

# Programovací jazyk C

## 1.1. Preprocesor

Konstrukce preprocesoru C:

- Definování makra:  
`#define jméno text rozvoje`
- Makro s parametrem  
`#define jméno(p0, p2, ...) rozvoj p0 p1 ...`
- Makro s textovým parametrem  
`#define jméno(par) printf("par =" #par)`
- Zrušení definice:  
`#undef jméno`
- Podmíněný překlad na hodnotě *konstanta*  
`#if konstanta`  
`#elif #else #endif`
- Vložení souboru z adr. uživatele  
`#include "soubor"`
- Vložení systémového souboru  
`#include <soubor>`
- Podmíněný překlad, pokud je makro definováno  
`#ifdef makro`  
`#elif #else #endif`
- Podmíněný překlad, pokud je makro nedefinováno  
`#ifndef makro`  
`#elif #else #endif`
- Předdefinovaná makra: `__LINE__`, `__FILE__`, `__TIME__`, `__DATE__`, `__STDC__`

## 1.2. Proměnné

- Celočíselné typy:
  - Se znaménkem (**signed**):
    - \* short (short int, signed short, signed short int)
    - \* int (signed int, signed)
    - \* long (long int, signed long, signed long int)
  - Bez znaménka (**unsigned**):
    - \* unsigned short
    - \* unsigned int
    - \* unsigned long
- Znakový typ:
  - char (signed char)
  - unsigned char
- Reálný typ:
  - float – 32bitová přesnost
  - double (long float) – 64bitová přesnost
  - long double – 80bitová přesnost
- Ukazatel:
  - void \*prt – generický ukazatel
  - (void \*)0 – NULL
- Pole:
  - *typ* pole[] –
- Výčtový typ:
  - enum
- Struktura:

```
struct název {  
    typ název1;  
    typ název2;  
    unsigned a:4; /* bitové pole o délce 4 bity */  
};
```

```
typedef struct NÁZEV {  
    typ název2;  
    struct *NÁZEV ukazatel;  
} název;
```

- Union
- Prázdný typ: void
- Paměťové třídy (před typ proměnné):
  - auto – lokální prom. na zásobníku,
  - extern – globální prom. v dat. oblasti,
  - static – lok. prom., ponechání hodnoty mezi jednotlivými voláními funkce,
  - register – lok. prom., uložení v registru počítače.

## 1.3. Výrazy

Čím větší číslo v indexu tím vyšší priorita.

- Unární výrazy:
  - *prom*++<sub>16</sub>, *prom*--<sub>16</sub>
  - ++*prom*<sub>15</sub>, --*prom*<sub>15</sub>
  - !<sub>15</sub> – log. negace
  - ~<sub>15</sub> – bit. negace (~0xFOF0=0x0FOF)
  - &<sub>15</sub> – adresový operátor
  - \*<sub>15</sub> – nepřímý operátor
  - (*typ*)<sub>14</sub> *proměnná* – přetypování
- Binární výrazy:
  - multiplikativní: \*<sub>13</sub> (násobení), /<sub>13</sub> (dělení), %<sub>13</sub> (zbytek po dělení)
  - aditivní: +<sub>12</sub> (plus), -<sub>12</sub> (mínus)
  - posuny: <<<sub>11</sub> (doleva), >><sub>11</sub> (doprava)
  - rovnosti: ==<sub>9</sub> (je roven), !=<sub>9</sub> (není roven)
  - relace: <<sub>10</sub>, <=<sub>10</sub>, ><sub>10</sub>, >=<sub>10</sub>
  - bitové: &<sub>8</sub> and, |<sub>6</sub> or, ^<sub>7</sub> xor
  - logické: &&<sub>5</sub> and, ||<sub>4</sub> or
- Podmíněný výraz<sub>3</sub>: *výraz-logické-or* ? *výraz* : *podmíněný výraz*
  - r = a ? b : c; ⇔
  - ⇔ if(a!=0) r=b; else r=c;
  - a ? b : c ? d : e ? f : g ⇔
  - a ? b : (c ? d : (e ? f : g))
- Dosazovací výrazy<sub>2</sub>:  
= += -= \*= /= %= <<= >>= &= ^= |=  
(priorita 2, asociativní z prava)
  - a op= b*    *a = a op b*
  - x\*=y=z*    *x\*=(y=z)*
  - a=b+d+7*   *a=(b=(d+7))*
- Příklady:
  - ++e ⇔ e+=1 – inkrem. před použitím
  - e++ – inkrementace po použití
  - --e ⇔ e-=1 – dekrem. před použitím
  - j = ++i ⇔ j = (i = (i + 1))
  - j = i-- ⇔ j = i; i = i - 1;

## 1.4. Příkazy

```
if (výraz1)
    příkaz1;
else if (výraz2)
    příkaz2;
else if (výraz3)
    příkaz3;
...
else
    příkazn;

while (výraz)
    příkaz;

do příkaz
    while (výraz);
    (oproti while se tělo
    cyklu provede alespoň jednou)
```

```
for (výraz1; výraz2; výraz3)
    příkaz;
```

```
switch (výraz){
    case konstanta: příkaz;
    break;
    default : příkaz;
}
```

```
goto návěští;
návěští:
    příkaz;
```

## 1.5. Některé funkce

### 1.5.1. Řídící znaky:

sekvence	hodnota	význam
<code>\n</code>	0x0A	nová řádka (LF)
<code>\r</code>	0x0D	návrat vozíku (CR)
<code>\f</code>	0x0C	nová stránka (FF)
<code>\t</code>	0x09	tabulátor (HT)
<code>\b</code>	0x08	posun doleva (BS)
<code>\a</code>	0x07	písknutí (BELL)
<code>\\</code>	0x5C	zpětné lomítko
<code>\'</code>	0x2C	apostrofový znak (single quote)
<code>\0</code>	0x00	nulový znak (NULL)

### 1.5.2. Formátové specifikace:

% [příznaky][šířka][.přesnost][modifikátor]konverze

- Konverze:

<code>%c</code>	znak
<code>%d</code>	desítkové číslo signed int
<code>%ld</code>	desítkové číslo signed long
<code>%u</code>	desítkové číslo unsigned int
<code>%lu</code>	desítkové číslo unsigned long
<code>%llu</code>	desítkové číslo unsigned long long
<code>%f</code>	float
<code>%Lf</code>	long double
<code>%lf</code>	double
<code>%x</code>	hexadecimální číslo, např. 1a2b
<code>%X</code>	hexadecimální číslo, např. 1A2B
<code>%o</code>	osmičkové číslo
<code>%s</code>	řetězec
<code>%p</code>	adresa argumentu
<code>%n</code>	netiskne, ukládá počet znaků

- Modifikátor konverze:

<code>h, l</code>	<code>d, i</code> → signed short int; <code>u, o, x, X</code> → unsigned short int
<code>L</code>	<code>f, e, E, g, G</code> → long double

- Šířka:
  - `n` tiskne alespoň `n` znaků, mezery zprava
  - `0n` tiskne alespoň `n` znaků, nuly doplňuje zleva
  - `*` počet znaků udává předchozí argument
- Přesnost:
  - `d, i, u, o, x, X` minimální počet cifer na výstupu
  - `f, e, E` počet cifer za desetinnou tečkou
  - `g, G` maximální počet významových cifer
  - `s` počet tištěných znaků
  - `*` počet znaků udává předchozí argument
- Příznak:
  - zarovnáva doleva a zprava doplňuje mezery
  - + tiskne číslo vždy se znaménkem `+/-`

### 1.5.3. Ukazatele

Příklady:

```
*poi = 5; - na adresu se zapíše 5
i = *poi; - získá se obsah na adrese
poi = &i; - ukazatel na adresu i
i = poi; - do i se uloží adresa poi
(* poi)++; - hodnota na adrese se inkrementuje
double (* p_fd)(); - ukazatel na funkci vracející double
```

```
int **xd; - xd je ukazatel na ukazatel na int,
            *xd je ukazatel na int,
            **xd je prvek int
```

Pole:

```
vícerozměrné - pole[řádek][sloupec]
x[i] ⇔ *(x+i)
x[i][j] ⇔ *(x[i]+j) ⇔ *(*x+i)+j
```

Alokace paměti:

```
int *poi;
poi = (int *) malloc(4*sizeof(int)) - přidělení
poi[0] ⇔ *p_i, poi[1] ⇔ *(poi+1)
free(poi) - uvolnění paměti
```

## 1.6. Příkazy gdb

Při použití gcc kompilovat s parametrem `-ggdb3` a bez optimalizací.

```
run argument - spustí program s argumentem
bt - (backtrace) vypíše použité funkce před pádem
b funkce - breakpoint na jméno funkce
b n - breakpoint na řádku číslo n
tb ... - dočasný breakpoint
CTRL+C - zastaví spuštěný program
info breakpoints - seznam nastavených breaků
info locals - seznam lokálních proměnných
delete n - zruší break číslo n
p jméno - vypíše obsah proměnné jméno
display - vypíše obsah sledovaných objektů
p* ukazatel - vypíše obsah na adrese ukazatele
up/down - pohyb po stack-framu
next - krokování programu
step - krokování se zanořuje do funkce
list n - výpis zdrojáka okolo n. řádky
thread - přepínání mezi vlákny
```