9. lekce

Úvod do jazyka C – 3. část

Základní příkazy jazyka C

Miroslav Jílek

Základní příkazy

Všechny příkazy se píšou malými písmeny!
Za většinou příkazů musí být středník (;)!

Základní příkazy

Všechny příkazy se píšou malými písmeny!
Za většinou příkazů musí být středník (;)!

Příkaz IF

if (podmínka) jeden_příkaz; else jeden_příkaz;

Když je podmínka (condition) True, pak se vykoná příkaz za podmínkou. Když je false splněna, pak se vykoná příkaz za Else. Else nemusí být použito.

Základní příkazy

Všechny příkazy se píšou malými písmeny!
Za většinou příkazů musí být středník (;)!

Příkaz IF

if (podmínka) jeden_příkaz; else jeden_příkaz;
Je-li podmínka True, pak se vykoná příkaz za podmínkou. Není-li splněna, pak se vykoná příkaz za Else. Else nemusí být použito.

```
if (podmínka)
{
    několik_příkazů
}
else
{
    několik příkazů
}
```

Stejný význam jako u jednoduchého příkazu if, rozdíl je v tom, že se vykonají všechny příkazy ve složených závorkách.

```
if (podmínka1)
{
    několik příkazů
}
else if (podmínka2)
{
    několik příkazů
}
else
{
    několik příkazů
}
```

Stejný význam jako u jednoduchého příkazu if, rozdíl je ve vloženém příkazu else if (podmínka2), který funguje jako samostatný if a je použit v případě, kdy je první podmínka false.

V případě složené podmínky se používají kulaté závorky!

Např.: ***((A>10) &&((B<=6)||(C==0)))***
((A>10) AND ((B<=6) OR (C==0)))

… to znamená:

((A>10) a také ((B<=6) nebo (C==0)))

Pozor!!!
Pokud zapíšeme (C=0), pak se do proměnné C přiřadí nula a pravdivostní hodnota je vždy false!!!

SWITCH

```
switch(proměnná)
{
    case hodnota1:
        příkazy
        break;
    case hodnota2:
        příkazy
        break;
    default:
        příkazy
}
```

SWITCH

```
switch(proměnná)
{
    case hodnota1:
        příkazy
        break;
    case hodnota2:
        příkazy
        break;
    default:
        příkazy
}
```

*Příkaz testuje obsah proměnné **proměnná** a pokud se rovná **hodnotě1**, pak se vykonají příkazy za příslušným **case**. **Break** končí blok příkazů **case**.*

Default** se provede pouze v případě, že testovaná proměnná neobsahuje žádnou z hodnot za příkazy **case**, tedy neobsahuje hodnotu **hodnota1** ani **hodnota2.

FOR

Příkaz cyklu.

for (proměnná; podmínka; krok) jeden_příkaz;

nebo

```
for (proměnná; podmínka; krok)
{
    příkazy
}
```

proměnná - řídící proměnná cyklu a její počáteční hodnota, proměnná může být int, double, char a pointer!
podmínka - dokud je podmínka true, cyklus se provádí
krok - hodnota, o kterou se řídící proměnná mění

Příklady použití příkazu FOR:

int a;
Proměnná a musí být definována před použitím v cyklu!

Příklady použití příkazu FOR:

int a;
Proměnná a musí být definována před použitím v cyklu!

for (a= -1; a < 10; a++) jeden_příkaz;

*Proměnná **a** nabývá hodnot **-1** až **9**, cyklus začíná **a = -1**, **postupně se a zvyšuje o jedničku** a **končí 9**. Po ukončení příkazu FOR je v proměnné a hodnota **10!***

Příklady použití příkazu FOR:

int a;
Proměnná a musí být definována před použitím v cyklu!

for (a= -1; a < 10; a++) jeden_příkaz;

*Proměnná **a** nabývá hodnot **-1** až **9**, cyklus začíná **a = -1**, **postupně se a zvyšuje o jedničku** a **končí 9**. Po ukončení příkazu FOR je v proměnné a hodnota **10**.*

int a[10];
int *b;
for (b=a; b < a+sizeof(a); b++) jeden_příkaz;

b nabývá hodnot adres jednotlivých prvků v poli a, postupně se a zvyšuje o jednu adresu a končí adresou prvku a[9]. Po ukončení příkazu FOR je v proměnné b adresa za poslední prvek pole a (první adresa, která už není v poli a).

V prvním cyklu bude v proměnné b adresa prvního prvku pole a. V každém dalším cyklu se k adrese připočte 4 (4 byty integeru) – např:

00fa7510, 00fa7514, 00fa7518, 00fa751c, 00fa7520, 00fa7524, 00fa7528, 00fa752c,...

b++ je posun pointeru v paměti, ve které jsou uloženy hodnoty pole a.

for (;1;) jeden_příkaz;

Nekonečný cyklus. Takový cyklus můžeme ukončit příkazem break, např.:

For (;1;) jeden_příkaz;
Nekonečný cyklus. Takový cyklus můžeme ukončit příkazem break, např.:

for (;1;) if (a<10) a++; else break;
K hodnotě proměnné a, která musí být deklarována před cyklem, přičítáme jedničku, dokud je a < 10, když je podmínka false, tak se cyklus zastaví – break – výstup z cyklu.

break *– předčasné ukončení (opuštění) cyklu*
continue *– pokračování cyklu další smyčkou (loop), příkazy v těle cyklu za Continue se v dané smyčce nevykonají*

Příklad cyklu for s proměnnou typu char a double

```
#include <stdio.h>
int main (void)
{
	char A;
	double B;
	for(A='a'; A<='z'; A++)
	{
		printf("%c",A);
	}
	printf("\n");
	for(B=-0.2; B<=0.2; B+=0.1)
	{
		printf("%.2f * ",B);
	}
	return 0;
}
```

```
abcdefghijklmnopqrstuvwxyz
-0.20 * -0.10 * 0.00 * 0.10 * 0.20 *
```

WHILE

Příkaz cyklu.

while(podmínka)
{
    příkazy;
}

nebo

do
{
    příkazy;
}
while(podmínka)

Cyklus se provádí, pouze když je podmínka true!
*Cyklus **do while** se provede vždy alespoň jednou!*

PRINTF

Slouží k zobrazení výstupů na konzoli.

PRINTF

Slouží k zobrazení výstupů na konzoli.

printf(„formátovací řetězec“, argumenty_oddělené_čárkami);

PRINTF

Slouží k zobrazení výstupů na konzoli.

printf(„formátovací řetězec“, argumenty_oddělené_čárkami);

Formátovací řetězce: Vlastní text, do kterého dosadíme hodnoty proměnných pomocí konverzních znaků.

PRINTF

Slouží k zobrazení výstupů na konzoli.

printf(„formátovací řetězec“, argumenty_oddělené_čárkami);

Formátovací řetězce: Vlastní text, do kterého dosadíme hodnoty proměnných pomocí konverzních znaků.

Konverzní znaky:

%d - celé číslo

PRINTF

Slouží k zobrazení výstupů na konzoli.

printf(„formátovací řetězec“, argumenty_oddělené_čárkami);

Formátovací řetězce: Vlastní text, do kterého dosadíme hodnoty proměnných pomocí konverzních znaků.

Konverzní znaky:

%d	- celé číslo
%5d	- celé číslo, doplněné zleva nulami na 5 znaků, pokud má více znaků, zobrazí se všechny

PRINTF

Slouží k zobrazení výstupů na konzoli.

printf(„formátovací řetězec“, argumenty_oddělené_čárkami);

Formátovací řetězce: Vlastní text, do kterého dosadíme hodnoty proměnných pomocí konverzních znaků.

Konverzní znaky:

%d	- celé číslo
%5d	- celé číslo, doplněné zleva nulami na 5 znaků, pokud má více znaků, zobrazí se všechny
%h	- celé číslo v šestnáctkové soustavě

PRINTF

Slouží k zobrazení výstupů na konzoli.

printf(„formátovací řetězec“, argumenty_oddělené_čárkami);

Formátovací řetězce: Vlastní text, do kterého dosadíme hodnoty proměnných pomocí konverzních znaků.

Konverzní znaky:

%d	- celé číslo
%5d	- celé číslo, doplněné zleva nulami na 5 znaků, pokud má více znaků, zobrazí se všechny
%h	- celé číslo v šestnáctkové soustavě
%b	- celé číslo ve dvojkové soustavě

PRINTF

Slouží k zobrazení výstupů na konzoli.

printf(„formátovací řetězec“, argumenty_oddělené_čárkami);

Formátovací řetězce: Vlastní text, do kterého dosadíme hodnoty proměnných pomocí konverzních znaků.

Konverzní znaky:

%d	- celé číslo
%5d	- celé číslo, doplněné zleva nulami na 5 znaků, pokud má více znaků, zobrazí se všechny
%h	- celé číslo v šestnáctkové soustavě
%b	- celé číslo ve dvojkové soustavě
%c	- jeden znak (jako proměnná typu char)

PRINTF

Slouží k zobrazení výstupů na konzoli.

printf(„formátovací řetězec“, argumenty_oddělené_čárkami);

Formátovací řetězce: Vlastní text, do kterého dosadíme hodnoty proměnných pomocí konverzních znaků.

Konverzní znaky:

%d	- celé číslo
%5d	- celé číslo, doplněné zleva nulami na 5 znaků, pokud má více znaků, zobrazí se všechny
%h	- celé číslo v šestnáctkové soustavě
%b	- celé číslo ve dvojkové soustavě
%c	- jeden znak (jako proměnná typu char)
%s	- pole znaků

PRINTF

Slouží k zobrazení výstupů na konzoli.

printf(„formátovací řetězec“, argumenty_oddělené_čárkami);

Formátovací řetězce: Vlastní text, do kterého dosadíme hodnoty proměnných pomocí konverzních znaků.

Konverzní znaky:

%d	- celé číslo
%5d	- celé číslo, doplněné zleva nulami na 5 znaků, pokud má více znaků, zobrazí se všechny
%h	- celé číslo v šestnáctkové soustavě
%b	- celé číslo ve dvojkové soustavě
%c	- jeden znak (jako proměnná typu char)
%s	- pole znaků
%20s	- zobrazí se 20 znaků zleva doplněné mezerami, pokud proměnná obsahuje více znaků, zobrazí se všechny

PRINTF

Slouží k zobrazení výstupů na konzoli.

printf(„formátovací řetězec“, argumenty_oddělené_čárkami);

Formátovací řetězce: Vlastní text, do kterého dosadíme hodnoty proměnných pomocí konverzních znaků.

Konverzní znaky:

%d	- celé číslo
%5d	- celé číslo, doplněné zleva nulami na 5 znaků, pokud má více znaků, zobrazí se všechny
%h	- celé číslo v šestnáctkové soustavě
%b	- celé číslo ve dvojkové soustavě
%c	- jeden znak (jako proměnná typu char)
%s	- pole znaků
%20s	- zobrazí se 20 znaků zleva doplněné mezerami, pokud proměnná obsahuje více znaků, zobrazí se všechny
%f	- desetinné číslo (float)

%5.3f - desetinné číslo, doplněné zleva nulami na 5 pozic před desetinnou čárkou (tečkou) a zprava nulami na tři pozice za desetinnou čárkou, pokud je za desetinnou čárkou více pozic, desetinné pozice se vypíšou pouze tři, bez zaokrouhlení

%5.3f - desetinné číslo, doplněné zleva nulami na 5 pozic před desetinnou čárkou (tečkou) a zprava nulami na tři pozice za desetinnou čárkou, pokud je za desetinnou čárkou více pozic, desetinné pozice se vypíšou pouze tři, bez zaokrouhlení

%lf - desetinné číslo (double, long float)

%5.3f	- desetinné číslo, doplněné zleva nulami na 5 pozic před desetinnou čárkou (tečkou) a zprava nulami na tři pozice za desetinnou čárkou, pokud je za desetinnou čárkou více pozic, desetinné pozice se vypíšou pouze tři, bez zaokrouhlení
%lf	- desetinné číslo (double, long float)
\n	- nový řádek

%5.3f	- desetinné číslo, doplněné zleva nulami na 5 pozic před desetinnou čárkou (tečkou) a zprava nulami na tři pozice za desetinnou čárkou, pokud je za desetinnou čárkou více pozic, desetinné pozice se vypíšou pouze tři, bez zaokrouhlení
%lf	- desetinné číslo (double, long float)
\n	- nový řádek
\	- zrušení významu konverzního znaku (např., pokud chceme mít ve výstupu symbol % nebo \ musíme před ním mít zpětné lomítko, tedy \%, \\

%5.3f	- desetinné číslo, doplněné zleva nulami na 5 pozic před desetinnou čárkou (tečkou) a zprava nulami na tři pozice za desetinnou čárkou, pokud je za desetinnou čárkou více pozic, desetinné pozice se vypíšou pouze tři, bez zaokrouhlení
%lf	- desetinné číslo (double, long float)
\n	- nový řádek
\	- zrušení významu konverzního znaku (např., pokud chceme mít ve výstupu symbol % nebo \ musíme před ním mít zpětné lomítko, tedy \%, \\
\t	- posunutí – tabulátor

Příklady použití příkazu PRINTF:

a) **int a=5;**
printf(“Hodnota proměnné \“a\“ je %d.\n“,a);

Vypíše: ***Hodnota proměnné “a“ je 5****.*
Kurzor bude na ***novém řádku****.*

Příklady použití příkazu PRINTF:

a) **int a=5;**
printf(“Hodnota proměnné \“a\“ je %d.\n“,a);
Vypíše: ***Hodnota proměnné “a“ je 5****.*
Kurzor bude na ***novém řádku****.*

b) **double a=5, b=6.6;**
printf(“%1.3lf+%lf=%1.2lf\n“, a, b,a+b);
Vypíše: ***5.000+6.600000=11.60***
Kurzor bude na novém řádku. Pokud není uveden počet desetinných míst, pak je použita defaultní hodnota 6.

Příklady použití příkazu PRINTF:

a) **int a=5;**
printf(“Hodnota proměnné \“a\“ je %d.\n“,a);
Vypíše: ***Hodnota proměnné “a“ je 5****.*
Kurzor bude na ***novém řádku****.*

b) **double a=5, b=6.6;**
printf(“%1.3lf+%lf=%1.2lf\n“, a, b,a+b);
Vypíše: ***5.000+6.600000=11.60***
Kurzor bude na novém řádku. Pokud není uveden počet desetinných míst, pak je použita defaultní hodnota 6.

c) **char a=48, b='1';**
printf(“%d %d %c %c“, a, b, a, b);
Vypíše: ***48 49 0 1***
To je ascii kód symbolu a symbol, který tento ascii kód reprezentuje.

SCANF

Čte hodnoty z vstupu – klávesnice.

scanf(„formátovací řetězec“, ukazatele na proměnné oddělené čárkami);
Má návratovou hodnotu určující počet úspěšných konverzí.
Ignoruje „bílé“ znaky, tedy mezerník, enter a tabulátor.

SCANF

Čte hodnoty z vstupu – klávesnice.

scanf(„formátovací řetězec“, ukazatele na proměnné oddělené čárkami);
Má návratovou hodnotu určující počet úspěšných konverzí.
Ignoruje „bílé“ znaky, tedy mezerník, enter a tabulátor.

Příklady:

a) **int a;**
scanf(“%d“,&a);

Načte celočíselnou hodnotu a vloží ji do proměnné a – na adresu v RAM, kde je proměnná a.

SCANF

Čte hodnoty z vstupu – klávesnice.

scanf(„formátovací řetězec“, ukazatele na proměnné oddělené čárkami);
Má návratovou hodnotu určující počet úspěšných konverzí.
Ignoruje „bílé“ znaky, tedy mezerník, enter a tabulátor.

Příklady:

a)
```
int a;
scanf(“%d“,&a);
```

Načte celočíselnou hodnotu a vloží ji do proměnné a – na adresu v RAM, kde je proměnná a.

b)
```
if (scanf(“%d %d“,&a, &b)==2)
{
    příkazy
}
```

Načte hodnotu do proměnné „a“ a „b“ pokud se konverze podaří u obou proměnných, pak provede tělo příkazu if. 2 znamená obě konverze!

STRLEN

- zjišťuje (vrací) délku pole charů, které je uvedeno v parametru
- (poslední znak pole je znak \0, za tímto znakem jsou další indexy ignorovány)

int a;
a=strlen(“Nějaký řetězec znaků nebo pole znaků“);
//výsledek bude 36

STRCPY

- kopíruje pole charů do jiného pole charů
- funkce má dva parametry, první je cíl, druhý zdroj

```
char *a="Ahoj studente";
char b[20];
strcpy(b,a);
```

STRCMP

- porovnává řetězce charů, výsledek je integer, který je záporný, když je první řetězec menší, nula, když jsou řetězce stejné a kladný, když je druhý řetězec menší. Vrácené číslo může být libovolné.
- funkce má dva parametry, dvě pole charů

int c;
char *a=“Ahoj studente”;
c=strcmp(a, “Ahoj studenti“);
// c bude mít zápornou hodnotu

Souhrn hlavních příkazů a potřebných knihoven

Knihovna	***Příkazy***
stdio.h	printf, scanf
stdlib.h	malloc, realloc, calloc
math.h	pow*(základ, exponent)*, sqrt, sin, cos, abs, fabs
assert.h	assert*(porovnání) – logická (algoritmická) kontrola programu*
string.h	strlen, strcpy, strcmp
Příkazy if, for, while, sizeof, return *jsou přímo v C a nepotřebují žádnou knihovnu*	

Testování správnosti algoritmu

Do kódu programu přidáme řádek s assertem.

Jestli je podmínka true, pak program pokračuje bez chyby, jestli je false, pak se zastaví a vypíše chybu:

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

int obsahctverce (int strana)
{
    return strana*strana;
}

int main (void)
{
    assert(obsahctverce(5)==25);
    assert(obsahctverce(8)==35);
    return 0;
}
```

```
C:\Users\Správce\Desktop\ČVUT-FIT\Přednášky UJOP\1. semestr\Programy C>a.exe
Assertion failed: obsahctverce(8)==35, file assert.c, line 12
```

Změření času běhu programu

Pro měření času exekuce programu (zpracování dat) vytvoříme dávkový soubor (*.BAT). Tento soubor umístíme do stejné složky, ve které máme program (a.exe) a soubor dat (data.txt). Spustíme dávkový soubor a ten nám na konzoli vypíše čas startu a čas dokončení exekuce. Z rozdílu těchto časů vypočítáme čas běhu programu (zpracování dat).

Obsah dávkového souboru:

```
echo %time%
a.exe <data.txt
echo %time%
```

(Vytvoříme ho v poznámkovém bloku a uložíme s koncovkou BAT, spouštíme ho z konzole příkazem ***jmeno_souboru.bat***)