23. Maturitná Otázka

A) Logické obvody (Hardvérová časť)

Boolová Algebra

- *Zahŕňa logické funkcie ,ktoré sa využívajú v informatike a elektrotechnike.*
- *Na základe týchto funkcií sa vytvárajú logické hradlá ,z ktorých sú zostavené tranzistory, integrované obvody a procesory.*
- *Boolová algebra využíva binárnu sústavu.*
- *Premenné v boolovej algebre teda môžu nadobúdať hodnoty 0 alebo 1.*
- *Vstupné hodnoty premenných a výslednú hodnotu funkcie zapisujeme do pravdivostnej tabuľky.*
- *Vstupné hodnoty označujeme ako nezávislé premenné, výstupnú hodnotu označujeme ako závislo-premennú.*

Negácia

– Je najjednoduchšou boolovskou operáciou, pri negácií nadobúda premenná opačnú hodnotu.

– Pravdivostná tabuľka:

x	*f (x)*
0	*1*
1	*0*

– Negácia sa označuje rovnou vodorovnou čiarou nad premennou.

$$\overline{X}$$

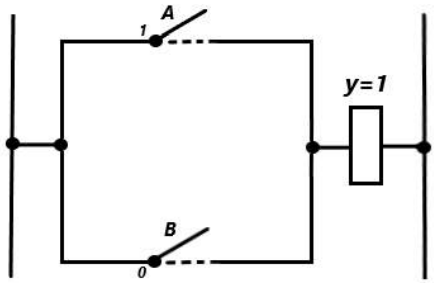
Logický súčet

– Je operácia disjunkcie (zjednotenie).

– Býva tiež označovaná $y = \overline{A + B}$

Disjunkcia *Premenné*

– Schematicky ju môžeme znázorniť takto:



A	B	y
0	0	0
1	0	1
0	1	1
1	1	1

– Pre operáciu logického súčtu je charakteristická spojka alebo.

A OR B

A + B

A alebo B

A ∨ B

Logický súčet

– Je operácia konjunkcie alebo prieniku, označuje sa aj takýmto znakom ∧.

– Pre logický súčin sú typické spojky a, aj, i.

A ∧ B

A & B

A · B

A AND B

– Výsledná hodnota funkcie je jedna len vtedy, ak všetky nezávislé premenné majú hodnotu 1.

A	B	Y
0	0	0
1	0	0
0	1	0
1	1	1

Ďalšie Boolovské funkcie

NAND (Negovaný súčin)

– Funkcia NAND má hodnotu 1, ak aspoň 1 nezávislá premenná má hodnotu 0.

A	B	Y
0	0	1
1	0	1
0	1	1
1	1	0

NOR (Negovaný súčet)

– Funkcia NOR má hodnotu 1, ak všetky vstupy majú hodnotu 0.

A	B	Y
0	0	1
1	0	0
0	1	0
1	1	0

XOR (Exkluzívny súčet)

– Funkcia XOR má hodnotu 1, ak vstupy majú rozdielne hodnoty.

A	B	Y
0	0	0
1	0	1
0	1	1
1	1	0

Karnaughové mapy

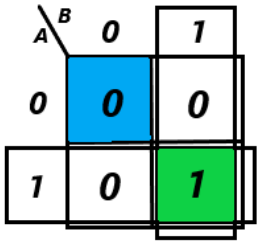
- *Slúži na zjednodušený zápis funkcie.*
- *Aby bola mapa karnaughovou mapou musí spĺňať tieto podmienky:*

1. *Karnaughová mapa má toľko políčok koľko má pravdivostná tabuľka riadkov (2^n).* *n=počet vstupov*
2. *V Karnaughovej mape musí existovať 1 bunka, v ktorej majú všetky nezávislé vstupy hodnotu 0.*
3. *V Karnaughovej mape musí existovať 1 bunka, v ktorej majú všetky nezávislé vstupy hodnotu 1.*

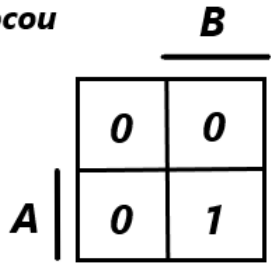
4. *Pre každý nezávislý vstup musí platiť ,že pre polovicu buniek Karnaughovej mapy má vstup hodnotu 0 a pre polovicu buniek Karnaughovej mapy má vstup hodnotu 1.*

2 spôsoby zápisu Karnaughovej mapy

Aritmetický zápis



Zápis pomocou svoriek



- *Do Karnaughovej mapy zapisujeme hodnotu funkcie, nie hodnotu vstupov.*

Preklápacie obvody

- *Je to obvod, ktorý sa dokáže dostať zo stavu 0 do stavu 1 a naopak.*

1. Monostabilný

– Má stabilný iba jeden stav, ak mu dáme povel (impulz), preklopí sa do opačného stavu a po určitom čase sa vráti do svojho stabilného stavu.

– Napríklad: svetlá vchodoch bytovke sú v stabilnej 0, po stlačení vypínača sa dajú do stabilnej 1 a po určitom čase sa vrátia do stabilnej 0 (zhasnú) a infrasnímač.

2. Bistabilný

– Má stabilné oba stavy, ak je v nule a dostane povel, preklopí sa do jednotky a na ďalší povel sa preklopí do nuly.

– Napríklad: mikrospínač na televízory.

3. Astabilný

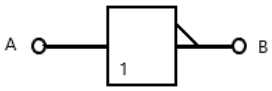
– Tento klopný obvod nemá stabilný ani jeden stav a preklápa sa z 0 do 1 alebo z 1 do 0.

– Napríklad: blikač na bicykli.

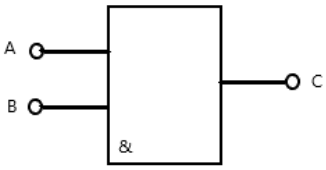
Hradlá

- *Sú základnými stavebnými prvkami logických obvodov.*
- *Každé hradlo má vstupy a jeden výstup.*
- *Pokiaľ hradlo funguje správne, jeho výstupná hodnota je závislá iba na vstupoch a môžeme ju popísať ako jednu zo základných funkcií boolovej algebry.*

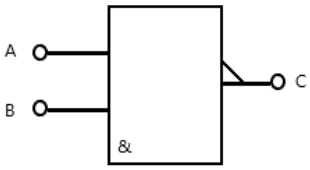
1. Negácia



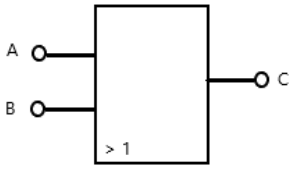
2. AND



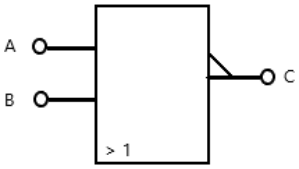
3. NAND



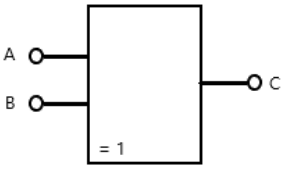
4. OR



5. NOR



6. XOR



B) Programovanie (Softvérová časť)

Programovanie

Programovacie jazyky

- *Programovacie jazyky sú jazyky, ktoré slúžia k tvorbe počítačových programov.*
- *Možno ich rozdeliť z mnohých hľadísk.*
- *Najčastejšie sa však delia takto:*

1. Programovacie jazyky nižšej úrovne

- *Sú primitívne jazyky, ktorých príkazy sú zhodné s príkazmi procesora. To znamená, že procesor vykonáva presne tie inštrukcie, ktoré programátor napíše.*
- *Tieto jazyky sú závislé na svojom procesore a sú neprenositeľné na iný nepríbuzný procesor (napríklad: program z 386-tky na Pentiu pôjde, ale na Atari s procesorom Motorola už nie).*
- *V praxi to vyzerá tak, že programátor musí vypisovať každý detail a aj jednoduchý program má neúmerne zložitý zdrojový kód.*
- *Na druhej strane sa takto programátor dostane i k funkciám počítača ku ktorým sa pomocou vyššieho programovacieho jazyka nedostane.*
- *Patrí sem jazyk symbolických adries ASSEMBLER a strojový kód (to je to čo vidíme, keď sa nám podarí natiahnuť obsah .exe súboru do textového editora).*

2. Programovacie jazyky vyššej úrovne

- *Problémovo orientované programovacie jazyky.*
- *Ich zdrojový kód je oveľa zrozumiteľnejší, jeho štruktúra je logická a nie sú závislé na type procesora, či strojových princípoch počítača.*
- *Patria sem všetky programovacie jazyky okrem ASSEMBLERA, napríklad: Pascal, Basic, Fortran, Prolog, Algol, Imagine, C++, C#, Java a iné.*
- *Často sa uvádza, že jazyk C je akýmsi prechodom medzi nižšími a vyššími programovacími jazykmi, ale bližšie má k vyšším.*

- *Podľa spôsobu prekladania programu delíme programovacie jazyky na:*

1. Interpretované programovacie jazyky

- *Sú programovacie jazyky, ktoré sa prekladajú do strojového kódu priebežne, po riadkoch, teda po jednotlivých príkazoch. Takto preložené príkazy sa hneď aj vykonávajú, preto je vykonávanie programov v interpretovaných programovacích jazykoch pomalšie.*

2. Kompilované programovacie jazyky

- *Tieto jazyky sú prekladané do strojového kódu pomocou kompilátora, ktorý neprekladá každý príkaz samostatne, ale preloží celý zdrojový kód naraz a vytvorí z neho spustiteľný exe súbor a až potom sa tento exe súbor spustí a my vidíme čo sme naprogramovali.*

- *Ďalej môžeme programovacie jazyky rozdeliť na:*

1. Štruktúrované programovacie jazyky

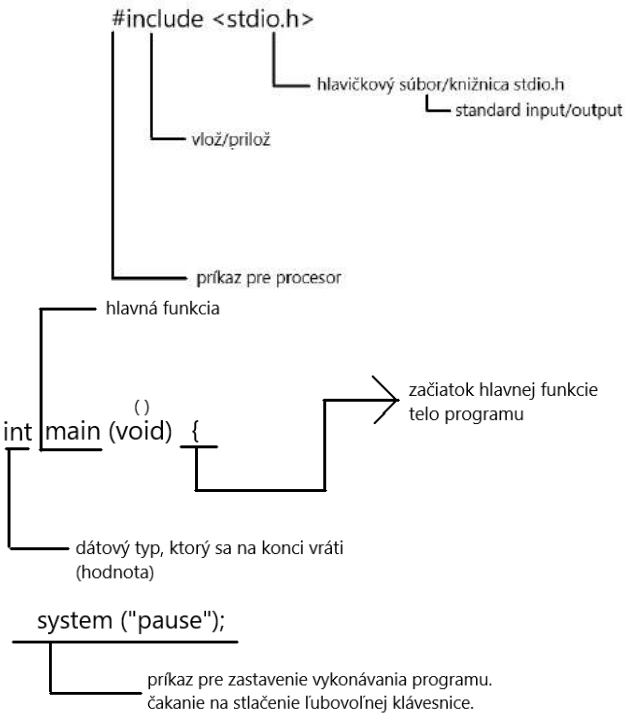
- *Využívajú programové štruktúry, ktoré sa dajú graficky znázorniť pomocou štrukturogramov. Tieto štruktúry vytvárajú nemennú kostru celého programu, tvoria telo hlavnej časti programu a aj telá vnorených podprogramov. U týchto programovacích jazykoch nie je možné vytvárať vlastné dátové štruktúry.*

2. Objektovo orientované programovacie jazyky

- *Pracujú s objektami, ktoré obsahujú premenné a s nimi súvisiace funkcie. Majú svoje vlastnosti alebo zdedené vlastnosti a komunikujú s ostatnými objektami.*

Programovacie jazyky C

- *Jazyk C je štruktúrovaný kompilovaný programovací jazyk.*
- *Programovací jazyk vyššej úrovne.*
- *Základná štruktúra programu DEV C++*




```
system ("pause");
```

príkaz pre zastavenie vykonávania programu.
čakanie na stlačenie ľubovoľnej klávesnice.

```
return 0;
```

funkcií MAIN sa vracia hodnota 0, ak všetko prebehlo korektne, inak sa vracia 1.

```
}
```

→ koniec funkcie MAIN

```
#include <stdio.h>
 int main ( ) {
 príkazy;
 system ("pause");
 return 0;
            }
```

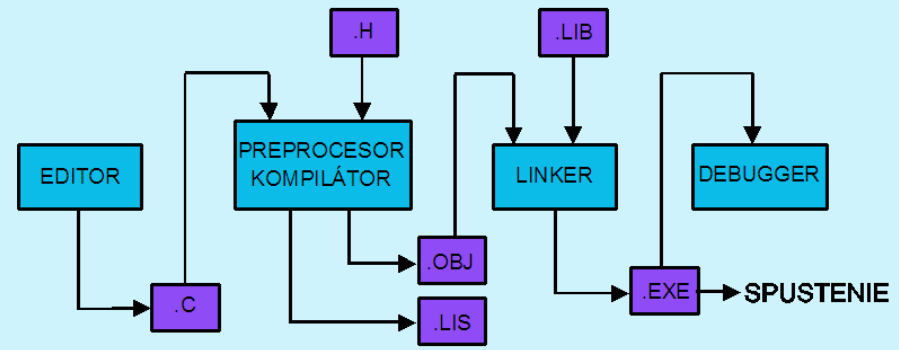
telo programu

```
printf ("Ahoj, ako ide zivot");
```

Jazyk C

- *Jazyk C je univerzálny štruktúrovaný programovací jazyk.*
- *Bol navrhnutý a implementovaný pod operačným systémom UNIX a takmer celý UNIX je aj v C-čku napísaný. Na UNIX sa ale nijako neviaže a neviaže sa ani na žiadne hardvérové alebo softvérové vybavenie počítača (hovoríme že je MULTIPLATFORMNÝ).*

- *Jazyk C je jazykom kompilovaným.*
- *Priebeh kompilácie súboru. Ako prebieha kompilácia, od zdrojového kódu napísaného v nejakom vhodnom editore (odporúčam DEV C++) až po spustiteľný EXE súbor, je znázornené na obrázku.*



EDITOR je aplikácia v ktorej vytvárame zdrojový .C súbor
PREPROCESOR predspracuváva zdrojový kód pre kompilátor. Vkladá do zdrojového súboru hlavičkové súbory z knižnice
KOMPILÁTOR robí preklad zdrojového súboru do relatívneho kódu počítača - vzniká .obj súbor. Relatívny preto, že ešte nie sú známe adresy premenných a funkcií
LINKER *(zostavovací program)* pridelí relatívnym adresám absolútne adresy a priloží funkcie uložené v súboroch .LIB. Vytvára spustiteľný .EXE súbor
DEBUGGER *(ladič)* slúži na ladenie, teda na vyhľadávanie chýb, ktoré vzniknú pri behu programu. Po nájdení a odstránení chyby sa celý proces (EDITOR - PREPROCESOR - KOMPILÁTOR - LINKER - DEBUGGER) opakuje dovtedy, kým DEBUGGER nevyhlasuje žiadnu chybu

- *Tento jazyk patrí k štruktúrovaným programovacím jazykom, čo znamená že využíva iba dané údajové štruktúry nie je možné v ňom vytvárať objekty z reálneho sveta.*

Základné pojmy a zásady v jazyku C

- *C-čko je **case sensitive jazyk**, z toho vyplýva že **rozlišuje** malé a veľké písmená. V praxi to znamená, že premenna, Premenna a PREMENNA sú tri rôzne identifikátory.*

- *Kľúčové slová jazyka C (napr. if, while, register….) **MUSIA** byť písané **malými písmenami**. Ak sú písané veľkými písmenami alebo kombináciou veľkých a malých písmen, neberú sa ako príkazy ale ako identifikátory.*

- *Zdrojový súbor .c ktorý je vytvorený v editore je nutné dopĺňať o **hlavičkové a zdrojové súbory**. Hlavičkové súbory sú súbory, ktoré obsahujú štandardné funkcie vstupu/výstupu, matematické funkcie, operácie na úpravu textových reťazcov a podobne. Tieto súbory .h používame najčastejšie. Vkladáme ich príkazom #include, napríklad #include<stdio.h>*

- ***Biele znaky** (white spaces) sú znaky, ktoré sú dôležité ale na obrazovke ich nevidíme. Sú to tzv. oddeľovacie znaky ako medzera, tabulátor, nový riadok, nová stránka, návrat na začiatok riadka a pod.*

- ***Komentáre** sú dôležitou súčasťnou zdrojového kódu, pretože sprehľadňujú program. Komentár slúži k tomu aby sa v našom zdrojovom kóde vyznal ktokoľvek cudzí alebo i my sami, keď sa k tomuto programu vrátime po nejako čase. Komentár môže byť jednoriadkový //toto je jednoriadkový komentár, alebo viac riadkový /* toto je viac riadkový komentár, ktorý môže byť ľubovoľne dlhý */.*

Dátové typy

Premenná

- *Premenná je miesto v pamäti, ktoré má svoje meno, adresu, dátový typ a hodnotu.*

- *Počas behu programu môže premenná nadobúdať rôzne hodnoty.*
- *Od dátového typu závisi aké veľké miesto v pamäti bude zaberať a aké hodnoty môžeme do nej vkladať.*

Konštanta

- *Konštanta je rovnako ako premenná miesto v pamäti, ktoré má svoje meno, adresu, dátový typ, ale na rozdiel od premennej nemôže počas behu programu meniť svoju hodnotu.*
- *Hodnotu konštantnej môžeme zmeniť pred spustením programu alebo po jeho ukončení.*

Dátové typy v jazyku C

- ***Najbežnejší a najpoužívanejší dátový typ je celé číslo.***

1. Int (integer)

- ***Celé číslo.***
- ***Zaberá v pamäti 2B (- 35 000 až 35 000).***
- ***Tento integer môže byť short integer (short int) alebo long integer (long int).***

2. Float

- *Reálne číslo.*
- *Zaberá v pamäti 4B.*

3. Double

- *Dvojnásobok reálneho číslo.*

4. Long double

5. Char

- *Znak.*
- *Zaberá v pamäti 1B.*
- *Napríklad: „ A “, „ X “, „ & “, „ * “*

Deklarácia a definícia premennej

Deklarácia premennej

- ***Oznamuje procesoru, že budeme používať danú premennú, aby sme na ňu vyhradili v pamäti miesto.***

dátový typ názov premennej;
int vek; ← deklarácia

Definícia premennej

- ***Pri definovaní premennej priradzujeme k premennej hodnotu.***

názov premennej = hodnota;
priradenie

vek = 18; ←definícia premennej

- ***Je možné urobiť súčasne deklaráciu a definíciu premennej súčasne.***

int vek = 18; ← deklarácia s definíciou

Operátor

Matematické operácie

- *Operátor sčítania* *+*
- *Operátor odčítania* *-*
- *Operátor násobenia* ***
- *Operátor delenia* */*
- *Operátor zvyšku po celočíselnom delení* *%* *(nazýva sa modulo)*

Typy delenia:

int/int => int

celočíselné delenie

7 / 2 = 3 *12 / 2 = 6*

zvyšok => 7 % 2 = 1 *12 % 2 = 0*

float/float => float

7 / 2.0 = 3.5

float/int => float

7 / 2 = 3.5

rovná sa = =

Logické operátory

- *Rovná sa* = =
- *Priradenie* =
- *Nerovná sa* ! =
- *Väčší* >
- *Menší* <
- *Väčší alebo rovný* > =
- *Menší alebo rovný* < =
- *And* &&
- *OR* | |

Inkrementácia

- *Zvyšovanie hodnoty premennej o jedna.*

a = 6;
a = 6 + 1; skrátený zápis a++
a => 7;

- *a++ => postfixový zápis*
- *++a => prefixový zápis*

a = 6;
b = a;
b => 6
b = a++;
a => 7
b => 6

- *Najprv sa vykoná priradenie a až potom inkrementácia.*

b = ++a;
a => 7
b => 7

- *Najprv sa a inkrementuje a až následne priradí hodnotu premennej b.*

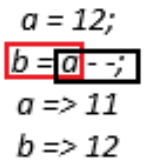
Dekrementácia

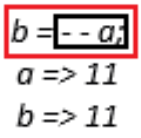
- *Znižovanie hodnoty premennej o jedna.*

a = 7;
a = a -1; skrátený zápis a - -;

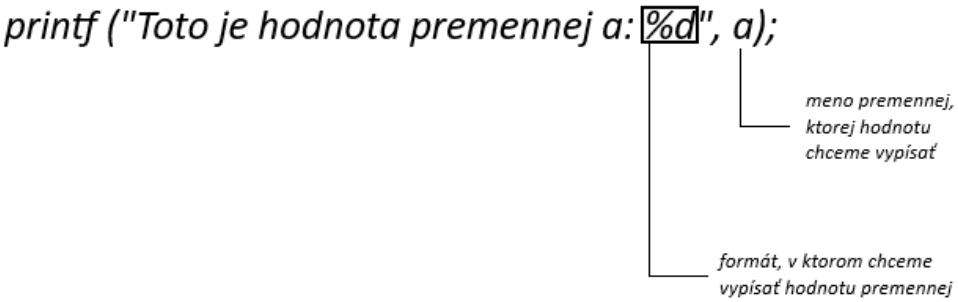
- *a-- => postfixový zápis*

- *--a => prefixový zápis*





- *Strácame efektívnosť, pretože používame moc prostriedkov.*
- *Printf vypisuje na obrazovku.*



Formáty načítavania a výpisu premenných

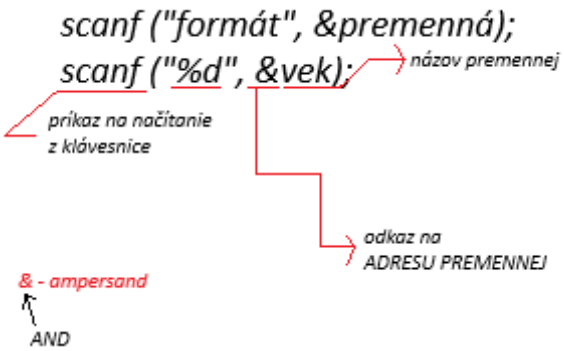
- *Pred každým formátom sa udáva znak %.*

%c => char => znak
%d => int => desiatková sústava
%x => int => šestnástková sústava, malé písmená
%X => int => šestnástková sústava, veľké písmená
%o => int => osmičková sústava
%s => reťazec => niekoľko znakov
%f => float => reálne čísla

Načítanie hodnoty premennej

- *Hodnoty načítame do premennej z klávesnice príkazom scanf.*

Syntax:



Vetvenie

- *Vetvenie používame v programe vtedy, ak máme viacero možností akými môže program pokračovať ďalej.*
- *Vetvenie delíme na jednoduché a úplné.*

Jednoduché vetvenie

- *Používa sa vtedy, ak sa má vykonať nejaká činnosť, ak je podmienka splnená.*
- *Ak podmienka splnená nie je, program ďalej pokračuje vo vykonávaní príkazov.*
- *Príkazy za IF sa vykonajú iba vtedy, ak je podmienka splnená. Ak podmienka splnená nie je, príkazy ignorujú.*

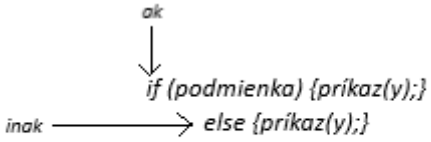
Syntax:

- *Kľúčové slovo if.*

```
if (podmienka)
{príkaz(y);}
```

Úplné vetvenie

- *Úplné vetvenia sa používa vtedy, ak má program niečo vykonať, ak je podmienka splnená a ak má program niečo vykonať aj keď podmienka splnená nie je.*
- *Teda či podmienka splnená je alebo nie je, program vždy vykoná nejaké príkazy.*




```
if (vek>18) {DOSPELÝ}
else {DIEŤA}
```

Zložené podmienky

- *Sú podmienky, ktoré pozostávajú z dvoch a viacerých podmienok.*
- *Môžeme ich spojiť logickým operátorom AND alebo OR.*
- *Ak ich spojíme logickým operátorom AND, musia byť splnené všetky podmienky v zloženej podmienke, aby sa príkaz vykonal.*
- *Ak je spojená logickým operátorom OR, stačí keď bude splnená iba jedna podmienka.*

Case switch

- *V prípadoch, keď máme niekoľko možností, že premenná môže nadobúdať konkrétnu hodnotu, nie je výhodné používať príkaz if, ale príkaz switch case.*
- *Tento príkaz, sa ale nedá použiť, ak chceme premennej nadefinovať nejaký interval, dá sa použiť jedine vtedy, ak premennej definujeme nejakú hodnotu.*

Názorný príklad na ukážku tejto funkcie:

switch-case

```
switch (premenná) {
case 'a': printf („Ahoj\n“); break;
case 'b': printf („Čaves\n“); break;
case 'c': printf („Sewa\n“); break;
default: printf („Nezadal si žiadnu z možností\n“);
}
```

Praktický príklad na ukážku tejto funkcie:

```
switch (typ) {
case 'k': if (váha<=20) {printf („Spaľovňa\n“);}
else {if (váha<=200) {printf („Lokálne smetisko\n“);}
else {printf („Centrálne smetisko\n“);}}
break;
case 's': if (váha<=10) {printf („Recyklácia\n“);}
else {printf („Triedenie\n“);}
break;
default: printf („Nezadali ste korektné údaje.\n“);
}
```

Cykly

- *Cyklus je opakovanie sa nejakého algoritmu (postupnosti príkazov).*
- *Rozlišujeme tri druhy cyklov:*

1. Cyklus so známym počtom opakovaní (FOR)

- *Používame ho vtedy, ak vopred vieme koľkokrát potrebujeme aký cyklus zbehol.*

2. Cyklus s podmienkou na začiatku (WHILE)

- *Cyklus sa vykoná iba vtedy, ak je podmienka splnená, to znamená, že sa nemusí vykonať ani raz.*

3. Cyklus s podmienkou na konci (DO-WHILE)

- *Cyklus sa vykonáva dovtedy, kým je podmienka platná.*
- *POZOR tento cyklus zbehne vždy aspoň raz.*

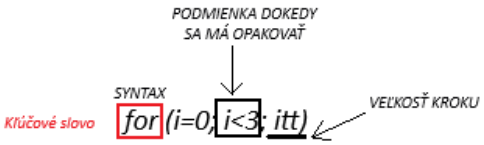
Cyklus so známym počtom opakovaní (FOR)

- *ZADANIE PRÍKLADU: Vytvorte program, ktorý načíta tri premenné a vypíše najväčšiu.*

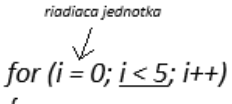
```
#include <stdio.h>
int main () {
int cislo; int max = - 30 000;
int i;
for (i=1; i<4; i++) {
scanf („%d“, &cislo);
if (cislo>max) {max=cislo;}
}
printf („Najväčšie číslo je %d\n“, max);
```

Syntax:

- *Kľúčové slovo for.*




```
{
príkazy;
}
```




```
for (i = 0; i < 5; i++)
{
printf ("Ahoj\n");
}
```

```
Ahoj
Ahoj
```

Generovanie náhodného čísla

```
int main () {
srand (time(NULL));
int a;
a = rand () %10; +1;
```

↳ generuje sa náhodné číslo z rozsahu 0 - 9
1 - 10

Cyklus s podmienkou na začiatku (WHILE)

Syntax:

- Kľúčové slovo while.

```
while (podmienka) {
telo cyklu (príkazy;)
}

while (a<10) {
b++;
a++;
}
```

Cyklus s podmienkou na konci (DO-WHILE)

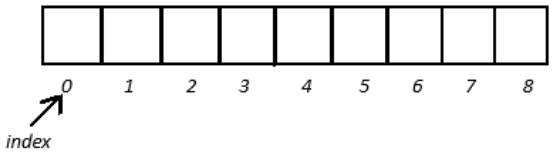
Syntax:

```
do {
príkaz (y);
} while (podmienka);

do {
scanf („%d“, &x);
} while (x<10);
```

Polia

int teplota [9]



teplota [2] = 34;

```
for (i=0; i<9; i++)
{scanf ("%d", &teplota[i]);
}
```

Praktický príklad na ukážku tejto funkcie:

```
#include <stdio.h>
//Vytvorte program ,ktorý vygeneruje 30 náhodných čísel do 100 a uloží ich do poľa. Ďalej program načíta 10 čísel od užívateľa a tiež ich vloží do ďalšieho poľa. Program zistí či sú čísla ,ktoré sú v jednom aj druhom poli vypíše ich.
int main ()  {
   srand (time(NULL));
   int i, j=0;
   int nacitane [10];
   int gener [30];
   int cisla [40];

   printf ("Zadaj 10 nahodnych cisel.\n");

   for (i=0; i<10; i++) {
   scanf ("%d", &nacitane[i]);
   printf (" * %d * \n", nacitane [i]);
}

  printf ("\nProgram vygeneruje 30 nahodnych cisel do 100.\n");

  for (j=1; j<31; j++) {
     gener [j] = rand () %100; +1;
     printf ("Hodnota %d premennej gener je: * %d *\n", j, gener [j]);
     }

   for (j=0; j<30; j++) {
      for (i=0; i<10; i++) {
   if (nacitane [i]==gener [j]) {printf ("* %d * ", nacitane [i]);}
}}
   system ("pause");
   return 0;
}
```

Funkcie

- *Funkcie sú vlastne podprogramy, ktoré nám uľahčujú prácu tým, že funkciu nedefinujeme iba raz, ale v tele programu ju môžeme zavolať (použiť) ľubovoľný počet krát.*
- *Tým prechádzame k tomu, aby sme v zdrojovom kóde opakovali, písali tie isté príkazy.*
- *Funkciu nikdy nedefinujeme v inej funkcii.*

Názorný príklad na ukážku tejto funkcie:

```
#include <stdio.h>
int spocitaj (void);  <= deklarácia funkcie
int main () {                          volanie funkcie
int c; c = spocitaj (void);
return 0;
}
int spocitaj (void) {  <= definícia funkcie
int a, b;
scanf („%d“, &a);
scanf („%d“, &b);
return (a + b);
}
```

Funkcie s parametrom

- *Funkcia s parametrom narozdiel od funkcie bez parametra posiela funkcii vstupné hodnoty, ktoré je potrebné spracovať.*

Praktický príklad na ukážku tejto funkcie:

//Vytvorte program ,ktorý vypočíta priemer z dvoch načítaných známok.

```
#include <stdio.h>
float priemer (float a, float b);
int main () {
   float c1, c2;
   float priemer1;

   scanf ("%f", &c1);
   scanf ("%f", &c2);

   priemer1 = priemer (c1, c2);

   printf ("Priemer známok je %0.02f.\n", priemer1);

   system ("pause");
   return 0;
}

float priemer (float a, float b) {
    return ((a+b)/2);
    }
```