća linija ima koordinate `[15,1,15,15]`, što znači da linija ide od točke (15, 1) do točke (15, 15). Kada se generalizira za proizvoljnu vrijednost N, koordinate postaju `[2*N-1,1,2*N-1,2*N-1]`. U ovom slučaju, izraz `2*N-1` predstavlja stalnu vrijednost `x` koordinate, što za `N=8` daje 2*8-1 = 15. Ova vertikalna linija proteže se od `y` koordinate 1 do `2*N-1`, što osigurava crtanje linije unutar cijele visine uzorka.

Četvrta linija ima koordinate `[22,1,22,15]`, što znači da linija ide od točke (22, 1) do točke (22, 15). U općenitom obliku, koordinate postaju `[3*N-2,1,3*N-2,2*N-1]`, gdje izraz `3*N-2` predstavlja krajnju vrijednost `x` koordinate, što za `N=8` daje 3*8-2 = 22. Ova linija predstavlja krajnju desnu vertikalu koja se proteže kroz cijelu visinu uzorka.

Općeniti oblici svake linije su:

- `[1,1,15,15]`, što u općenitom obliku postaje `[1,1,2*N-1,2*N-1]`
- `[8,1,22,15]`, što u općenitom obliku postaje `[N,1,3*N-2,2*N-1]`
- `[15,1,15,15]`, što u općenitom obliku postaje `[2*N-1,1,2*N-1,2*N-1]`
- `[22,1,22,15]`, što u općenitom obliku postaje `[3*N-2,1,3*N-2,2*N-1]`

Na temelju analize koordinata, određuju se granice for petlji za iscrtavanje ovog uzorka:

- Vanjska petlja i ide od 1 do `2*N`, budući da uzorak ima visinu od `2*N` redaka.

- Unutarnja petlja j ide od 1 do `3*N-2`, jer uzorak ima širinu od `3*N-2` stupaca.

Prva linija općenitog oblika `[1,1,2*N-1,2*N-1]` predstavlja dijagonalu koja se proteže od gornjeg lijevog do donjeg desnog dijela kraja uzorka. Primjećuje se da za ovu liniju vrijedi pravilo da su `x` i `y` koordinate jednake. Dakle, uvjet za crtanje ove linije može se zapisati kao `j==i`.

Za druga liniju općenitog oblika `[N,1,3*N-2,2*N-1]`, analizom se primjećuje da postoji konstantna razlika između `x` i `y` koordinata koja iznosi `N-1`. Stoga se uvjet za crtanje ove linije može izraziti kao `j-i == N-1 && j>=N`.

Treća linija općenitog oblika `[2*N-1,1,2*N-1,2*N-1]` koja predstavlja prvu vertikalnu liniju na desnoj strani uzorka ima stalnu `x` koordinata (`j`) i iznosi `2*N-1`, dok `y` koordinata (`i`) varira od 1 do `2*N-1`. Uvjet ispisa linije je `j==2*N-1 && i>=1 && i<2*N`.

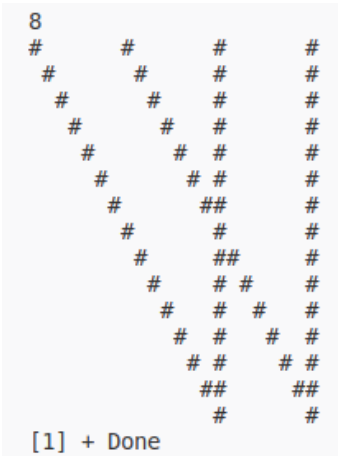
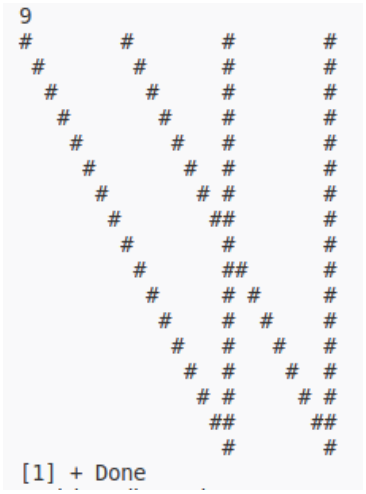
Četvrta linija, `[3*N-2,1,3*N-2,2*N-1]` predstavlja drugu vertikalnu liniju, ali na krajnjem desnom rubu uzorka. Sada, `x` koordinata (`j`) je stalna i iznosi `3*N-2`, dok `y` koordinata (`i`) varira od 1 do `2*N-1`. Stoga se uvjet za crtanje ove linije može zapisati kao `j==3*N-2 && i>=1 && i<2*N`.

Kombiniranjem svih analiziranih uvjeta za crtanje linija u uzorku, konačni uvjet unutar `if` strukture može se izraziti pomoću logičkih operatora. Objedinjeni uvjet glasi: `(j==i) || (j-i==N-1 && j>=N) || (j==2*N-1 && i>=1 && i<2*N) || (j==3*N-2 && i>=1 && i<2*N)`. U *Ispis programa 4.5* prikazan je konačan program za ovaj primjer.

```c
#include <stdio.h>
int main(){
    int i,j, N;
    scanf("%d",&N);
    for( i = 1; i<2*N; i++){
        for(j = 1; j< 3*N -1 ; j++){
            if ((j == i) || (j-i == N-1 && j>=N)
            || (j == 2*N-1 && i >= 1 && i < 2*N)
            || (j == 3*N-2 && i >= 1 && i < 2*N)){
                printf("#");
            }
            else{
                printf(" ");
            }
        }
        printf("\n");
    }
    return 0;
}
```

Ispis programa 4.5 Primjer 2 umjerene težine: program za ispis uzorka.

Nakon što se program pokrene, korisnik unosi veličinu uzorka `N`, a rezultat ispisa za `N=8` i `N=9` prikazan je u *Tablica 4.9*.

N = 8	N = 9
	

Tablica 4.9 Primjer 2 umjerene težine: ispis uzorka za N=8 i N=9.

Ovaj uzorak može se dodatno proširiti rotacijom kako bi se dobilo više verzija zadatka približno jednake težine. U *Tablica 4.10* su prikazane različite varijacije ovog uzorka za `N=8`, koje ilustriraju kako se uzorak može mijenjati zadržavajući slične faktore.

Ovaj uzorak je širok, što ga čini prikladnijim za vježbu ispisa uzoraka, ali ne za provjeru znanja, budući da se zbog njegove veće dimenzije troši više vremena na analizu nego inače.

Tablica 4.10 Primjer 2 umjerene težine: varijacije rotiranog uzorka.

4.4.3 Primjer 3 umjerene težine

Ovaj primjer kombinira nekoliko dijagonalnih linija s dodatnom zanimljivom linijom koja prolazi kroz središnji dio uzorka, ali ne dotiče rubove uzorka. U *Tablica 4.11* su prikazani uzorci za različite vrijednosti `N`, od 8 do 12.

N = 8	N = 9	N = 10
		
N = 11	**N = 12**	
		

Tablica 4.11 Primjer 3 umjerene težine: uzorak za veličine od N=8 do N=12.

Kao i u prethodnim primjerima, analiza koordinata se provodi za zadanu vrijednost `N=8` i koordinate linija definirane su na sljedeći način:

- `[1,8,8,1]`
- `[8,1,15,8]`
- `[8,15,15,8]`
- `[5,4,12,11]`

Prva linija ima koordinate `[1,8,8,1]`, što označava da linija ide od točke (1, 8) do točke (8, 1). Općeniti oblik je `[1,N,N,1]`. Primjećuje se da se za ovu liniju zbrajanjem koordinata prve točke (1, 8) i druge točke (8, 1) dobiva rezultat 9, što je `N+1` (8 + 1 = 9). Zato se uvjet za crtanje ove linije može izraziti kao `j+i == N+1`, jer zbroj indeksa redaka i stupaca uvijek mora biti jednak N + 1.

Druga linija ima koordinate `[8,1,15,8]`, što u općenitom obliku postaje `[N,1,2*N-1,N]`. Pri analizi ove linije primjećuje se da postoji konstantna razlika između `x` i `y` koordinata koja iznosi `N-1`. U primjeru za `N=8`, početna točka linije je (8, 1), a krajnja točka (15, 8). Razlika između `x` i `y` koordinata iznosi 7, što je 8 - 1 = 7. Stoga se uvjet za crtanje ove linije može izraziti kao `j-i == N-1`. Ovaj uvjet osigurava da je razlika između `x` i `y` koordinata stalna.

Treća linija ima koordinate `[8,15,15,8]`, općenitog oblika `[N,2*N-1,2*N-1,N]`. Ovdje se primjećuje da se zbrajanjem koordinata prve točke (8, 15) i druge točke (15, 8) dobiva rezultat 23, što je `3*N-1` (3*8 - 1 = 23). Stoga se uvjet može izraziti kao `j+i == 3*N-1`.

Četvrta linija predstavlja središnju dijagonalnu liniju s koordinatama `[5,4,12,11]`, koja se proteže kroz sredinu uzorka. U općenitom obliku, ove koordinate postaju `[N/2+1,N/2,N+N/2,N+N/2-1]`. U primjeru za `N=8`, početna točka je (5, 4), a krajnja točka (12, 11). Razlika između `x` i `y` koordinata ove linije iznosi 1, stoga se uvjet može zapisati kao `j-i == 1`. Potrebno je dodatno ograničenje u uvjetu linije jer linija ne ide do granica uzorka stoga je potpuni uvjet `j>N/2 && j<=N+N/2 && i>N/2 && i <= N+N/2-1`. Ovi uvjeti osiguravaju da se linija iscrtava samo unutar pravilnog raspona koordinata.

Na temelju analize koordinata, određuju se granice `for` petlji za ispis uzorka:

- Vanjska petlja `i` iterira od 1 do `2*N`, budući da uzorak ima visinu od `2*N` redaka.
- Unutarnja petlja `j` iterira od 1 do `2*N-1`, jer uzorak ima širinu od `2*N-1` stupaca.

Kombiniranjem svih uvjeta za ispis linija, konačni uvjet unutar `if` strukture može se izraziti pomoću logičkih operatora, a objedinjeni uvjet glasi:

```
(j+i == N+1) || (j-i ==N-1) || (j+i == 3*N-1) ||
(j-i == 1 && j>N/2 && j <= N+N/2 && i > N/2   && i <=N+N/2-1).
```

U *Ispis programa 4.6* prikazan je program za ovaj primjer.

```
1  #include <stdio.h>
2  int main(){
3      int i,j, N;
4      scanf("%d",&N);
5      for( i = 1; i<2*N; i++){
6          for(j = 1; j<2*N; j++){
7              if ((j +i == N+1) || (j-i == N-1)
8              || (j + i == 3*N-1)
9              || (j-i == 1 && j>N/2 && j <=N+N/2  && i>N/2 && i<= N+N/2-1)){
10                 printf("#");
11             }
12             else{
13                 printf(" ");
14             }
15         }
16         printf("\n");
17     }
18     return 0;
19 }
```

Ispis programa 4.6 Primjer 3 umjerene težine: program za ispis uzorka.

Nakon što se kod pokrene, korisnik unosi veličinu uzorka `N`, a rezultat ispisa za `N=8` i `N=9` prikazan je u *Tablica 4.12*.

N = 8	N = 9
	

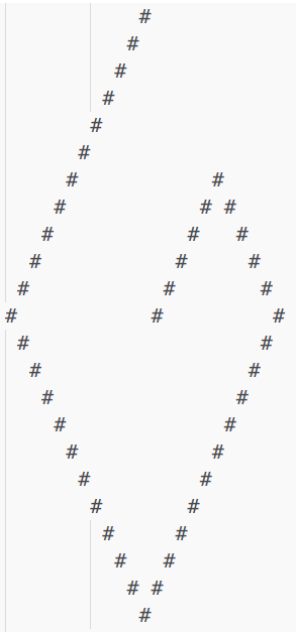
Tablica 4.12 Primjer 3 umjerene težine: ispis uzorka za N=8 i N=9.

Na temelju ovog uzorka moguće je izvesti više varijacija jednostavnom rotacijom uzorka. Svaka od tih varijacija zadržava osnovne karakteristike i težinu izvornog zadatka, ali mijenja orijentaciju ili raspored linija, čime se dobivaju dodatni zadaci slične težine. U *Tablica 4.13* su prikazane varijacije uzorka.

Tablica 4.13: Primjer 3 umjerene težine: varijacije rotiranog uzorka.

4.4.4 Primjer 4 umjerene težine

Sljedeći uzorak (*Tablica 4.14*) predstavlja novi izazov jer zahtijeva pažljivu analizu odnosa između koordinata s ovisno o parnosti broja `N`. U nastavku će se detaljno analizirati kako se te razlike manifestiraju u odnosima koordinata i na koji način se pravilno kreiraju uvjeti za ispisivanje uzorka.

N = 8	N = 9	N = 10
		
N = 11	**N = 12**	
		

Tablica 4.14 Primjer 4 umjerene težine: uzorak za veličine od N = 8 do N = 12.

Uzorak se sastoji od pet linija koje tvore geometrijski oblik slova G. Kao i ranije, početna analiza se provodi za zadanu vrijednost `N=8`.

Prva linija ima koordinate `[1,8,8,1]`, što označava da linija ide od točke (1, 8) do točke (8, 1). Općenit oblike je `[1,N,N,1]`. Zbrajanjem koordinata prve točke (1, 8) i druge točke (8, 1) dobiva se rezultat 9, što je `N+1`. Stoga se uvjet za ispis ove linije može izraziti kao `j+i == N+1`.

Druga linija ima koordinate `[1,8,8,15]`, što u općenitom obliku postaje `[1,N,N,2*N-1]`. Ovdje se primjećuje da razlika između `x` i `y` koordinata ostaje konstantna i iznosi `N-1`. Stoga se uvjet za ispis ove linije može izraziti kao `i-j == N-1`.

Treća linija ima koordinate `[8,15,15,8]`, što u općenitom obliku postaje `[N,2*N-1,2*N-1,N]`. Ovdje se zbrajanjem koordinata prve točke (8, 15) i druge točke (15, 8) dobiva rezultat 23, što je `3*N-1`. Uvjet za ispis ove linije može se izraziti kao `j+i == 3*N-1`.

Četvrta linija ima koordinate `[9,8,12,5]`, što u općenitom obliku postaje `[N+1,N,N+N/2,N/2+1]`. Početna točka linije je (9, 8), a krajnja točka (12, 5). Primjećuje se da zbrajanje koordinata dobiva 17, što je `2*N+1`. Ova linija se ispisuje unutar ograničenog dijela uzorka, pa se uvjet dodatno specificira kao `j+i == 2*N+1 && j >= N+1 && j <= N+N/2 && i >= N && i<=N/2+1`.

Peta linija ima koordinate `[12,5,15,8]`, što u općenitom obliku postaje `[N+N/2,N/2+1,2*N-1,N]`. Ovdje je razlika između `x` i `y` koordinata konstantna i iznosi `N-1`. Dodatno ograničenje uvjetuje ispisivanje linije unutar specifičnih koordinata, pa uvjet glasi: `j-i == N-1 && j >= N+N/2 && i > N/2`.

Na temelju analize koordinata, određuju se granice `for` petlji za iscrtavanje ovog uzorka:

- Vanjska petlja `i` iterira od 1 do `2*N`, budući da uzorak ima visinu od `2*N` redaka.
- Unutarnja petlja `j` iterira od 1 do `2*N`, budući da uzorak ima širinu od `2*N` stupaca.

Kombiniranjem svih analiziranih uvjeta za crtanje linija u uzorku, konačni uvjet unutar `if` strukture može se izraziti pomoću logičkih operatora. Objedinjeni uvjet glasi:

`(j+i == N+1) || (i-j == N-1) || (j+i == 3*N-1) || (j+i == 2*N+1 && j >= N+1 && j <= N+N/2 && i >= N && i <= N/2+1) || (j-i == N-1 && j >= N+N/2 && i > N/2).`

Prvi uvjet (`j+i == N+1`) kontrolira crtanje gornje lijeve linije, dok drugi uvjet (`i-j == N-1`) osigurava ispis donje lijeve linije. Treći uvjet (`j+i == 3*N-1`) omogućava ispis donje desne linije, a posljednja dva uvjeta kontroliraju ispis linija u sredini uzorka.

U *Ispis programa 4.7* prikazan je oblik koda za ovaj primjer:

```c
#include <stdio.h>
int main(){
    int i,j, N;
    scanf("%d",&N);
    for( i = 1; i<2*N; i++){
        for(j = 1; j<2*N; j++){
            if ((j +i == N+1) || (i-j == N-1) || ( j + i == 3*N-1)
            || (j+i == 2*N+1 && j>=N+1 && j<=N+N/2 && i<=N && i>=N/2+1)
            || (j-i== N-1 && j>=N+N/2 && i >N/2)){
                printf("#");
            }
            else{
                printf(" ");
            }
        }
        printf("\n");
    }
    return 0;
}
```

Ispis programa 4.7 Primjer 4 umjerene težine: program za ispis uzorka.

N = 8	N = 9
	

Tablica 4.15: Primjer 4 umjerene težine: ispis uzorka za N=8 i N=9.

Pokretanjem programa provjerava se ispravnost ispisa uzorka za različite vrijednosti veličine N, uključujući N=8 i N=9 u *Tablica 4.15*. Prilikom ispisa uzorka za N=8, rezultati su u skladu s očekivanim uzorkom. Međutim, kod ispisa uzorka za N=9, uočava se neispravnost – uzorak se ne podudara sa zadanim uzorkom na početku zadatka. Ova neispravnost ukazuje na potrebu za dodatnom analizom i prilagodbom uvjeta unutar programa kako bi se pravilno obradili slučajevi kada je N neparan.

Problem koji se javlja kod ispisa uzorka za neparnu vrijednost N, poput N=9, rješava se analizom linije koja nije ispravno ispisana. Za N=9, koordinate male linije u sredini, odnosno četvrte linije, glase [9,9,13,5]. U prethodnom primjeru, za N=8, koordinate te linije bile su [9,8,12,5], što je dalo općeniti oblik [N+1,N,N+N/2,N/2+1]. Međutim, ovaj općeniti oblik ne funkcionira ispravno za neparne vrijednosti N jer se kod neparnih brojeva x1 koordinata mora prilagoditi.

Naime, za neparni N, x1 koordinata više nije jednostavno N+1 u općenitom obliku, već se mora smanjiti za 1, čime postaje N+1-1. Da bi se ovo prilagodilo, koristi se operator modulo %. Modulo operator % omogućava provjeru parnosti ili neparnosti broja. U ovom slučaju, ako je N neparan, modulo vraća vrijednost 1, što omogućuje smanjenje x1 koordinata za 1, dok za parne brojeve nema promjene. Stoga se općeniti oblik za 4. liniju, koji ispravno funkcionira za oba slučaja, može zapisati kao: [N+1-N%2,N,N+N/2,N/2+1]i time se osigurava da x1 koordinata ispravno prilagođava ovisno o parnosti vrijednosti N, što rezultira pravilnim iscrtavanjem uzorka bez obzira na zadanu vrijednost N.

Četvrta linija ima koordinate [9,8,12,5], općenitog oblika [N+1-N%2,N,N+N/2,N/2+1]. U primjeru za N=8, početna točka linije je (9, 8), a krajnja točka (12, 5). Primjećuje se da zbrajanje koordinata daje 17, što je 2*N+1. Međutim, kada je N neparan, početna x koordinata se prilagođava korištenjem izraza N+1-N%2 kako bi se osigurao ispravan ispis linije. Stoga se uvjet za crtanje ove linije generalizira kao j+i == 2*N+1-N%2.

Dodatno, ova linija se crta unutar ograničenog dijela uzorka, pa su granice za x koordinatu (j) također prilagođene: j >= N+1-N%2 && j <= N+N/2, dok y koordinata (i) varira između N i N/2+1. Konačni uvjet za crtanje ove linije može se zapisati

kao: `(j+i == 2*N+1-N%2 && j >= N+1-N%2 && j <= N+N/2 && i >= N && i <= N/2+1).`

Ispravljen kod program je prikazan u *Ispis programa 4.8.*

```c
#include <stdio.h>
int main(){
    int i,j, N;
    scanf("%d",&N);
    for( i = 1; i<2*N; i++){
        for(j = 1; j<2*N; j++){
            if ((j +i == N+1) || (i-j == N-1) || ( j + i == 3*N-1)
            || (j+i == 2*N+1-N%2 && j >= N+1-N%2 && j<= N+N/2 && i <= N && i >= N/2+1)
            || (j-i == N-1 && j>= N+N/2 && i > N/2)){
                printf("#");
            }
            else{
                printf(" ");
            }
        }
        printf("\n");
    }
    return 0;
}
```

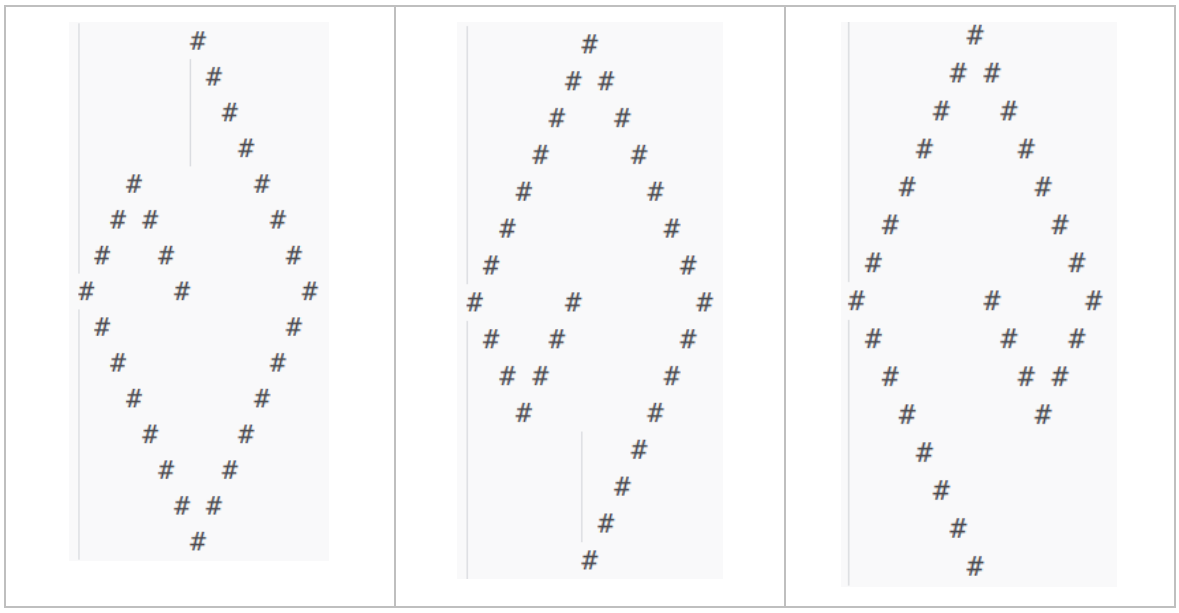
Ispis programa 4.8 Primjer 4 umjerene težine: ispravak programa za ispis uzorka.

U *Tablica 4.16* je prikazana provjera ispravnosti uzorka za `N=8` i `N=9`.

N = 8	N = 9
	

Tablica 4.16: Primjer 4 umjerene težine: točan ispis uzorka za N=8 i N=9.

Ponovno je moguće napraviti varijacije ovog uzorka kako je prikazano u *Tablica 4.17*.

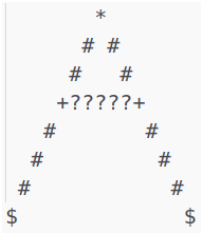
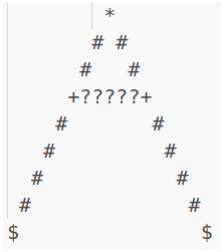
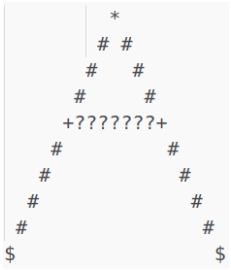
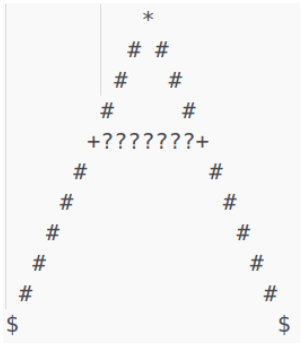
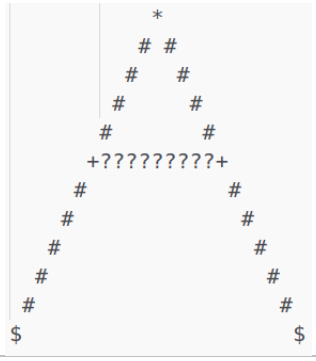


Tablica 4.17: Primjer 4 umjerene težine: varijacije rotiranog uzorka.

4.5 Primjer i analiza uzorka veće težine

U ovom poglavlju prelazi se na teži primjer zadataka koji zahtijeva naprednije analitičko razmišljanje i precizniju primjenu uvjeta grananja. Za razliku od prethodnih primjera, ovaj uzorak ne uključuje velik broj linija ispisanih istim znakom, već sadrži različite simbole koji označavaju sjecišta i krajeve linija. Time se uzorak razlikuje od zadataka idealne težine, zbog čega je njegova težina povećana.

Uzorak je prikazan u *Tablica 4.18*.

N = 8	N = 9	N = 10
		
N = 11	**N = 12**	
		

Tablica 4.18 Primjer veće težine: uzorak za veličine od N=8 do N=12.

Ovaj uzorak sadrži dvije kose linije iscrtane znakom `#`, koje se međusobno križaju i tvore oblik slova A. Uz njih, u sredini se nalazi horizontalna linija sa znakom `?`, koja se razlikuje od kosih linija po svom znaku. Rubovi i spojevi ovih linija označeni su posebnim simbolima — na vrhu se nalazi zvjezdica `*`, na dnu znaci dolara `$`, dok su spojevi središnje linije označeni znakom plusa `+`. Ovi različiti znakovi dodaju težini uzorka i zahtijevaju precizno postavljanje uvjeta za njihov ispis.

Prvi korak u rješavanju ovog zadatka uključuje određivanje koordinata kosih linija:

- Lijeva linija ima koordinate `[1,8,8,1]`, što u općenitom obliku postaje `[1,n,n,1]`, a uvjet za crtanje ove linije je `i+j==N+1`.
- Desna linija ima koordinate `[8,1,15,8]`, što u općenitom obliku postaje `[n,1,2*n-1,n]`, a uvjet za crtanje je `j-i==N-1`.

Nakon definiranja linija, potrebno je odrediti koordinate posebnih simbola koji se koriste za označavanje određenih točaka i središnje linije:

- Zvijezda `*`: Ova točka ima koordinate `[N,1]`, što znači da je njezin uvjet za ispisivanje `j==N i i==1`.

- Središnja linija sa znakom `?`: Ova linija prolazi kroz središnji dio uzorka, a njezine koordinate su `[6,4,10,4]`, što u općenitom obliku postaje `[N/2+2+N%2,N/2,N+N/3,N/2]`. Uvjet za ispis ove linije je `i==N/2` i `j>=N/2+2+N%2` te `j<=N+N/3`.
- Posebni znakovi + koji se nalaze na lijevom i desnom dijelu uzorka ispisuju se prema sljedećim uvjetima:
    - Lijevi znak +: Koordinate su `[5,4]`, odnosno `[N/2+1+N%2,N/2]`, a uvjet za crtanje je `j==N/2+1+N%2` i `i==N/2`.
    - Desni znak +: Koordinate su `[11,4]`, odnosno `[N+N/2-1,N/2]`, a uvjet je `j==N+N/2-1` i `i==N/2`.
- Znak dolara `$` koji se nalazi na dnu uzorka ima sljedeće koordinate:
    - Lijevi dolar: `[1,8]` odnosno `[1,N]`, što se opisuje uvjetom `j==1` i `i==N`.
    - Desni dolar: `[15,8]` odnosno `[2*N-1,N]`, a uvjet je `j==2*N-1` i `i==N`.

Na temelju analize koordinata, određuju se granice `for` petlja za ispis uzorka:

- Vanjska petlja `i` iterira od 1 do `N`, budući da uzorak ima visinu od `N` redaka.
- Unutarnja petlja `j` iterira od 1 do `2*N-1`, budući da uzorak ima širinu od `2*N-1` stupaca.

Unutar petlje koristi se grananje s pomoću `if, else if, else` uvjeta kako bi se ispisali različiti znakovi na odgovarajućim pozicijama unutar uzorka. Grananje počinje provjerom uvjeta za ispis najrjeđeg simbola, koji se nalazi na vrhu uzorka — zvjezdice `*`. Ako su zadovoljeni uvjeti za ispis zvjezdice, ona se ispisuje na vrhu, točno u sredini.

Nakon zvjezdice, provjerava se uvjet za ispis znakova dolara `$` koji se nalaze na donjim rubovima uzorka. Sljedeći u provjeri su uvjeti za znakove plus +, koji se pojavljuju na spojevima središnje linije sa znakom upitnika `?`.

Nakon toga, ispisuje se središnja linija sastavljena od znaka upitnika `?`. Ova linija prolazi vodoravno kroz sredinu uzorka, a za nju se provjerava jesu li zadovoljeni uvjeti za njezin ispis unutar određenog raspona stupaca.

Zadnji dio grananja odnosi se na ispis dviju kosih linija sa znakom #, koje se protežu od gornjih rubova prema dolje, tvoreći oblik, sličan strelici, koji je obrađen kao jednostavni primjer. Ovi uvjeti kontroliraju crtanje dijagonalnih linija i definiraju njihovu točnu poziciju.

Na kraju, uvjet `else` osigurava ispis praznina (razmaka) za sve pozicije koje ne zadovoljavaju niti jedan od prethodnih uvjeta, čime se osigurava pravilno popunjavanje cijelog uzorka.

Konačno rješenje programa je prikazano u *Ispis programa 4.9.*

```c
#include <stdio.h>
int main(){
    int i,j, N;
    scanf("%d",&N);
    for( i = 1; i<=N; i++){
        for(j = 1; j<2*N; j++){
            if(j==N && i==1){
                printf("*");
            }
            else if ((j==1 && i==N) || (j==2*N-1 && i==N)){
                printf("$");
            }
            else if((j==N/2+1+N%2 && i==N/2) || (j==N+N/2-1 && i==N/2)){
                printf("+");
            }
            else if (i == N/2 && j>=N/2+2+N%2 && j<=N+N/3){
                printf("?");
            }
            else if ((i+j == N+1) || (j-i == N-1)){
                printf("#");
            }
            else{
                printf(" ");
            }
        }
        printf("\n");
    }
    return 0;
}
```

Ispis programa 4.9 Primjer veće težine: program za ispis uzorka.

U *Tablica 4.19* je prikazana provjera ispravnosti uzorka za `N=8` i `N=9`.

N = 8	N = 9
8 * # # # # +?????+ # # # # # # $ $ [1] + Done	9 * # # # # +?????+ # # # # # # # # $ $ [1] + Done

Tablica 4.19 Primjer veće težine: ispis uzorka za N=8 i N=9.

Iako uzorak na prvi pogled izgleda jednostavno i sadrži samo tri glavne linije, težina zadatka raste zbog prisutnosti različitih simbola koji nisu uobičajeni za jednostavne primjere. Ovi dodatni simboli, poput zvjezdice `*`, znaka dolara `$`, pluseva `+` i središnje linije sa znakovima upitnika `?`, zahtijevaju preciznu analizu koordinata i posebno definirane uvjete za njihovo pravilno iscrtavanje. Svaki simbol ima svoje specifične pozicije i uvjete unutar uzorka, pa je važno pažljivo postaviti `if-else` uvjete kako bi se svi elementi ispisali na odgovarajućim mjestima.

Studenti koji nauče postavljati uvjete ispisa koristeći prvenstveno logičke operatore često neće odmah primijetiti kako u ovom primjeru to neće biti moguće ostvariti unutar jednog `if` uvjeta. Umjesto toga, potrebno je koristiti `if, else if, else` strukturu kako bi se ispravno ispisali svi simboli. Redoslijed uvjeta također je ključan za ispravan ispis uzorka. Ako u prvi `if` uvjet postavimo crtanje znaka `#` preko cijelih linija pod kutom, a zatim u sljedeće `else if` uvjete postavimo druge simbole, može doći do neispravnog ispisa uzorka.

Primjerice, ako uzmemo znakove zvjezdice, plusa i dolara, te u prvom `if` grananju postavimo uvjet za ispis znaka `#` na lijevoj kosoj liniji, kad petlja dođe do koordinata na kojima bi se trebala ispisati zvjezdica, uvjet za ispis linije `#` također će biti zadovoljen, te će se umjesto zvjezdice ispisati znak `#`. Isto se događa i sa znakom plus, jer se i on pojavljuje na koordinatama kroz koje prolazi kosa linija. Ako uvjet za liniju ima prednost, ispisat će se znak `#`, a ne plus. Slična situacija nastaje i sa znakom dolara, gdje se znak `#` ispisuje na mjestu gdje bi trebao biti dolar, ako je uvjet za liniju prvi. Zbog toga je važno prvo postaviti uvjete za najrjeđe i posebne simbole, kao što su zvjezdica, dolar i plus, posebno ako se ti simboli nalaze na linijama koje je jednostavno definirati, jer bi u suprotnom uvjeti za ispis linija mogli prebrisati te simbole. Zatim slijede uvjeti za ispis linija sa znakom `#`, a na kraju se postavlja `else` uvjet koji ispisuje bjeline.

Iako se ovaj uzorak može dodatno podijeliti na manje dijelove, gdje bi se ispis dijagonalnih linija sa znakom `#` podijelio u dva dijela, to bi značajno produžilo vrijeme rješavanja zadatka, jer bi bilo potrebno analizirati veći broj pravaca. Ipak, takav pristup može biti koristan kao vježba za studente, jer ih prisiljava na dublju analizu i razumijevanje uvjeta.

Zaključno, iako uzorak može djelovati jednostavno, zbog prisutnosti različitih simbola i većeg broja posebnih uvjeta za ispis, ovaj zadatak predstavlja dobar primjer težeg

problema koji zahtijeva dublju analizu i pažljivu organizaciju logičkih uvjeta. Ovakvi zadaci potiču studente na detaljnije promišljanje, što doprinosi jačanju njihovih vještina u postavljanju uvjeta i pravilnoj uporabi kontrolnih struktura. Rješavanjem ovakvih zadataka, studenti razvijaju vještine i znanja koja su ključna za daljnje savladavanje naprednijih programskih zadatka na raznim kolegijima programiranja.

4.6 *Analiza rješenja i česte pogreške*

1.6.1. Uobičajeni postupak rješavanja zadataka

Rješavanje zadataka koji uključuju ispis uzoraka s pomoću programskih petlji u programskoj jeziku C obično započinje analizom strukture samog uzorka. Prvi korak u tom procesu je vizualizacija uzorka i njegovo razbijanje na pojedinačne linije ili segmente. Studenti najprije trebaju prepoznati uzorak kao skup linija koje se mogu opisati s pomoću koordinata, čime započinje analiza odnosa između tih koordinata. Proučavanjem početnih i krajnjih točaka svake linije dolazi se do općenitog oblika koji definira te linije za bilo koji zadani broj `N`.

Nakon definiranja koordinata, sljedeći korak je određivanje granica `for` petlji koje će iterirati kroz redove i stupce uzorka. Ovdje je ključno pravilno identificirati koje petlje kontroliraju `y` koordinate, a koje `x` koordinate. U uobičajenom postupku, vanjska petlja (koja obično iterira kroz redove) obuhvaća `y` koordinate, dok unutarnja petlja (koja iterira kroz stupce) obuhvaća `x` koordinate. Granice ovih petlji definiraju dimenzije uzorka i određuju raspon unutar kojeg se linije crtaju.

Nakon definiranja granica petlji, slijedi formulacija uvjeta unutar `if` grananja koji određuju kada će se ispisati znak `#`, a kada bjelina. Studenti pristupaju identifikaciji uzorka u vrijednostima koordinata. Na primjer, u uzorcima s dijagonalnim linijama, često se zbrajaju ili oduzimaju `x` i `y` koordinate kako bi se pronašao zajednički obrazac. Ako je uzorak teži i sadrži više linija koje se preklapaju ili presijecaju, potrebno je postaviti logičke uvjete koji kombiniraju više kriterija, obično korištenjem operatora poput `&&` (AND) ili `||` (OR).

Kako bi se osiguralo da rješenje bude primjenjivo za različite vrijednosti `N`, testiranje se obično provodi prvo s parnim, a zatim s neparnim vrijednostima `N`. Uobičajen postupak uključuje počinjanje s jednostavnijim, parnim brojevima poput `N=8`, budući da oni često imaju simetrične uzorke. Međutim, kod testiranja s neparnim brojevima poput `N=9`, mogu

se pojaviti nesavršenosti koje zahtijevaju dodatnu analizu i prilagodbu uvjeta unutar programa.

Kroz cijeli postupak rješavanja zadataka, studenti trebaju često provjeravati ispravnost rezultata analiziranjem generiranog uzorka i uspoređivanjem s očekivanim uzorkom. Ako uzorak nije ispravno ispisan, vraćaju se na početnu analizu koordinata i granica kako bi prepoznali gdje je došlo do greške. Tek nakon što se svi uvjeti pravilno definiraju, uzorak će biti točno ispisan za sve vrijednosti N.

Ovaj sustavni pristup omogućuje studentima da logički analiziraju svaki dio zadatka, postupno prilagođavajući rješenje kako bi pokrili sve moguće scenarije. Također je važno napomenuti da je opisani pristup rješavanja zadataka samo jedan od mogućih načina kako se zadaci s ispisivanjem uzoraka mogu riješiti. Iako je ovaj pristup sustavan i može biti vrlo učinkovit, studenti imaju slobodu istraživati i otkrivati vlastite metode koje mogu dovesti do jednako dobrih ili čak boljih rješenja. U uvodnim zadacima korisno je potaknuti studente da samostalno dođu do rješenja, a nakon toga im se može prikazati standardizirani postupak kako bi usporedili svoje metode s uobičajenim pristupom. Time se omogućuje fleksibilnost u razmišljanju i razvijanje kreativnog pristupa problemima.

1.6.2. Česte pogreške i izazovi

Tijekom analize 56 netočnih rješenja studenata u zadacima s ispisom uzoraka iz jedne akademske godine kolegija „Računarstvo i praktikum“ (to je od ukupno 154 rješenja - preostalih 98 rješenja je točno), uočene su razne vrste pogrešaka koje su dovele do pogrešnog rješavanja zadataka. Analiza je provedena kako bi se utvrdile najčešće pogreške studenata. Ove pogreške obično se svode na nepravilnosti u definiranju uvjeta, korištenju operatora, granicama petlji, ili neispravnoj logici unutar `if-else` grananja.

Sistematizacija grešaka podijeljena je u osam glavnih kategorija. Kao što je prikazano na *Slika 4.6*, najučestalija vrsta pogrešaka odnosi se na veliku nepreciznost uvjeta, koja je prisutna u 51 od 56 rješenja zadataka, dok su najrjeđe greške bile vezane uz malu nepreciznost uvjeta uzorka, što je zabilježeno kod 1 od analiziranih 56 krivih rješenja.

Konkretno, *velika nepreciznost* koja je javlja u 51 rješenju uključuje: potpuno pogrešan izgled uzorka, nedostatak dijelova uzorka, duplicirane uvjete, uvjete koji nisu definirani u ovisnosti varijable N, pogrešnu lokaciju ispisa novog reda, pogrešnu definiciju `scanf` i `printf` funkcija i neodređivanje granica linija.

Sljedeća greška koja se javlja u 21 rješenju, odnosi se na *parnost varijable N* te ona obuhvaća postavljanje pogrešnog uvjeta za parnost brojeva ili nedostatak uvjeta za parne ili neparne brojeve.

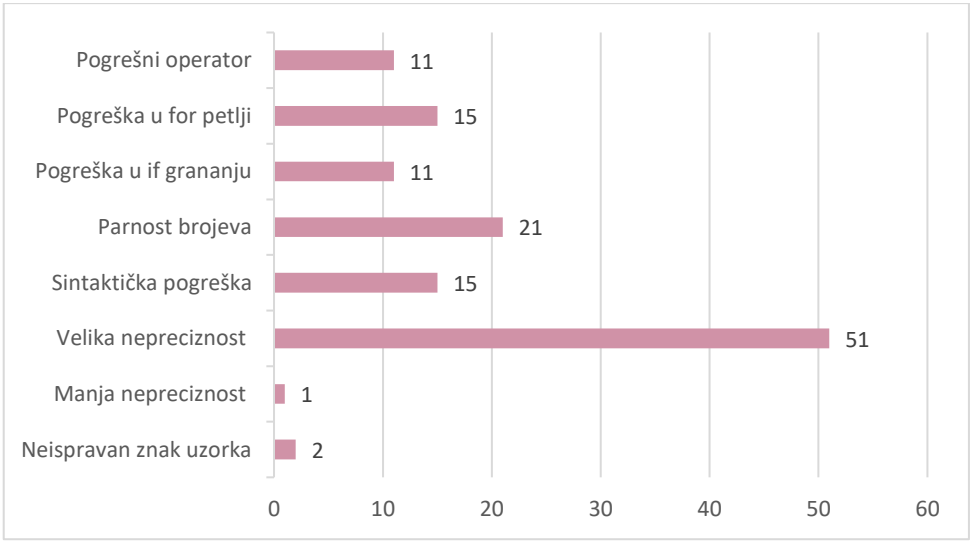
Nadalje, u *sintaktičku pogrešku* koja je uočena u 15 rješenja ubrajamo: nedovršeni uvjet gdje logički operatori ostaju 'visjeti' u nizanju uvjeta, odsustvo znaka točke-zareza na kraju naredbe i aritmetičkih operatora (izraz 2n umjesto `2*n`) te zagrade koje nisu zatvorene.

Pogreška u for petlji se također javlja u 15 rješenja te se odnosi na pogrešno postavljene granice petlje i pogrešno definiranje `for` petlje.

Zatim *pogreška u if grananju* koja se javlja u 11 pogrešnih rješenja obuhvaća pogrešnu definiciju grananja kao i za pogrešni redoslijed `if`, `else if` i `else` dijelova.

Pogreška koja se odnosi na *operatore* se također javlja u 11 od 56 rješenja gdje ubrajamo zamjenu operatora, na primjer pridruživanja i usporedbe ili *modulo* s operatorom dijeljenja.

Dvije najrjeđe vrste pogrešaka su *neispravni znak uzorka* koji je uočen u 2 rješenja što znači da se u ispisu uzorka koristi pogrešan znak i rješenje se ne prihvaća, te *manja nepreciznost* postavljanje uvjeta zabilježena u 1 rješenju gdje dolazi do pomicanja cijelog ispravnog uzorka za jedno mjesto udesno. Iako je sam oblik uzorka ispravan, rješenje se ne prihvaća jer uzorak nije ispisan zadanim znakom, kako se program ocjenjuje strojno.



Slika 4.6 Graf prikaza broja pogrešaka koje su uočene tijekom analize uzorka 56 netočnih rješenja.

Česte pogreške se mogu izbjeći dodavanje uvjeta unutar `if` grananja korak po korak i testiranje svakog uvjeta zasebno, osobito kod težih uzoraka. Ovaj postupak omogućava studentima da postupno grade ispravan kod, tako što nakon svakog dodanog uvjeta isprintaju trenutni ispis uzorka i usporede ga sa zadanim ciljem. Ako dio uzorka odgovara očekivanom, tada je uvjet ispravan i može se dodati sljedeći uvjet za sljedeću liniju. Na taj način, studenti mogu biti sigurni da su svi uvjeti na kraju valjani i da ispravno funkcioniraju zajedno. Kod težih uzoraka, ako se svi uvjeti ubace odjednom, postoji rizik da će neki od njih biti pogrešno postavljeni, što može rezultirati dugotrajnim analizama i otkrivanjem izvora problema. Podjelom problema na manje dijelove i postupnim testiranjem, ovaj se rizik smanjuje, čime se štedi vrijeme. Takav pristup posebno je učinkovit u situacijama poput provjera znanja ili ispita, gdje je vrijeme ograničeno, pa je od ključne važnosti brzo doći do ispravnog rješenja bez nepotrebnog gubljenja vremena na dodatnu analizu grešaka.

Ova analiza 56 netočnih rješenja studenata i čestih grešaka ukazuje na važnost temeljite provjere rješenja s različitim vrijednostima N, kako bi se postiglo univerzalno rješenje koje je kompaktno i pokriva sve moguće scenarije.

Nakon analize pogrešnih rješenja se može zaključiti kako određeni ključni koncepti i dalje predstavljaju izazov studentima čak i na provjerama znanja. Identifikacijom najčešćih pogrešaka, kao što su nepravilno definirani uvjeti, pogreške u `for` petljama i sintaktičke greške, postaje jasno koji aspekti programiranja zahtijevaju dodatnu pažnju i vježbu. Ova analiza može omogućiti profesorima da prepoznaju gdje su studentima potrebne dodatna objašnjenja, čime se može unaprijediti proces učenja i poboljšati uspjeh studenata u rješavanju zadataka.

U sljedećem poglavlju, fokus će biti na klasifikaciji zadataka prema težini, čime se dobiva bolji uvid u različite vrste uzoraka i razine težine koje zadaci mogu imati.

5. Klasifikacija zadataka za ispis uzoraka

Zadaci s ispisivanjem uzoraka mogu se klasificirati prema različitim stupnjevima težina. Ova klasifikacija pomaže u razumijevanju koji zadaci su prikladni za početnike, a koji za iskusnije studente, omogućujući strukturirano vježbanje i procjenu znanja.

5.1 Klasifikacija prema težini (subjektivnom procjenom)

U zadacima s ispisivanjem uzoraka, težina zadatka ne ovisi samo o vrsti linija koje čine zadani uzorak, već i o nizu drugih čimbenika koji utječu na poteškoću pri ispisivanju pravilnog uzorka. Kako bi se omogućila detaljna procjena težine svakog zadatka, razvijena je tablica težina koja sadrži ključne faktore.

Kada se procjenjuje težina zadatka, prvi korak je odrediti osnovni broj linija koje čine uzorak. Zadaci s manje od tri linije dobivaju početnu težinu od 1, dok zadaci s više od tri linije dobivaju težinu 1,4. Ovaj osnovni faktor težine služi kao početna vrijednost, na koju se potom dodaju dodatni faktori koji utječu na ukupnu težinu zadatka. Svaki dodatni faktor koji utječe na težinu zadatka množi se s prethodnom vrijednošću, čime se dobiva konačna težina zadatka. Faktori težine i njihove odgovarajuće vrijednosti prikazani su u *Tablica 5.1*.

R.br.	Faktor težine	Vrijednost težine
1	Horizontalna ili vertikalna linija	1
2	Linija pod kutom	1,3
3	Kosa linija koja ne dira rubove uzorka	1,4
4	Uzorak ne počinje u gornjem lijevom dijelu	1,2
5	Broj spojeva manji od 4	1
6	Broj spojeva veći od 4	1,2
7	Manje od 3 linija	1
8	Više od 3 linija	1,4
9	Više od 1 vrste znaka u uzorku	2,5
10	Parni i neparni n	1,2

Tablica 5.1 Prikaz faktora i vrijednosti težina faktora.

Težinski faktori prikazani u *Tablica 5.1* određeni su kako bi se kvantificirao utjecaj pojedinih elemenata na ukupnu težinu zadatka. Svaki faktor dodaje određenu težinu na osnovu toga koliko povećava zahtjevnost rješavanja zadatka. Nisu korišteni izrazito visoki iznosi težina jer, unatoč raznolikosti uzoraka, razlike u težini između pojedinih faktora nisu

toliko značajne da bi zahtijevale veće vrijednosti. Cilj ovakvog sustava je da osigura pravednu procjenu težina.

Primjerice zadatak na *Slika 5.1*, ima:

- više od 3 linije (težina 1,4)
- linije pod kutom (težina 1,3)
- kosu liniju koja ne dira rubove uzorka (težina 1,4)
- uzorak ne počinje u gornjem lijevom dijelu (težina 1,2)
- broj spojeva manji od 4 (težina 1)

Slika 5.1 Primjer uzorka u računanju težine zadatka.

Konačna težina tog zadatka računa se kao umnožak svih tih faktora:

```
Težina = 1,4 * 1,3 * 1,4 * 1,2 * 1 = 3,1
```

Ova metoda omogućuje precizno ocjenjivanje težine svakog zadatka na temelju njegovih specifičnih karakteristika.

Važna je i subjektivna procjena težine, koja uzima u obzir potrebnu analizu za točno rješavanje zadatka unutar 20 minuta, uz dopuštenje za pokretanje i isprobavanje programa. Zadaci koji imaju više različitih simbola ili neobičnih kombinacija linija i simbola mogu premašiti taj vremenski okvir, ali to također ovisi o sposobnosti studenata da brzo analiziraju zadatak.

Na temelju rezultata izračuna težina osmišljenih zadataka prikazanih u tablici u Dodatak Pregled uzoraka, može se donijeti nekoliko ključnih zaključaka u vezi s

klasifikacijom zadataka prema težini. Ocjene težine kreću se od 1 do 3,9, pri čemu zadaci s nižom ocjenom predstavljaju jednostavnije uzorke, a oni s višom ocjenom zahtjevnije.

Zadaci s težinom do 1,8 klasificiraju se kao jednostavni uzorci, idealni za početak. Ovi zadaci pružaju solidnu osnovu za razumijevanje osnovnih principa programiranja i omogućuju studentima da se postupno upoznaju s ispisom uzoraka i osnovnim strukturama, poput `for` petlji i grananja.

Zadaci s ocjenom težine u rasponu između 2 i 3,7, predstavljaju idealne zadatke za vježbu i provjere znanja. Ovi zadaci imaju uravnoteženu težinu i omogućuju studentima da testiraju i unaprijede svoje znanje kroz rješavanje težih primjera.

Zadaci s ocjenom težine iznad 3,7, su vjerojatno prezahtjevni za kraće provjere znanja koje su ograničene na 20 minuta rješavanja. Ovi zadaci zahtijevaju dublju analizu, naprednije vještine i više vremena za rješavanje, što ih čini prikladnijima za vježbu, ali ne i za standardne ispitne provjere.

Dakle, sustavna procjena težine zadataka pomaže profesorima u odabiru zadataka koji su najbolje prilagođeni različitim razinama težina, što je posebno važno u slučaju potrebe za velikim brojem zadataka.

6. Zaključak

Nakon uvoda, u drugom poglavlju dan je kratki pregled osnovnih koncepata operatora, grananja i petlji u programskom jeziku C, a u trećem poglavlju prikazane su poteškoće [12] u razumijevanju petlji i grananja te različite metode koje pomažu tijekom predavanja i vježbe. Metoda programiranja uživo istaknuta je kao korisna zbog smanjenja kognitivnog opterećenja kod studenata [15] i povećanja angažmana studenata tijekom predavanja. U četvrtom poglavlju, osmišljeni su i analizirani primjeri interaktivnih zadataka ispisa uzoraka, pri čemu je zaključeno da, iako neki uzorci na prvi pogled izgledaju jednostavno, zahtijevaju temeljitu analizu koja produbljuje razumijevanje kontrolnih struktura. U skladu s tim, na samom kraju ovog rada u poglavlju Dodaci nalaze se primjeri pregleda uzoraka za spomenute zadatke. Usto, analizirana su pogrešna studentska rješenja koja se također nalaze u poglavlju Dodataka i sistematizirane su česte pogreške. Peto poglavlje donosi subjektivnu klasifikaciju zadataka za ispis uzoraka prema težini, koja omogućava praktičnu procjenu prikladnosti zadataka za vježbu ili provjere znanja.

Iako se u ovom radu koristi programski jezik C, opisani pristup zadacima i rješavanju lako je prilagodljiv i za druge programske jezike.

Dodaci

Dodatak Pregled uzoraka

R.BR	Prikaz uzorka N = 8	Manje od 3 linije	Više od 3 linije	Horizontalna ili vertikalna linija	Linija pod kutom	Kosa linija koja ne dira rubove uzorka	Uzorak ne počinje u gornjem lijevom dijelu	Broj spojeva manji od 4	Broj spojeva veći od 4	Neparan drugači ji	Više od 1 vrste znaka u uzorku	OCJENA TEŽINE
1		1		1				1				1
2		1			1,3		1,2	1				1,6
3			1,4	1	1,3			1				1,8
4			1,4	1	1,3			1				1,8
5			1,4	1	1,3			1				1,8

6			1,4	1	1,3		1,2	1				2,2
7		1		1	1,3			1				1,3
8		1		1	1,3			1				1,3
9		1		1	1,3		1,2	1		1,2		1,9
10		1		1	1,3			1				1,3

11			1,4		1,3	1,4	1,2	1				3,1
12			1,4		1,3	1,4	1,2	1				3,1
13			1,4		1,3	1,4	1,2	1				3,1
14			1,4		1,3	1,4	1,2	1				3,1

15			1,4		1,3	1,4	1,2	1		1,2		3,7
16			1,4		1,3	1,4	1,2	1				3,1
17			1,4		1,3	1,4	1,2	1		1,2		3,7
18			1,4		1,3	1,4	1,2	1				3,1

19			1,4		1,3	1,4	1,2	1		1,2		3,7
20			1,4		1,3	1,4	1,2	1		1,2		3,7
21			1,4		1,3	1,4	1,2	1		1,2		3,7
22			1,4	1	1,3	1,4	1,2	1		1,2		3,7

23			1,4	1	1,3		1,2	1				2,2
24			1,4	1	1,3		1,2	1				2,2
25			1,4	1	1,3		1,2	1				2,2
26			1,4	1	1,3		1,2	1				2,2

27			1,4	1	1,3	1,4	1,2	1				3,1
28			1,4	1	1,3	1,4	1,2	1				3,1
29			1,4	1	1,3	1,4	1,2	1				3,1
30			1,4	1	1,3	1,4	1,2	1				3,1

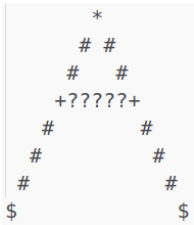
31			1,4		1,3	1,4	1,2	1		1,2		3,7
32			1,4		1,3	1,4	1,2	1		1,2		3,7
33			1,4		1,3	1,4	1,2	1		1,2		3,7
34			1,4		1,3	1,4	1,2	1		1,2		3,7
35			1,4	1	1,3	1,4	1,2	1		1,2		3,7

36			1,4	1	1,3	1,4	1,2	1		1,2		3,7
37			1,4	1	1,3	1,4	1,2	1		1,2		3,7
38			1,4	1	1,3	1,4	1,2	1		1,2		3,7
39			1,4		1,3	1,4	1,2	1		1,2		3,7

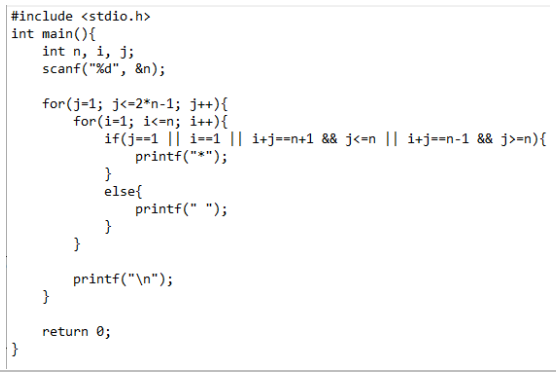
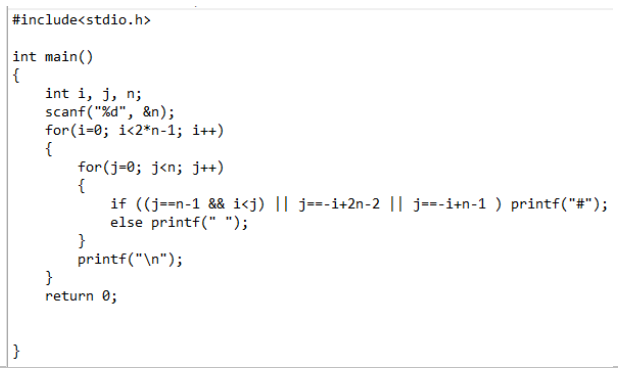
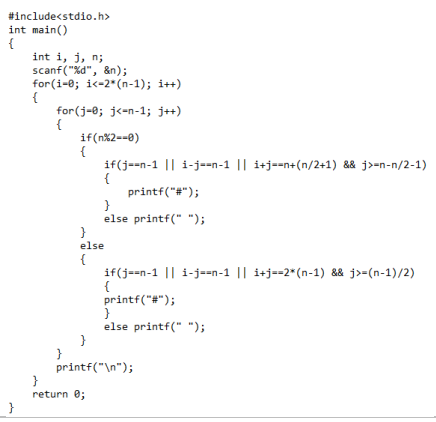
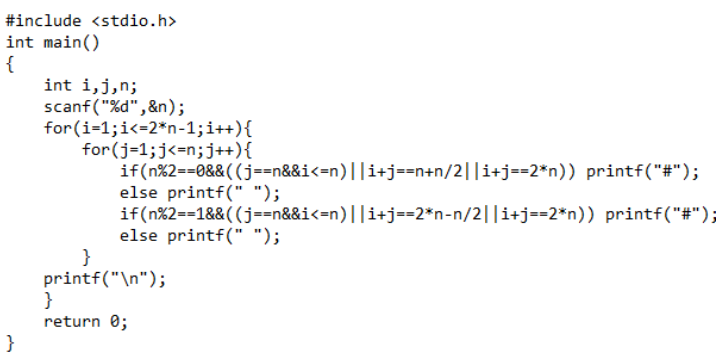
40			1,4		1,3	1,4	1,2	1		1,2		3,7
41	N=8  N=9 	1			1,3	1,4	1,2	1		1,2		2,6
42	N=8  N=9	1			1,3	1,4		1		1,2		2,2

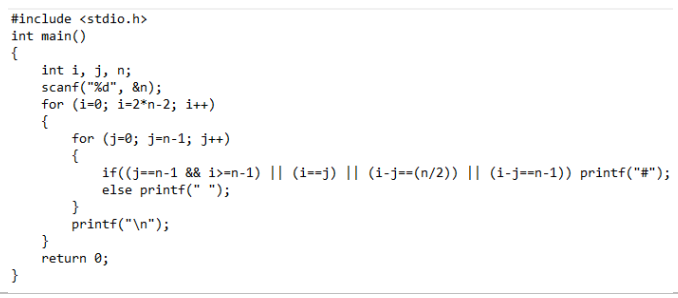
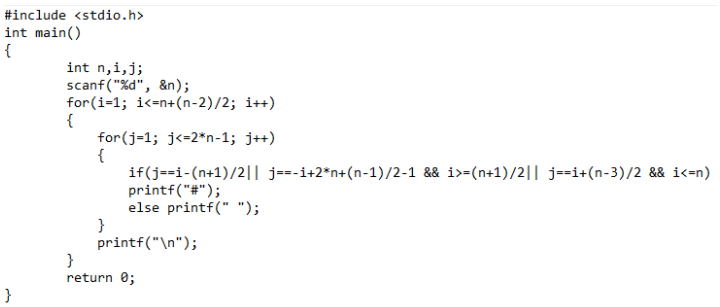
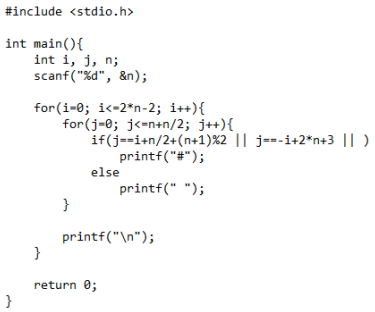
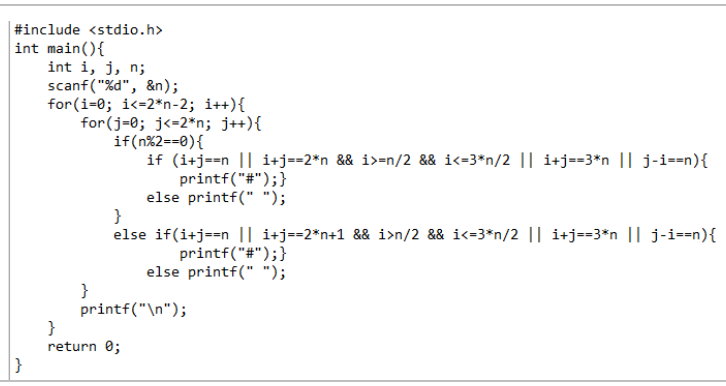
												
43	N=8  N=9 	1			1,3	1,4	1,2	1		1,2		2,6
44	N=8  N=9	1			1,3	1,4		1		1,2		2,2

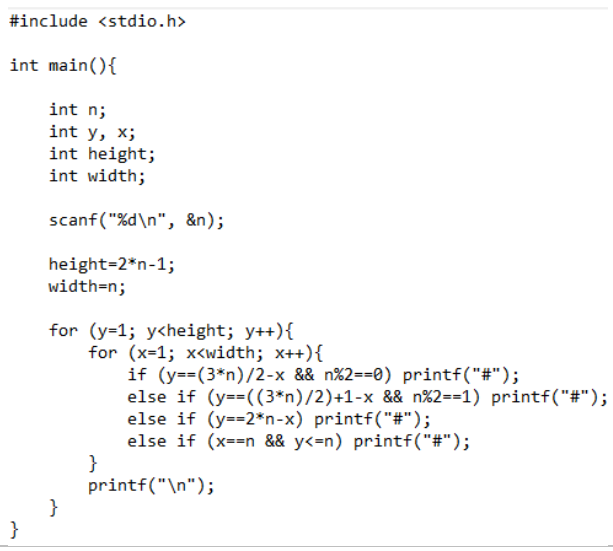
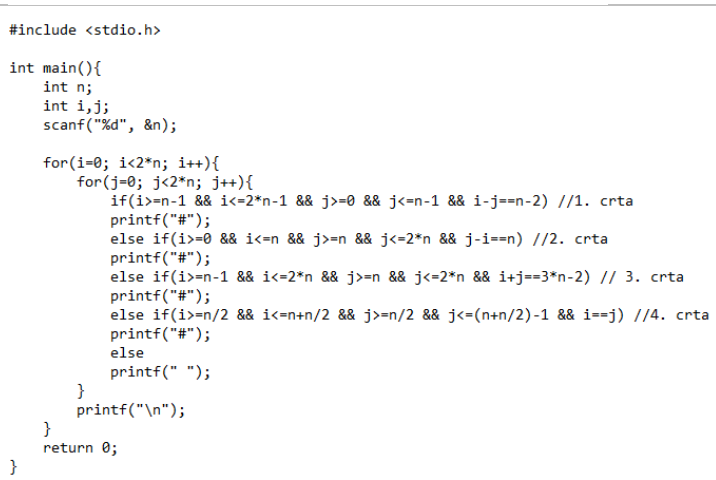
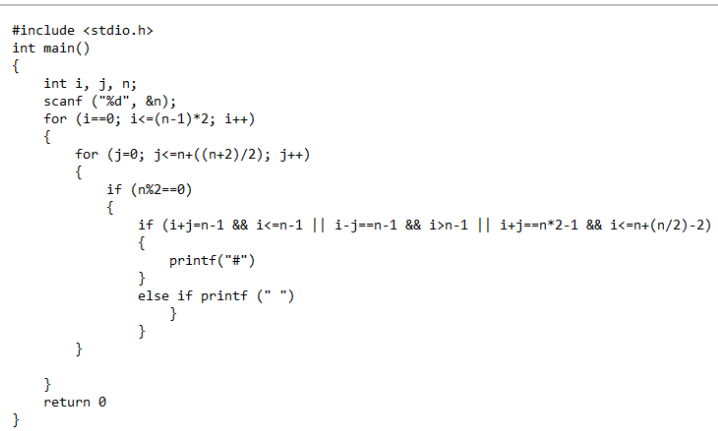
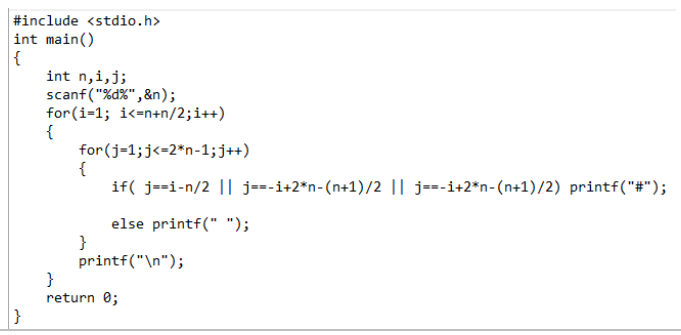
												
45		1		1	1,3		1,2	1				1,6
46		1		1	1,3		1,2	1				1,6
47		1		1	1,3		1,2	1				1,6

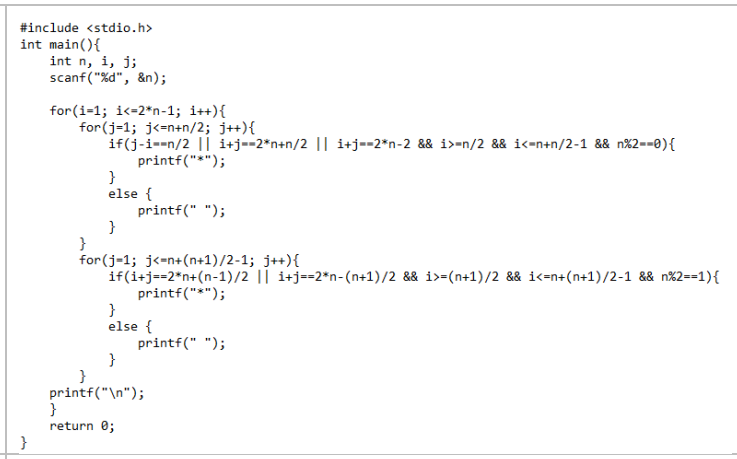
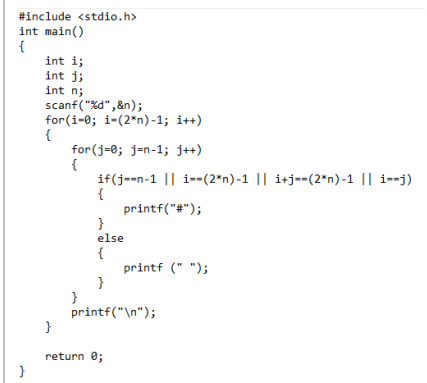
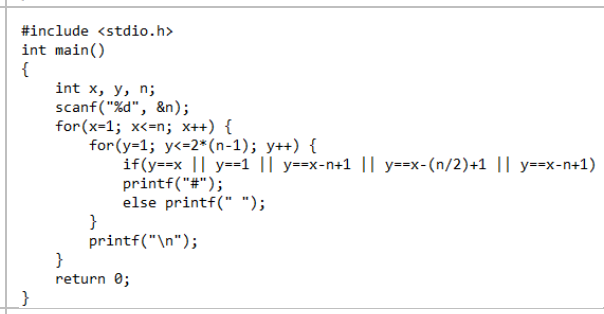
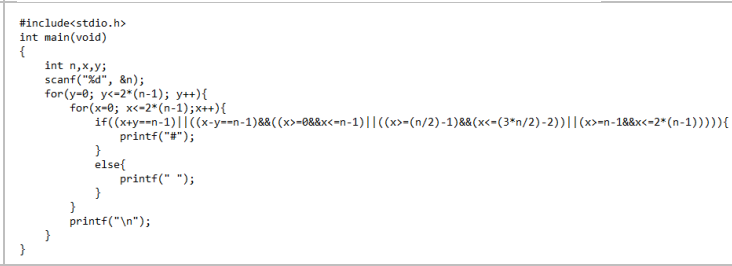
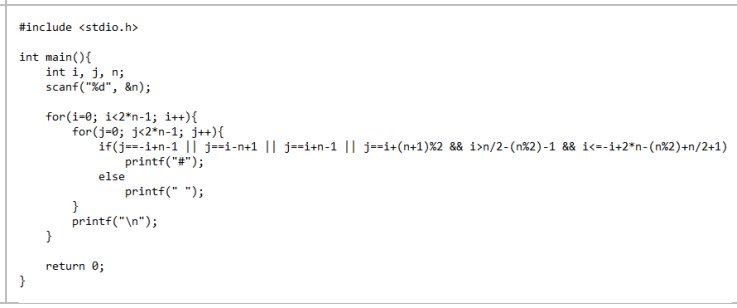
48		1		1	1,3		1,2	1				1,6
49		1		1	1,3		1,2	1			2,5	3,9

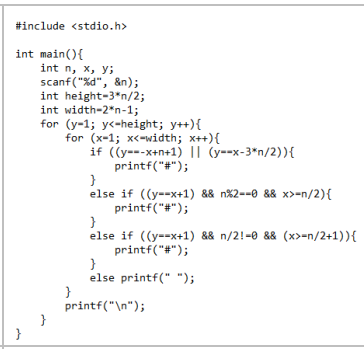
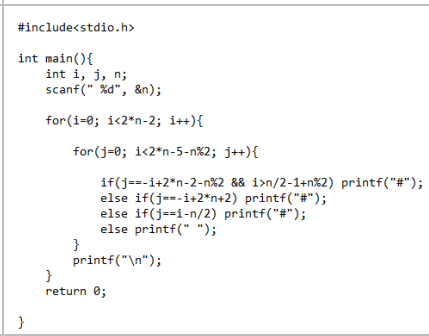
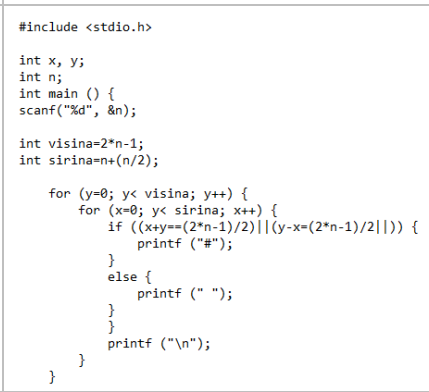
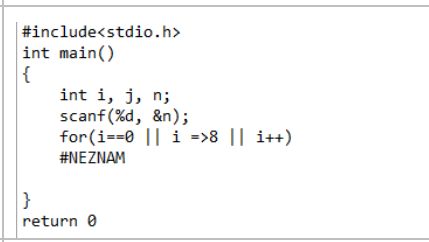
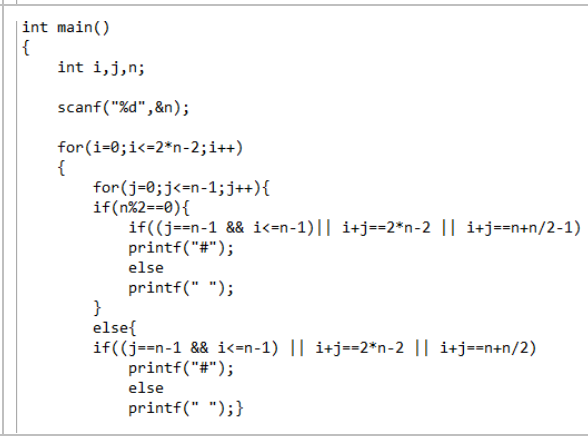
Dodatak Pregled analiziranih rješenja

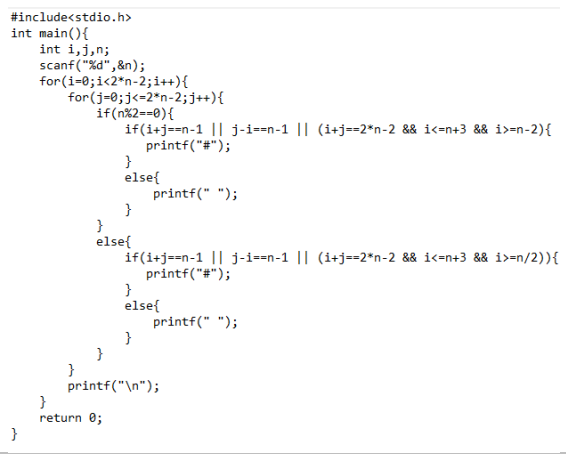
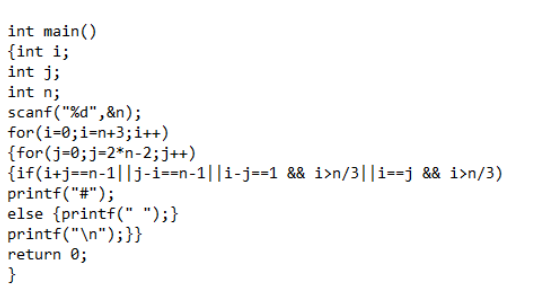
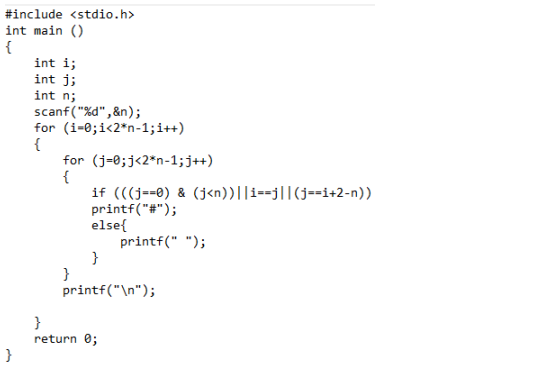
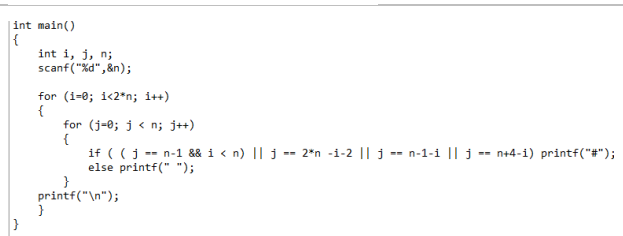
R.BR	PROGRAM	GREŠKE U PROGRAMU
1		- neispravan znak uzorka - v.n. – krivi uvjet
2		- sintaktička greška - v.n. – krivi uvjet
3		- v.n. – krivi uvjet - parnost brojeva
4		- v.n. –krivi uvjet - if grananje – pogrešno korištenje

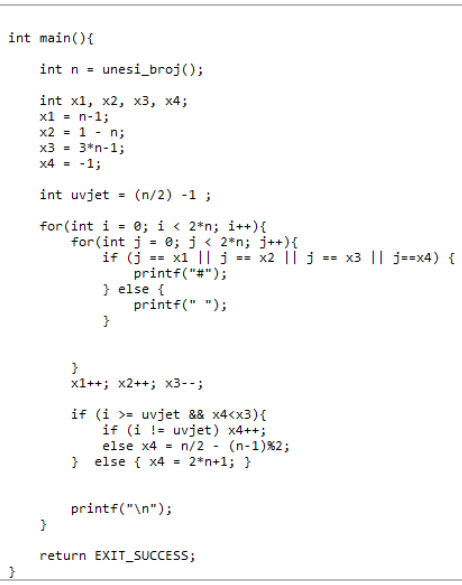
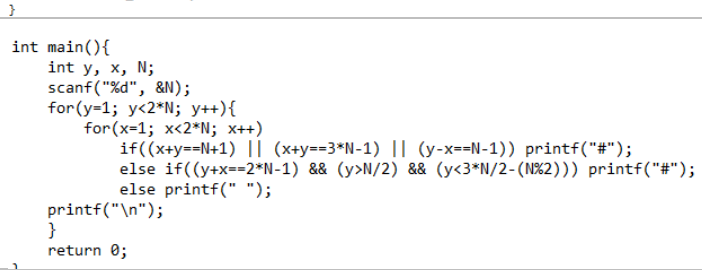
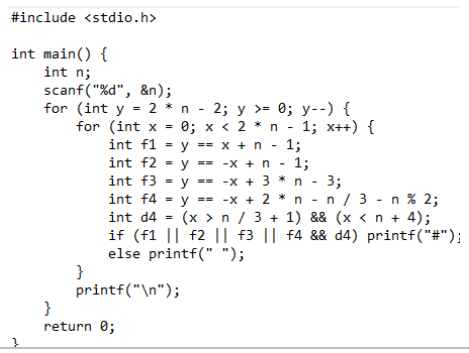
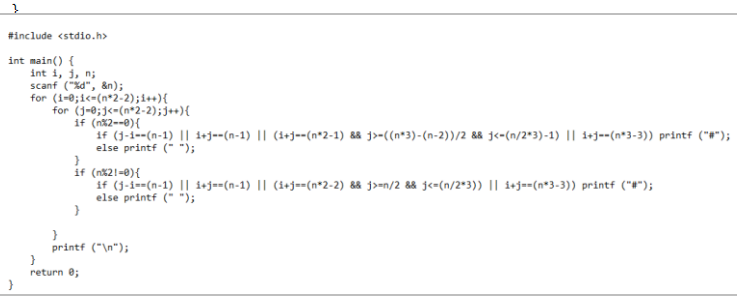
5		- for petlja – definicija - pogrešni operator (pridruživanja umjesto relacijskog)
6		- v.n. – krivi uvjet
7		- v.n. – krivi uvjet - if grananje – definicija - sintaktička greška
8		- v.n. –krivi uvjet - parnost brojeva

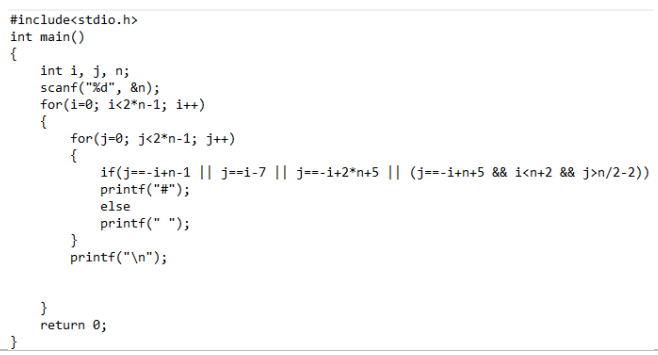
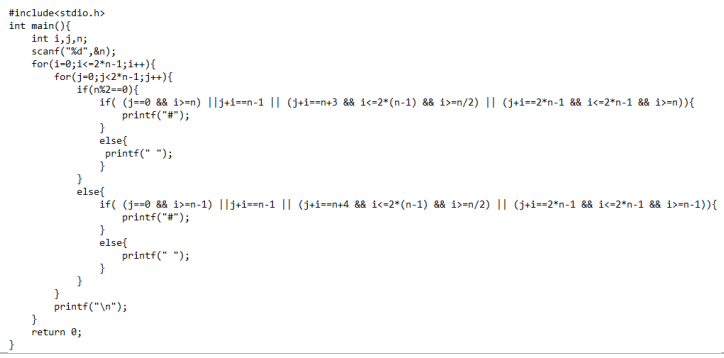
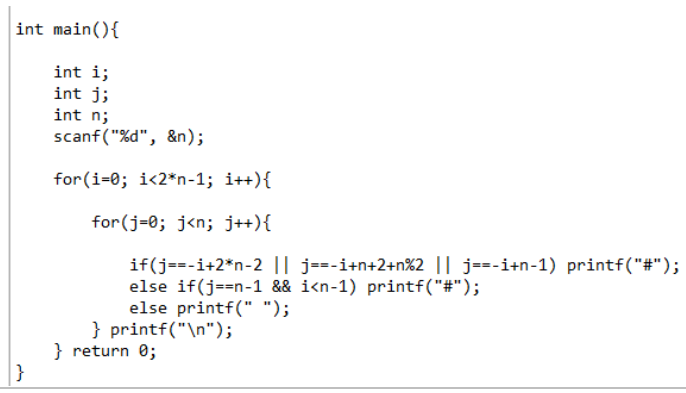
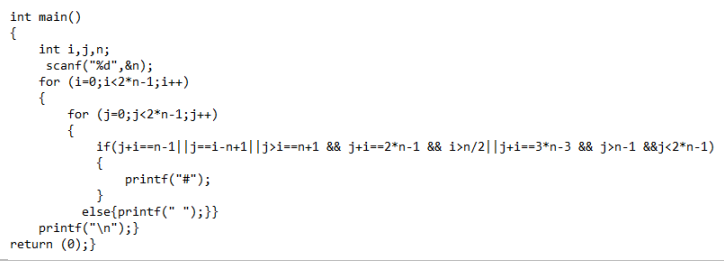
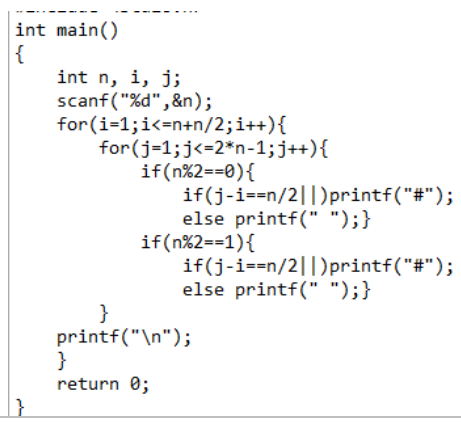
9	 #include <stdio.h> int main(){ int n; int y, x; int height; int width; scanf("%d\n", &n); height=2*n-1; width=n; for (y=1; y<height; y++){ for (x=1; x<width; x++){ if (y==(3*n)/2-x && n%2==0) printf("#"); else if (y==((3*n)/2)+1-x && n%2==1) printf("#"); else if (y==2*n-x) printf("#"); else if (x==n && y<=n) printf("#"); } printf("\n"); } }	- sintaktička greška - if grananje – definicija - v.n – krivi uvjet - parnost brojeva
10	 #include <stdio.h> int main(){ int n; int i,j; scanf("%d", &n); for(i=0; i<2*n; i++){ for(j=0; j<2*n; j++){ if(i>=n-1 && i<=2*n-1 && j>=0 && j<=n-1 && i-j==n-2) //1. crta printf("#"); else if(i>=0 && i<=n && j>=n && j<=2*n && j-i==n) //2. crta printf("#"); else if(i>=n-1 && i<=2*n && j>=n && j<=2*n && i+j==3*n-2) // 3. crta printf("#"); else if(i>=n/2 && i<=n+n/2 && j>=n/2 && j<=(n+n/2)-1 && i==j) //4. crta printf("#"); else printf(" "); } printf("\n"); } return 0; }	- v.n. – krivi uvjet - parnost brojeva
11	 #include <stdio.h> int main() { int i, j, n; scanf ("%d", &n); for (i==0; i<=(n-1)*2; i++) { for (j=0; j<=n+((n+2)/2); j++) { if (n%2==0) { if (i+j=n-1 && i<=n-1 || i-j==n-1 && i>n-1 || i+j==n*2-1 && i<=n+(n/2)-2) { printf("#") } else if printf (" ") } } } } return 0 }	- pogrešni operator (pridruživanja umjesto relacijskog) - if grananje - parnost brojeva - sintaktička greška – zagrade, točka-zarez
12	 #include <stdio.h> int main() { int n,i,j; scanf("%d%",&n); for(i=1; i<=n+n/2;i++) { for(j=1;j<=2*n-1;j++) { if(j==i-n/2 || j==-i+2*n-(n+1)/2 || j==-i+2*n-(n+1)/2) printf("#"); else printf(" "); } printf("\n"); } return 0; }	- v.n. – krivi uvjet - parnost brojeva

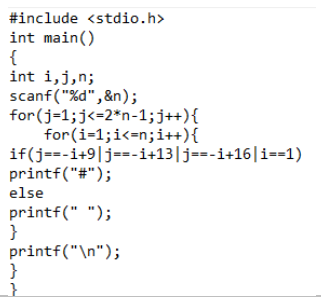
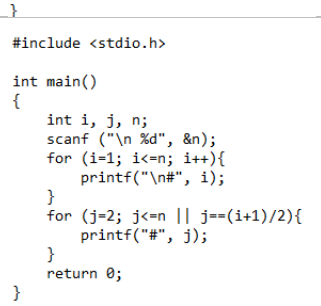
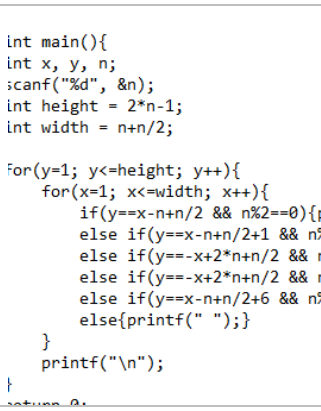
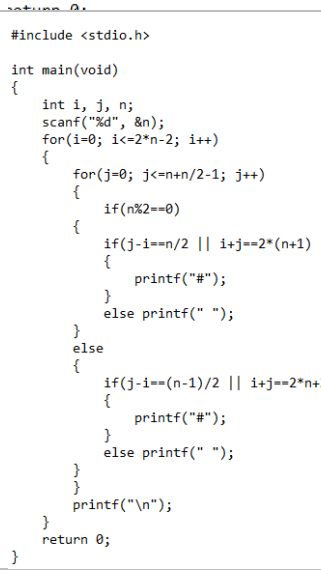
13		- neispravan znak uzorka - v.n. – krivi uvjet - parnost brojeva
14		- for petlja – pogrešne definicija - pogrešni operator (pridruživanja umjesto relacijskog) - v.n. – krivi uvjet
15		- v.n. – krivi uvjet - for petlja – pogrešne granice
16		- v.n. – krivi uvjet - parnost brojeva
17		- v.n. – krivi uvjet - parnost brojeva

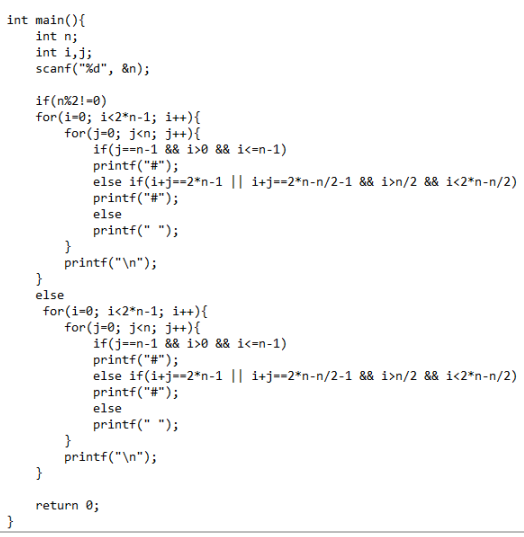
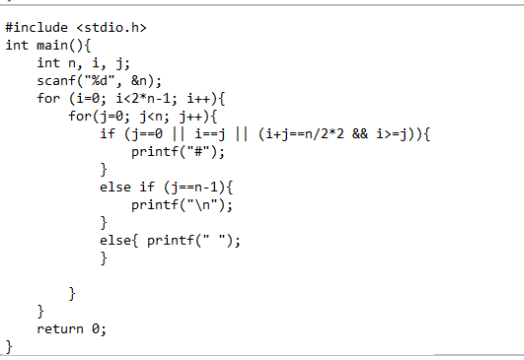
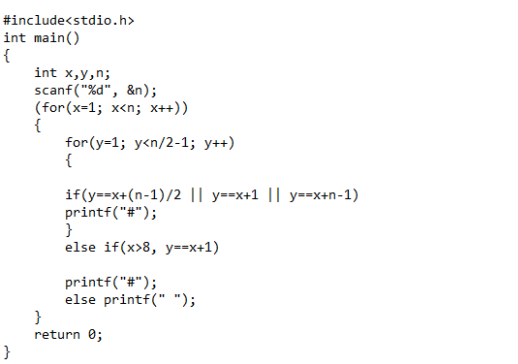
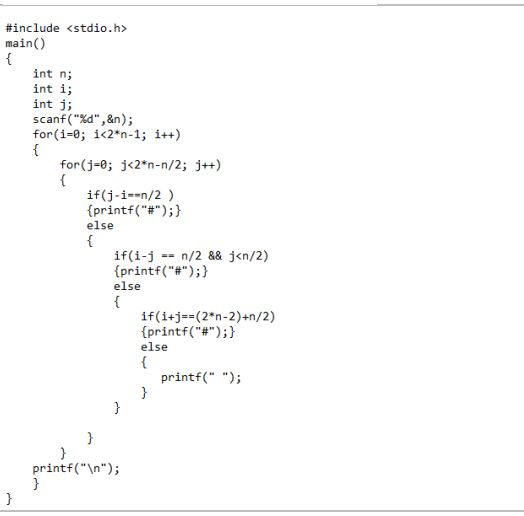
18	 #include <stdio.h> int main(){ int n, x, y; scanf("%d", &n); int height=3*n/2; int width=2*n-1; for (y=1; y<=height; y++){ for (x=1; x<=width; x++){ if ((y==-x+n+1) || (y==x-3*n/2)){ printf("#"); } else if ((y==x+1) && n%2==0 && x>=n/2){ printf("#"); } else if ((y==x+1) && n/2!=0 && (x>=n/2+1)){ printf("#"); } else printf(" "); } printf("\n"); } }	- v.n. – krivi uvjet - parnost brojeva - pogrešni operator
19	 #include<stdio.h> int main(){ int i, j, n; scanf(" %d", &n); for(i=0; i<2*n-2; i++){ for(j=0; i<2*n-5-n%2; j++){ if(j==-i+2*n-2-n%2 && i>n/2-1+n%2) printf("#"); else if(j==-i+2*n+2) printf("#"); else if(j==i-n/2) printf("#"); else printf(" "); } printf("\n"); } return 0; }	- v.n. – krivi uvjet - for petlja – pogrešna definicija - parnost
20	 #include <stdio.h> int x, y; int n; int main () { scanf("%d", &n); int visina=2*n-1; int sirina=n+(n/2); for (y=0; y< visina; y++) { for (x=0; y< sirina; x++) { if ((x+y==(2*n-1)/2)||(y-x=(2*n-1)/2||)) { printf ("#"); } else { printf (" "); } } printf ("\n"); } }	- v.n – krivi uvjet - if grananje – definicija - sintaktička greška - pogrešni operator
21	 #include<stdio.h> int main() { int i, j, n; scanf(%d, &n); for(i==0 || i =>8 || i++) #NEZNAM } return 0	- for petlja – definicija - sintaktička greška
22	 int main() { int i,j,n; scanf("%d",&n); for(i=0;i<=2*n-2;i++) { for(j=0;j<=n-1;j++){ if(n%2==0){ if((j==n-1 && i<=n-1)|| i+j==2*n-2 || i+j==n+n/2-1) printf("#"); else printf(" "); } else{ if((j==n-1 && i<=n-1) || i+j==2*n-2 || i+j==n+n/2) printf("#"); else printf(" ");}	- v.n – krivi uvjet - parnost brojeva

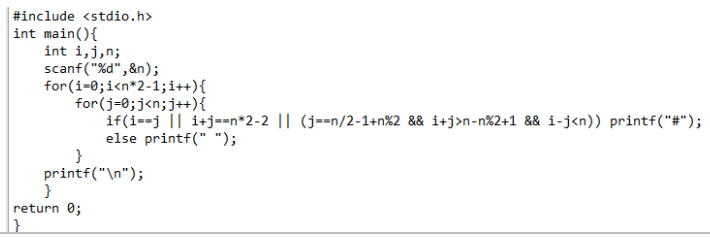
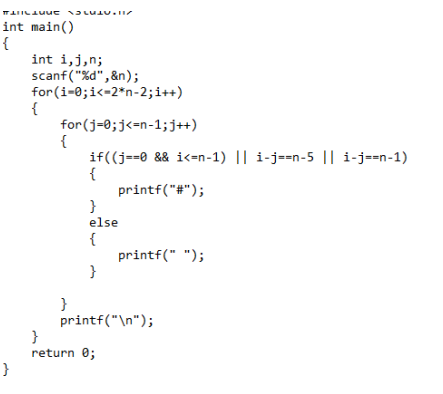
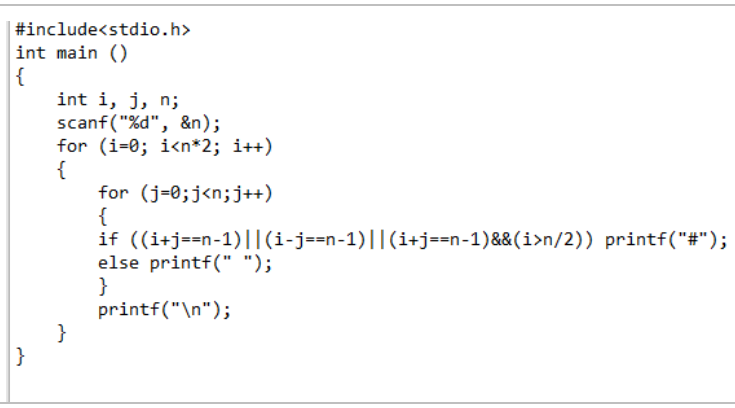
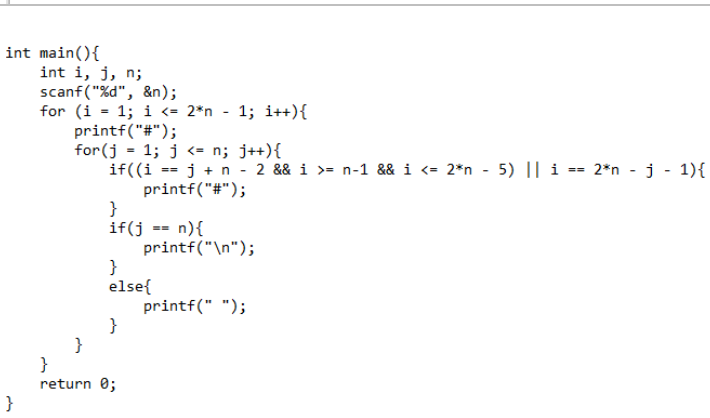
23		- sintaktička greška - for petlja – kriva definicija - v.n – krivi uvjet - parnost brojeva
24		- v.n – pogrešno postavljeno za novi red, krivi uvjet - for petlja – loša definicija, krivi operator - pogrešni operator (pridruživanja umjesto relacijskog)
25		- v.n – krivi uvjet - pogrešni operator (& umjesto &&) - for petlja - krive granice
26		- v.n – krivi uvjet

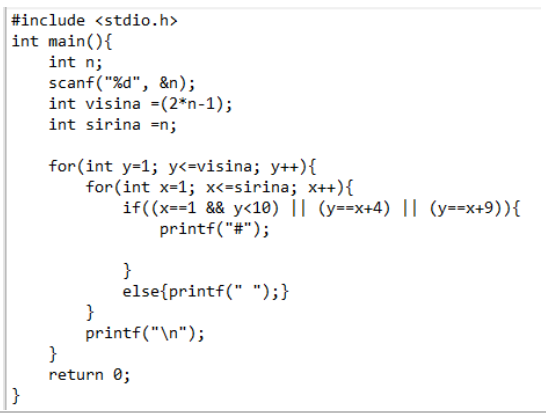
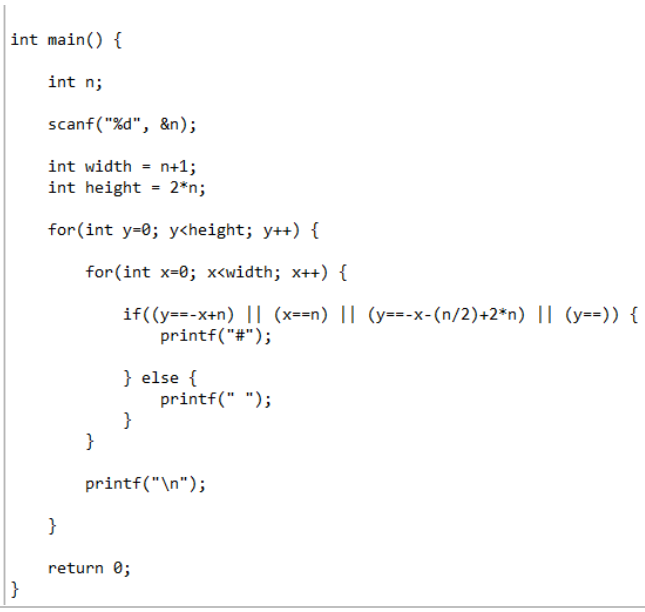
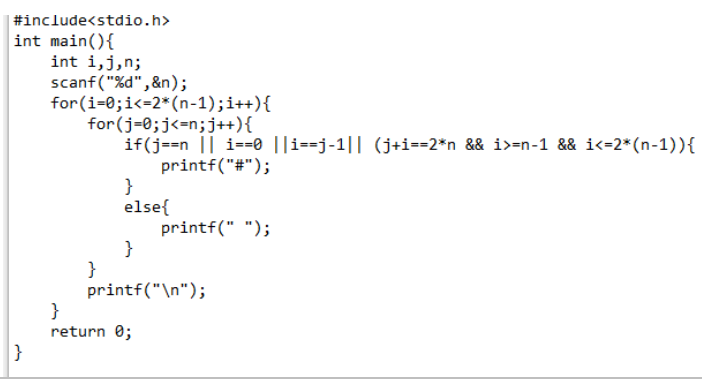
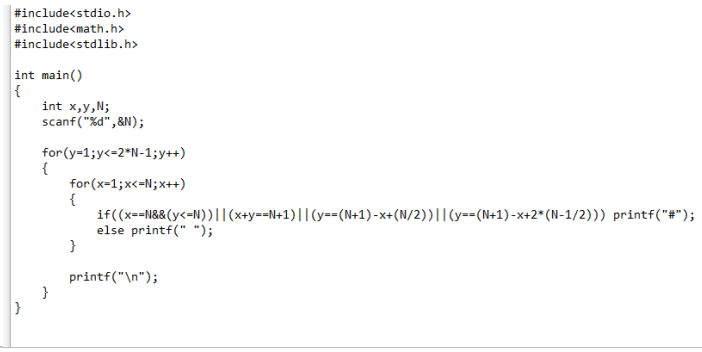
27		- v.n – krivi uvjet - parnost brojeva – krivi
28		- v.n – krivi uvjet - parnost brojeva – krivi
29		- v.n – krivi uvjet - parnost brojeva –loš za neparne
30		- v.n – krivi uvjet - parnost brojeva –loš za parne

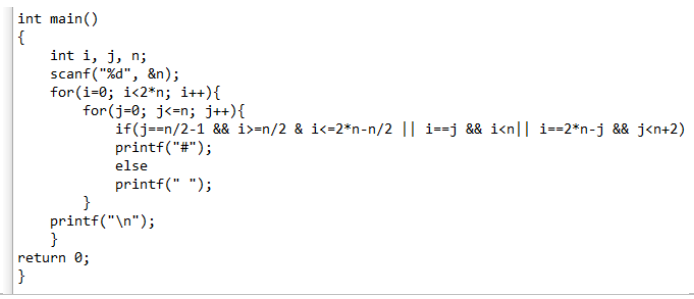
31		- v.n – krivi uvjet - parnost brojeva - nema
32		- v.n – krivi uvjet - parnost brojeva nepotreban
33		- v.n – krivi uvjet
34		- v.n – krivi uvjet
35		- sintaktička greška – logički operatori vise - v.n. – krivi uvjet - if grananje - definicija

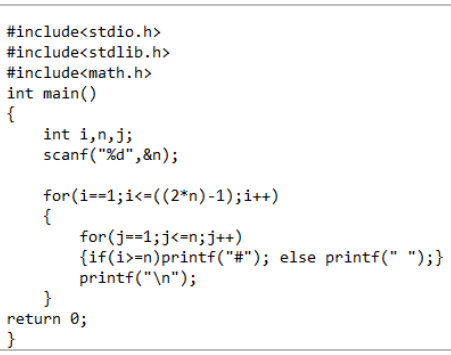
36		- v.n – krivi uvjeti - if grananje - uvjet nije definira u ovisnosti o n
37		- sintaktička pogreška (printf) - v.n – krivi uvjet - for petlja – kriva definicija - if grananje – nije definirano
38		- v.n – krivi uvjet - parnost brojeva nepotreban
39		- v.n – krivi uvjeti - parnost brojeva nepotreban

40	 `int main(){` `  int n;` `  int i,j;` `  scanf("%d", &n);` `  if(n%2!=0)` `  for(i=0; i<2*n-1; i++){` `  for(j=0; j<n; j++){` `  if(j==n-1 && i>0 && i<=n-1)` `  printf("#");` `  else if(i+j==2*n-1 || i+j==2*n-n/2-1 && i>n/2 && i<2*n-n/2)` `  printf("#");` `  else` `  printf(" ");` `  }` `  printf("\n");` `  }` `  else` `  for(i=0; i<2*n-1; i++){` `  for(j=0; j<n; j++){` `  if(j==n-1 && i>0 && i<=n-1)` `  printf("#");` `  else if(i+j==2*n-1 || i+j==2*n-n/2-1 && i>n/2 && i<2*n-n/2)` `  printf("#");` `  else` `  printf(" ");` `  }` `  printf("\n");` `  }` `  return 0;` `}`	- m.n – krivi uvjet (uzorak pomaknut za jedan dolje)
41	 `#include <stdio.h>` `int main(){` `  int n, i, j;` `  scanf("%d", &n);` `  for (i=0; i<2*n-1; i++){` `  for(j=0; j<n; j++){` `  if (j==0 || i==j || (i+j==n/2*2 && i>=j)){` `  printf("#");` `  }` `  else if (j==n-1){` `  printf("\n");` `  }` `  else{ printf(" ");` `  }` `  }` `  }` `  return 0;` `}`	- v.n – krivi uvjet
42	 `#include<stdio.h>` `int main()` `{` `  int x,y,n;` `  scanf("%d", &n);` `  (for(x=1; x<n; x++))` `  {` `  for(y=1; y<n/2-1; y++)` `  {` `  if(y==x+(n-1)/2 || y==x+1 || y==x+n-1)` `  printf("#");` `  }` `  else if(x>8, y==x+1)` `  printf("#");` `  else printf(" ");` `  }` `  return 0;` `}`	- sintaktička greška - for petlja – definicija - if grananje – struktura je pogrešna - v.n – krivi uvjet
43	 `#include <stdio.h>` `main()` `{` `  int n;` `  int i;` `  int j;` `  scanf("%d",&n);` `  for(i=0; i<2*n-1; i++)` `  {` `  for(j=0; j<2*n-n/2; j++)` `  {` `  if(j-i==n/2 )` `  {printf("#");}` `  else` `  {` `  if(i-j == n/2 && j<n/2)` `  {printf("#");}` `  else` `  {` `  if(i+j==(2*n-2)+n/2)` `  {printf("#");}` `  else` `  {` `  printf(" ");` `  }` `  }` `  }` `  }` `  printf("\n");` `  }` `}`	- sintaktička greška - if grananje - v.n – krivi uvjet

44		- v.n – krivi uvjet
45		- v.n – krivi uvjet
46		- v.n – krivi uvjet
47		- v.n – krivi uvjet

48	 #include <stdio.h> int main(){ int n; scanf("%d", &n); int visina =(2*n-1); int sirina =n; for(int y=1; y<=visina; y++){ for(int x=1; x<=sirina; x++){ if((x==1 && y<10) || (y==x+4) || (y==x+9)){ printf("#"); } else{printf(" ");} } printf("\n"); } return 0; }	- v.n – krivi uvjeti - parnost brojeva
49	 int main() { int n; scanf("%d", &n); int width = n+1; int height = 2*n; for(int y=0; y<height; y++) { for(int x=0; x<width; x++) { if((y==-x+n) || (x==n) || (y==-x-(n/2)+2*n) || (y==)) { printf("#"); } else { printf(" "); } } printf("\n"); } return 0; }	- sintaktička greška (nedovršen uvjet) - v.n – krivi uvjet
50	 #include<stdio.h> int main(){ int i,j,n; scanf("%d",&n); for(i=0;i<=2*(n-1);i++){ for(j=0;j<=n;j++){ if(j==n || i==0 ||i==j-1|| (j+i==2*n && i>=n-1 && i<=2*(n-1)){ printf("#"); } else{ printf(" "); } } printf("\n"); } return 0; }	- sintaktička greška (nedostaje zatvorena zagrada u uvjetu if) - v.n – krivi uvjet
51	 #include<stdio.h> #include<math.h> #include<stdlib.h> int main() { int x,y,N; scanf("%d",&N); for(y=1;y<=2*N-1;y++) { for(x=1;x<=N;x++) { if((x==N&&(y<=N))||(x+y==N+1)||(y==(N+1)-x+(N/2))||(y==(N+1)-x+2*(N-1/2))) printf("#"); else printf(" "); } printf("\n"); } }	- v.n – krivi uvjet

52	 ``` int main() { int i, j, n; scanf("%d", &n); for(i=0; i<2*n; i++){ for(j=0; j<=n; j++){ if(j==n/2-1 && i>=n/2 & i<=2*n-n/2 || i==j && i<n|| i==2*n-j && j<n+2) printf("#"); else printf(" "); } printf("\n"); } return 0; } ```	- for petlja – pogrešno određena granica - pogrešni operator (& umjesto &&) - v.n – krivi uvjeti
53	``` #include <stdio.h> main() { int i,j,n; scanf("%d",&n); for(i=0;i<n+(n/2);i++) { for(j=0;j<2*n-1;j++) { if(i-j==n/2) {printf("#");} else{ if(n<10 && j+i==2*n+2) {printf("#");} else{ if(n>=10 && j+i==2*n+4) {printf("#");} else{ if(n%2==0 && j>=n/2 && i+j==n+(n/3)) {printf("#");} else{ if(n%2!=0 && j>n/2 && i+j==n+(n/3)) {printf("#");} else{printf(" ");} }}}} }printf("\n"); }return 0; } ```	- sintaktička greška (nedefiniran povratni tip funkcije main()) - vn – pogrešni uvjeti za n>9
54	``` #include<stdio.h> int main() { int i,j,n; scanf(" %d", &n); for (i==0; i<n; i++); { for (j==0; j<=2*n-1; j++); { if ((i==0 && j<=2*n-1) || (j==2*n-2 && i<=n-1) || j==-i+n-1 || j==i+n-1) printf("#"); else printf(" "); } printf("\n"); } return 0; } ```	- pogrešni operator (usporedba umjesto pridruživanja u for petlji) - for petlja (definicija – operator i ';') - v.n – pogrešni uvjeti
55	``` #include <stdio.h> int main() { int i,j,n; scanf("%d", &n); for(i=0; i<=2*n-2; i++){ for(j=0; j=2*n-2; j++){ if(i+j==n-1||j-i==n-1||i+j==3*n-3||i+j==2*n-1-n%2 &&i>(n-1)/2-n%2){ printf("#"); } else{ printf(" "); }} printf("\n"); } return 0; } ```	- pogrešni operator (usporedba umjesto pridruživanja u for petlji) - for petlja (definicija) - v.n – pogrešan uvjet

56		- pogrešni operator (usporedba umjesto pridruživanja)

Literatura

[1] Griffiths, D.; Griffiths, D. Head First C. 1st ed. Sebastopol, CA: O'Reilly Media, 2012.

[2] King, K. N. C Programming: A Modern Approach. 2nd ed. New York: W. W. Norton & Company, 2008.

[3] Prata, S. C Primer Plus. 5th ed. Upper Saddle River, NJ: Sams Publishing, 2004.

[4] Jurak, M. Programski jezik C: Predavanja. Zagreb: Prirodoslovno-matematički fakultet, 2003/04. (Dostupno na: https://web.math.pmf.unizg.hr/~jurak/C.pdf, pristup 13.4.2024.)

[5] Kernighan, B. W.; Ritchie, D. M. The C Programming Language. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1988.

[6] For Loops in C. (Dostupno na: https://www.w3schools.com/c/c_for_loop.php, pristup 20.4.2024.)

[7] Nested Loops in C. (Dostupno na: https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/nested-loops-in-c/, pristup 20.4.2024.)

[8] Plauger, P. J. The C Standard Library. Upper Saddle River, NJ: Prentice Hall, 1991.

[9] How to Run C Program in Command Prompt. (Dostupno na: https://www.wikihow.com/Run-C-Program-in-Command-Prompt, pristup 6.9.2024.)

[10] Compiling C Programs (Cassio Kuhlmann). (Dostupno na: https://cass-kul.github.io/tutorials/compiling-c/, pristup 7.9.2024.)

[11] Synthesis of Nested Loop Exercises for Practice in Introductory Programming. // Egyptian Informatics Journal, Vol. 24, Issue 2 (2023), str. 113-120. (Dostupno na: https://www.sciencedirect.com/science/article/pii/S1110866523000166, pristup 20.7.2024.)

[12] Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. (Dostupno na: https://www.researchgate.net/publication/320679287_Students%27_Misconceptions_and_Other_Difficulties_in_Introductory_Programming_A_Literature_Review, pristup 1.8.2024.)

[13] Ten Simple Rules for Writing a Computational Paper. // PLOS Computational Biology, 2018. (Dostupno na: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006023, pristup 6.9.2024.)

[14] Winslow, L. E. Programming Pedagogy: A Psychological Overview. // Proceedings of the 24th SIGCSE Technical Symposium on Computer Science Education, ACM, 1993, str. 19-23. (Dostupno na: https://dl.acm.org/doi/10.1145/234867.234872, pristup 11.8.2024.)

[15] Live-coding vs Static Code Examples: Which is Better with Respect to Student Learning and Cognitive Outcomes? // Proceedings of the ACM Conference on Learning at Scale, 2020. (Dostupno na: https://dl.acm.org/doi/abs/10.1145/3373165.3373182, pristup 11.8.2024.)

[16] Learning Styles Impact Students' Perceptions on Active Learning Methodologies: A Case Study on the Use of Live Coding and Short Programming Exercises. // Education Sciences, Vol. 14, No. 3 (2023), str. 250-260. (Dostupno na: https://www.mdpi.com/2227-7102/14/3/250, pristup 1.8.2024.)

[17] The Effectiveness of Live-Coding to Teach Introductory Programming. // Proceedings of the ACM Technical Symposium on Computer Science Education, 2013. (Dostupno na: https://dl.acm.org/doi/10.1145/2445196.2445388, pristup 1.8.2024.)