1. Naštejte osnovne podatkovne tipe in opišite pravila, ki veljajo za deklaracijo spremenljivk.

Podatkovni tip	Število zlogov	Območje oz. obseg
short	2 zloga (16 bitov)	-32768 .. 32767 (predznačena) 0 .. 65535 (nepredznačena)
int	2 zloga (16 bitov) ali 4 zlogi (32 bitov) *	
long	4 zlogi (32 bitov)	2147483648 .. 2147483647 (predznačena) 0 .. 4294967295 (nepredznačena)
char	1 zlog	-128 .. +127 (predznačena) 0 .. 255 (nepredznačena)
float	4 zlogi	3.4e + / - 38 (7 mest)
double	8 zlogov	1.7e + / - 308 (15 mest)
long double	10 zlogov	1.2e + / - 4932 (19 mest)
bool	1	true ali false

2. Predstavite sintakso if stavka in zapišite primer uporabe gnezdenega if stavka.

```
if (pogoj)
stavek1;
else
stavek2;
```

Gnezden:

```
if (pogoj)
stavek1;
else
if (pogoj2)
stavek2;
```

3. Predstavite sintakso switch stavka in zapišite primer uporabe switch stavka.

```
switch (izraz)
{
case konstanta_1:
zaporedje_stavkov_1;
break;
case konstanta_2:
zaporedje_stavkov_2;
break;
.
.
.
default:
privzeto_zaporedje_stavkov;
}
```

4. Opišite razliko med while in do while stavkom in zapišite primer izpisa števil od 100 do 200 z obema stavkoma.

Do while stavek se razlikuje od **while** stavka po tem, da se ne glede na pogoj izvede vsaj enkrat, kar je posledica tega, da se
pogoj testira po izvr.itvi stavka.

Primer z DO WHILE stavkom:

int i=100;

```
do
{
cout<<i;
i++;

}
while (i<=200);
```

Primer z WHILE stavkom:

int i=100;

```
while (i<=200)
{
cout<<i;
i++;
}
```

5. Predstavite in razložite sintakso for stavka.

```
for (inicializacija; pogoj; sprememba_stevca)
{
stavek1;
stavek2;
...
}
```

Stavek ali blok stavkov v **for** stavku se izvr.uje dokler je izpolnjen pogoj. V inicializacijskem delu deklariramo
in določimo začetno vrednost spremenljivke, ki jo uporabimo za .tevec v zanki. V delu sprememba .tevca pa
določimo spremembo .tevca (povečujemo oz. zmanj.ujemo spremenljivko .tevca - korak).

6. Predstavite sintakso do while stavka in zapišite primer uporabe.

```
do
{
stavek1;
stavek2;
...
}
while (pogoj);
```

```
// for zanka z zmanj.evanjem .tevca
#include <iostream.h>
int main ()
{
      for (int n=10; n>0; n--) //ponavljanje dokler je n > 0
      {
            cout << n << ", "; //izpis vrednosti .tevca
      }
```

```
	cout << "Konec!";
	return 0;
}
//izpiše
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, Konec!
```

7. Predstavite sintakso while stavka in zapišite primer uporabe.

```
while (pogoj)
{
stavek1;
stavek2;
...
}
```

```
// ponavljanje z zmanj.evanjem .tevca
#include <iostream.h>
int main ()
{
	int n;
	cout << "Vpiši celo število > ";
	cin >> n; //preberemo .tevilo
	// ponavljamo dokler je .tevilo večje od 0
	while (n>0)
		{
			cout << n << ", "; // izpi.emo .tevilo
			--n; // za ena zmanj.amo n
		}
	cout << "Konec!";
	return 0;
}

//izpi.e
Vpiši celo število > 5
5, 4, 3, 2, 1, Konec!
```

8 Predstavite sintakso za definicijo funkcije.

Sintaksa:
podatkovni_tip ime_funkcije (argument1, argument2, ...)
stavek

Podatkovni_tip je podatkovni tip podatka, ki ga vrne funkcija (podatkovni tip rezultata).
Ime funkcije je smiselno ime, ki ga uporabimo pri klicu funkcije.
Argumenti nam omogočajo prenose podatkov v in iz funkcije. Vsak argument ima
ime argumenta (identifikator) npr. int stevilo_kock. Argumente ločimo med seboj
Stavek predstavlja telo funkcije in je lahko enostaven stavek ali blok stavkov.
Vrednost funkcije določimo s stavkom **return**, ki ima sintakso: **return** izraz.

9. Predstavite pravila in značilnosti pri prenosu parametrov funkcije (po vrednosti in po referenci).

Do sedaj smo prenašali parametre v funkcije po vrednosti (by value). Ob klicu funkcije smo prenesli vrednosti
parametrov v argumente funkcije.
Zelo pogosto pa moramo prenesti več vrednosti iz funkcije, ki jo kličemo. Vemo že, da preko vrednosti funkcije
lahko prenesemo le eno vrednost. V takšnih primerih uporabimo prenos parametrov po referenci (by reference).

```
// prenos parametrov po referenci
#include <iostream.h>
void dvakrat_povecaj (int & a, int & b, int & c)
{
a*=2;  // novi vrednosti a-ja priredimo staro vrednost a-ja pomno.eno z 2
b*=2; c*=2;
}
int main ()
{
int x=1, y=3, z=7;
//klic funkcije s prenosom po referenci
dvakrat_povecaj (x, y, z);
// izpis vrednosti spremenljivk
cout << "x=" << x << ", y=" << y << ", z=" << z;
return 0;
}
// izpiše
x=2, y=6, z=14
```

10. Kaj je prototip funkcije in kdaj ga uporabimo.

Do sedaj smo deklarirali funkcije pred glavno funkcijo main. Če zapi.emo deklaracijo funkcije za funkcijo main, potem bi nam prevajalnik javil napako. Pisanju funkcij pred glavno funkcijo se izognemo z uporabo prototipov funkcij. Prototip funkcije je deklaracija glave funkcije, ki jo nato v celoti zapi.emo v nadaljevanju
programa.
Deklaracija prototipa je enaka deklaraciji glave funkcije.

Sintaksa:
podatkovni_tip ime_funkcije (argument1, argument2, ...);

Prototip funkcije ne vsebuje telesa funkcije in se zaključi s podpičjem.
V deklaracji argumentov je dovolj, da zapi.emo podatkovni tip argumentov. Kljub temu (v praksi) zapi.emo
tudi ime argumenta, saj je s tem program razumljivej.i.

Zgled:

```
// uporaba prototipov
#include <iostream.h>
void liho (int a);
void sodo (int a);
int main ()
{.......
```

11. Opišite namen uporabe stavkov break in continue.

Stavek *break*
Če uporabimo **break** v bloku stavkov zanke, potem se izvajanje zanke **prekine** ne glede na to, da pogoj .e ni
izpolnjen. Uporabimo ga lahko za izhod iz neskončne zanke (pogoj je vedno izpolnjen).

Stavek continue
Continue povzroči, da program preskoči preostale stavke v zanki v trenutnem ponavljanju zanke in nadaljuje
izvajanje, kot da bi zaključil vse stavke v bloku stavkov zanke (izvede se naslednja ponovitev zanke).
Uporabimo ga lahko samo v zankah.

12. Opišite pravila, ki veljajo za doseg spremenljivk.

Lokalne spremenljivke so deklarirane znotraj funkcije (v kateremkoli bloku). Dostopne so le znotraj funkcije
oz. bloka v katerem so deklarirane. Obstajajo le v času izvajanja funkcije oz. bloka. Lokalna spremenljivka ima
doseg in .ivljensko dobo od mesta definicije do konca bloka, v katerem je definirana.
Globalne spremenljivke so deklarirane izven funkcij. Običajno jih deklariramo na začetku programa, preden
definiramo prvo funkcijo. Spremenljivka, deklarirana na ta način, je dostopna znotraj vseh funkcij programa in
njeno vrednost lahko spremenimo kjerkoli v programu. Zato njihova uporaba ni priporočljiva, saj program postaja nepregleden in se zelo poveča mo.nost različnih napak, ki jih te.ko odkrijemo. Globalne spremenljivke
obstajajo ves čas dokler se program izvaja. Če na začetku deklaracije uporabimo extern pomeni, da lahko to
globalno spremenljivko uporabljamo na nivoju celotnega programa.

13. Kakšne posebnosti veljajo za nize znakov v c++ in katero funkcijo uporabimo za primerjavo dveh nizov.

Nizi znakov so eden izmed najbolj pogosto uporabljenih sestavljenih tipov. Zapisujemo jih lahko kot polje znakov ali kot kazalec na znake.

Primer zapisa s poljem:

```
char niz[] = "test"
```

Ekvivalenten zapis je:

```
niz[0] = 't'; niz[1] = 'e'; niz[2] = 's'; niz[3] = 't'; niz[4] = '\0';
```

ali:

```
char niz[] = {'t', 'e', 's', 't', '\0'}
```

uporaba pri vhodnem toku:

```
char niz1[20];
cin >> niz1;
```

in izhodnem toku:

```
cout << niz1;
```

Primer izpisa niza znakov s presledki med znaki:

```
for (int i=0; niz[i]!='\0'; i++)

cout << niz[i] << ' ';
```

Podobno lahko pri definiciji namesto polja znakov:

```
char niz[] = "Lep dan"; uporabimo char * niz = "Lep dan"
```

Obe definiciji rezervirata toliko znakov, kot je .tevilo znakov niza in en dodaten ničelni znak. Razlika med definicijama je, da prvi predstavlja identifikator niz naslov prvega elementa polja, v drugi pa je identifikator niz
kazalec, ki vsebuje naslov začetka rezerviranega polja znakov.

int strcmp (const char *s1, const char *s2) . primerja niza med seboj;

14. Predstavite enodinemzionalne tabele (deklaracija, opis pomnilniške slike za tabelarično spremenljivko, dostop do elementov…).

Sintaksa deklaracije:
***podatkovni_tip ime* [*st_elementov*];**

Dostop do elementov tabele
Do vrednosti posameznega elementa tabele dostopamo:
`ime_tabelaricne_spremenljivke[indeks]`

15. Kakšne značilnosti veljajo med tabelarično spremenljivko in kazalci.

Koncept tabel je zelo povezan s konceptom kazalcev. Tabelarična spremenljivka ima vrednost naslova prvega
elementa v tabeli. Kazalec pa ima naslov prvega elementa na katerega ka.e. Npr. za deklaracijo

```
int stevila [10];
int * p;
```

je veljaven naslednji stavek:

```
p = stevila;
```

16. Predstavite kazalčno spremenljivko (kaj je kazalec, deklaracija kazalca, primer uporabe).

17. Katere algoritme za urejanje podatkov smo spoznali. Predstavite enega izmed njih.

Urejanje po metodi izbora najmanjšega elementa:

poi.čemo najmanj.e .tevilo v neurejeni tabeli,

postavimo ga na prvo mesto neurejenega dela tabele, hkrati pa .tevilo, ki je bilo do tedaj na prvem mestu neurejenega dela tabele postavimo na mesto, na katerem smo na.li najmanj.e .tevilo v neurejeni tabeli,

neurejeni del tabele se s tem zmanj.a za eno .tevilo,

to ponavljamo dokler v neurejenem delu tabele ne ostane samo eno .tevilo, ki je hkrati največje v urejeni tabeli.

Urejanje po metodi mehurčkov (Bubble sort)

Urejanje z vstavljanjem

Navadno vstavljanje

Binarno vstavljanje

18. Kakšna je osnovna ideja urejanja podatkov po metodi izbora najmanjšega elementa. Zapišite tudi primer urejanja za 6 celih števil.

Urejanje po metodi izbora najmanjšega elementa:

poi.čemo najmanj.e .tevilo v neurejeni tabeli,

postavimo ga na prvo mesto neurejenega dela tabele, hkrati pa .tevilo, ki je bilo do tedaj na prvem mestu neurejenega dela tabele postavimo na mesto, na katerem smo na.li najmanj.e .tevilo v neurejeni tabeli,

neurejeni del tabele se s tem zmanj.a za eno .tevilo,

to ponavljamo dokler v neurejenem delu tabele ne ostane samo eno .tevilo, ki je hkrati največje v

urejeni tabeli.

Zgled:
Neurejene tabela tab ima 5 elementov:

indeks i->	0	1	2	3	4
tab[i]	23	4	12	34	5

Začetno mesto neurejene tabele ima indeks 0. Poi.čemo najmanj.i element od mesta neurejene tabele naprej.
Ugotovimo, da je to .tevilo 4, ki je na indeksu 1. Zamenjamo .tevili na indeksih 0 in 1.

indeks i->	0	1	2	3	4
tab[i]	4	23	12	34	5

Začetno mesto neurejene tabele ima sedaj indeks 1. Poi.čemo najmanj.i element od mesta neurejene tabele naprej. To je .tevilo 5, ki je na indeksu 4 v tabeli. Zamenjamo .tevili na začetnem mestu neurejene tabele in
mestu najmanj.ega v neurejeni tabeli (indeksa 1 in 4).

indeks i->	0	1	2	3	4
tab[i]	4	5	12	34	23

Začetno mesto neurejene tabele ima indeks 2. Poi.čemo najmanj.i element od mesta neurejene tabele naprej. To
je .tevilo 12, ki je na indeksu 2 v tabeli. Zamenjava ni potrebna, saj je najmanj.e .tevilo v neurejeni tabeli .e na
prvem mestu neurejene tabele.

indeks i->	0	1	2	3	4
tab[i]	4	5	12	34	23

Začetno mesto neurejene tabele ima indeks 3. Poi.čemo najmanj.i element od mesta neurejene tabele naprej. To
je .tevilo 23, ki je na indeksu 4 v tabeli. Zamenjamo .tevili na začetnem mestu neurejene tabele in mestu najmanj.ega v neurejeni tabeli (indeksa 3 in 4).

indeks i->	0	1	2	3	4
tab[i]	4	5	12	23	34

V neurejenem delu tabele je ostalo samo eno .tevilo in s tem je tabela urejena po velikosti.

19. Kakšna je osnovna ideja urejanja podatkov po metodi mehurčkov. Zapišite tudi primer urejanja za 6 celih števil.

Urejanje po metodi mehurčkov (Bubble sort)

začnemo na začetku tabele, primerjamo i-ti in i+1-vi element tabele; če nista v predpisanem zaporedju, ju zamenjamo,

v enem prehodu skozi tabelo pomaknemo največje .tevilo na konec tabele, manj.a .tevila pa pomikamo proti začetku tabele,

zgornja koraka ponavljamo dokler ni tabela urejena.

Zgled:

indeks i-> 0 1 2 3 4
tab[i] 23 4 12 34 5

Primerjamo prvi in drugi element tabele in ju zamenjamo.

indeks i-> 0 1 2 3 4
tab[i] 4 23 12 34 5

Primerjamo drugi in tretji element tabele in ju zamenjamo.

indeks i-> 0 1 2 3 4
tab[i] 4 12 23 34 5

Primerjamo tretji in četrti element tabele; zamenjava ni potrebna.

indeks i-> 0 1 2 3 4
tab[i] 4 12 23 34 5

Primerjamo predzadnji in zadnji element tabele in ju zamenjamo.

indeks i-> 0 1 2 3 4
tab[i] 4 12 23 5 34

Končali smo prvi prehod, največje .tevilo 34 je na zadnjem mestu v tabeli. Na enak način nadaljujemo .e s preostalimi prehodi in opravimo naslednje zamenjave.

indeks i-> 0 1 2 3 4
tab[i] 4 12 5 23 34

indeks i-> 0 1 2 3 4
tab[i] 4 5 12 23 34

indeks i-> 0 1 2 3 4
tab[i] 4 5 12 23 34

20. Kakšna je osnovna ideja binarnega iskanja. Zapišite tudi primer iskanja za 8 celih števil.

Binarno iskanje

Problem iskanja določenega elementa v urejenem zaporedju elementov.
Pogoj za binarno iskanje je, da so elementi tabele urejeni po velikosti. Urejeno tabelo razdelimo enakomerno na
dve polovici. Iskani podatek je lahko v prvi ali v drugi polovici. Ustrezno polovico tabele izberemo s pomočjo
primerjave iskanega podatka s sredinskim elementom tabele. Tako izločimo polovico tabele. Iskanje nadaljujemo na isti način v izbrani polovici. V dveh korakih izločimo tri četrtine podatkov. Postopek ponavljamo dokler ne najdemo .tevila, ki ga iščemo ali dokler ne ugotovimo, da iskanega števila ni v tabeli.

Zgled: Ugotovimo če je število 23 element tabele?

indeks i-> 0 1 2 3 4
tab[i] 4 5 12 23 34
Spodaj Zgoraj

Določimo sredinski element:

Sred = (Spodaj + Zgoraj) / 2 = (0 + 4) / 2 = 2

Ker je iskano število večje od tab[2], postavimo spodnjo mejo na Sred + 1 = 3 in nadaljujemo iskanje v zgornji
polovici tabele.

indeks i-> 0 1 2 3 4
tab[i] 4 5 12 23 34
Spodaj Zgoraj

Sred = (Spodaj + Zgoraj) / 2 = (3+4) / 2 = 3

Po dveh primerjavah ugotovimo, da je število 23 element tabele.

Če bi iskali število 22, bi pri.li do naslednje situacije:
Iskano število 22 je manjše od sredinskega .tevila 23 in zgornjo mejo postavimo na sredinsko .tevilo.
Veljalo bi: Spodaj = Zgoraj = 3.

Torej pri neuspešnem iskanju velja Spodaj = Zgoraj = Sred in lahko pogoj Spodaj < Zgoraj uporabimo za ponavljanje iskanja.

21 Predstavite operatorja new in delete. Zapišite tudi primer uporabe.

Operatorja new in new[]
Operator new uporabimo za dodeljevanje pomnilni.kega prostora. Za njim zapi.emo podatkovni tip in kot opcijo lahko navedemo tudi .tevilo elementov v oglatih oklepajih.
Vrne nam kazalec na začetek novo dodeljenega pomnilni.kega prostora.
Sintaksa je:
kazalec = new podatkovni_tip // kazalec na en element izbranega podatkovnega tipa

Operator delete
Ko ne potrebujemo več dinamičnega pomnilnika, ga sprostimo z uporabo operatorja delete.
Sintaksa:
delete kazalec;
ali
delete [] kazalec;
Delete kazalec uporabimo za sprostitev enega elementa. Drugo obliko pa za spro.čanje pomnilnika za več elementov. Po izvr.itvi spro.čanja ima kazalec vrednost NULL.

```
#include <iostream.h>
#include <stdlib.h>


int main ()
{
char input [100];
int i,n;
long * l, total = 0;
```

```
cout << "Koliko stevil zeliš vnesti? ";
cin.getline (input,100); i=atoi (input);
l= new long[i];
if (l == NULL) exit (1);
for (n=0; n<i; n++)
{
cout << "Vnesi stevilo: ";
cin.getline (input,100);
l[n]=atol (input);
}
cout << "Vnesel si: ";
for (n=0; n<i; n++)
cout << l[n] << ", ";
delete [] l;
return 0;
}
```

22 Predstavite sintakso za strukture in zapišite preprost primer uporabe.

Sintaksa:
struct naziv_strukture
{
podatkovni_tip element1**;**
podatkovni_tip element2**;**
podatkovni_tip element3**;**
...
} naziv_objekta**;**

Definicija strukture se prične z rezerviranko struct, ki ji sledi seznam deklaracij njenih komponent (elementov
strukture) med zavitima oklepajema. Struktura ima svoje ime (naziv_strukture), ki ga imenujemo tudi oznaka
strukture (structure tag). Za zavitim oklepajem, ki zaključi seznam komponent, lahko sledi seznam spremenljivk
(naziv_objekta), ki so podatkovnega tipa te strukture. Do posamezne komponente strukture dostopamo z izrazom naziv_strukture.komponenta. Strukturni operator "." naredi povezavo med imenom strukture in njeno
komponento.

23 Namen uporabe strukture in tabel.

Struktura omogoča zdru.itev podatkov različnega podatkovnega tipa v nov podatkovni tip. S strukturami zdru.imo sorodne podatke (lahko različnega podatkovnega tipa) pod enim imenom in jih nato

obravnavamo kot
celoto, ki vsebuje posamezne elemente oz. komponente. S strukturami zdru.ujemo podatke, ki tvorijo neko smiselno celoto (npr. podatke o osebi, podatke o izdelku.). Podatkom lahko dodamo .e metode, ki se izvajajo
nad temi podatki. Najprej se osredotočimo samo na zdru.evanje podatkov, ki tvorijo neko smiselno celoto.

Tabele oz. polja so zaporedja elementov oz. spremenljivk **enakega podatkovnega tipa**, ki so shranjene na sosednjih pomnilni.kih lokacijah. Do posameznega elementa tabele dostopamo tako, da za ime tabelarične spremenljivke dodamo indeks med oglatima oklepajema.
V deklaraciji spremenljivk lahko rezerviramo npr. devet spremenljivk celo.tevilčnega podatkovnega tipa npr.
int, ki bodo shranjena na sosednjih pomnilni.kih lokacijah (zvezno).