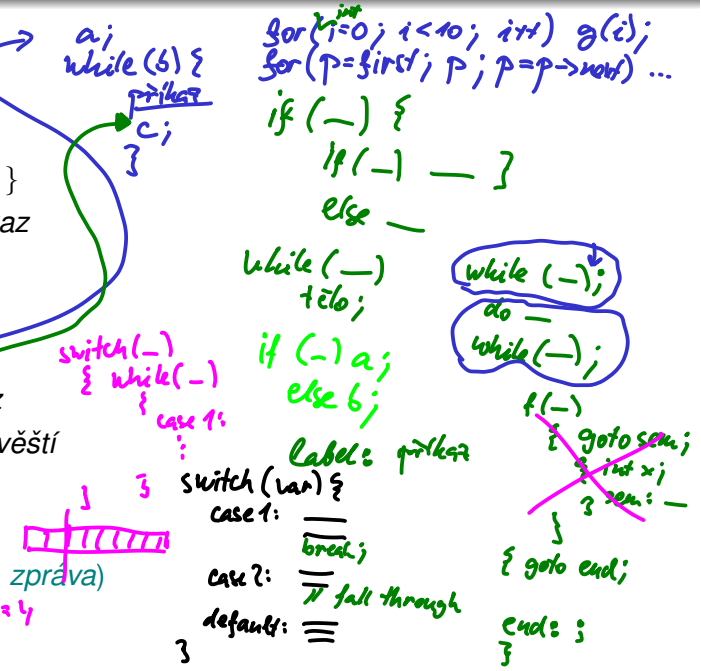
Příkazy

- prázdný příkaz `;`
- *výraz* `;`
- `{` *příkaz*`;` *deklarace*`;` `...` `}`
- `if (`*výraz*`)` *příkaz* `else` *příkaz*
- `while (`*výraz*`)` *příkaz*
- `do` *příkaz*`;` `while (`*výraz*`)`
- `for (`*a*`;` *b*`;` *c*`)` *příkaz*
- `for (`*deklarace*`;` *b*`;` *c*`)` *příkaz*
- `break`, `continue`, `goto` *návěští*
- `return` *výraz*
- `switch`, `case`, `default`
- `_Static_assert`(*podmínka*, *zpráva*) C11



Funkce s proměnlivým počtem argumentů

```
#include <stdarg.h>

int f(char *s, ...) {
  va_list args;
  va_start(args, s);
  while (*s)
    switch (*s++) {
      case 'i': got_int(va_arg(args, int));
                break;
      case 's': got_str(va_arg(args, char *));
                break;
    }
  va_end(args);
}
```

(pozor na implicitní typové konverze)

Zpracování argumentů vícekrát

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

char *memprintf(char *s, ...) {
  va_list args;
  va_start(args, s);

  int len = vsnprintf(NULL, 0, s, args) + 1;
  char *p = malloc(len);
  vsnprintf(p, len, s, args);

  va_end(args);
  return p;
}
```

Zpracování argumentů vícekrát

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

char *memprintf(char *s, ...) {
  va_list args, args2;
  va_start(args, s);
  va_copy(args2, args);
  int len = vsnprintf(NULL, 0, s, args2) + 1;
  char *p = malloc(len);
  vsnprintf(p, len, s, args);
  va_end(args2);
  va_end(args);
  return p;
}
```

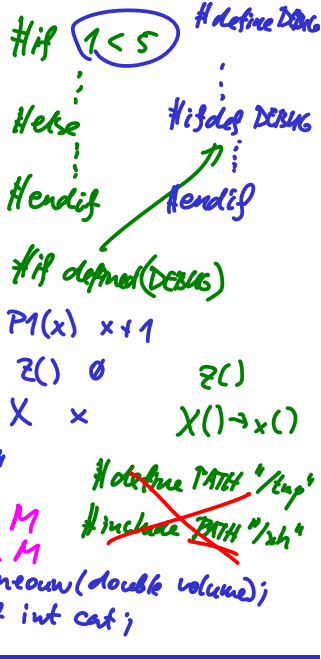
Fáze překladu

1. Konverze do kompilační znakové sady a nahrazení trigrafů (`??=` → `#` apod.)
2. Slepení řádků končících na '\'
3. Rozdělení na pp-tokeny a mezery, komentáře → mezery
4. Preprocesor (direktivy, makra, includy)
5. Konverze do cílové znakové sady
6. Slepení navazujících stringových literálů
7. Konverze pp-tokenů na tokeny, zapomenutí mezer
8. Syntaktická a sémantická analýza
9. Linker

Direktivy preprocesoru

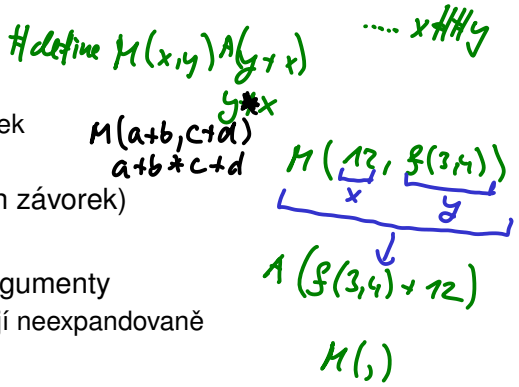
<mezery> <direktiva ...>

- `#if` výraz, `#else`, `#endif`, `#elif` výraz
  - integerové konstantní výrazy bez přetypování
  - počítají se v `(u)intmax_t`
  - navíc `defined(` ident `)`, `defined` ident
  - všude jinde se nahrazují makra
  - nedefinované identifikátory (i klíčová slova) → 0
- `#ifdef` ident, `#ifndef` ident
- `#define` ident`[(`arg`)]` expanze, `#undef` ident
  - pozor na mezery před `(arg)`
- `#include` <jméno> / "jméno" / makra
  - pozor, po expanzi maker se neslepují řetězce
  - ochrana proti vícenásobné inkluzi
- `#warning`, `#error`
- `#line` řádek `["soubor"]`
- `#pragma`, `_Pragma( ... )`



Nahrazování maker

1. Máme opravdu nahrazovat?
   - "funkcovité" makro nelze volat bez závorek
   - uvnitř direktiv se někdy nenahrazuje
2. Posbírají se argumenty (včetně vnořených závorek)
3. Argumenty se rekurzivně expandují
4. Volání makra se nahradí jeho definicí s argumenty
   - Argumenty operátorů `#` a `##` se nahrazují neexpandovaně
5. Provedou se preprocesorové operátory:
   - `#` stringifikuje
   - `##` slepuje tokeny (i prázdné)
6. Ve výsledné posloupnosti pp-tokenů se hledají další makra
7. Případně blokujeme rekurzi



Použití preprocesoru

Jednoduchá makra:

```
#define PLUS(a,b) ((a)+(b))
#define MIN(a,b) ((a)<(b) ? (a) : (b))
#define ARRAY_SIZE(a) (sizeof(a) / sizeof(*(a)))
```

Makro jako příkaz:

```
#define DBG(msg) do { \
 if (debug) puts(msg); } while(0)
```

Blokování rekurze je užitečné:

```
int f(int x);
#define f f
```

Makra s proměnlivým počtem argumentů:

```
#define F(...) do { \
 if (debug) printf(__VA_ARGS__) } while(0)
```

Temné kouty preprocesoru

Na tokenech občas záleží:

```
#define Ex +2
```

Pak `12Ex` není totéž jako `12␣Ex`

Pořadí vyhodnocování:

```
#define LEP(x,y) x ## y
#define LEP2(x,y) LEP(x,y)
#define A aa
LEP(a,1) →
```

Temné kouty preprocesoru

Na tokenech občas záleží:

```
#define Ex +2
```

Pak `12Ex` není totéž jako `12␣Ex`

Pořadí vyhodnocování:

```
#define LEP(x,y) x ## y
#define LEP2(x,y) LEP(x,y)
#define A aa
```

`LEP(a,1)` → `a1`

`LEP(A,1)` →

Temné kouty preprocesoru

Na tokenech občas záleží:

```
#define Ex +2
```

Pak `12Ex` není totéž jako `12␣Ex`

Pořadí vyhodnocování:

```
#define LEP(x,y) x ## y
#define LEP2(x,y) LEP(x,y)
#define A aa
```

`LEP(a,1)` → `a1`

`LEP(A,1)` → `A1`

`LEP2(A,1)` →

Temné kouty preprocesoru

Na tokenech občas záleží:

```
#define Ex +2
```

Pak `12Ex` není totéž jako `12␣Ex`

Pořadí vyhodnocování:

```
#define LEP(x,y) x ## y
#define LEP2(x,y) LEP(x,y)
#define A aa
```

`LEP(a,1)` → `a1`

`LEP(A,1)` → `A1`

`LEP2(A,1)` → `aa1`

Temné kouty preprocesoru

Na tokenech občas záleží:

```
#define Ex +2
```

Pak `12Ex` není totéž jako `12␣Ex`

WALK(LEFT);

Pořadí vyhodnocování:

```
#define LEP(x,y) x ## y
#define LEP2(x,y) LEP(x,y)
#define A aa
LEP(a,1) → a1
LEP(A,1) → A1
LEP2(A,1) → aa1
```

Parametry s čárkami:

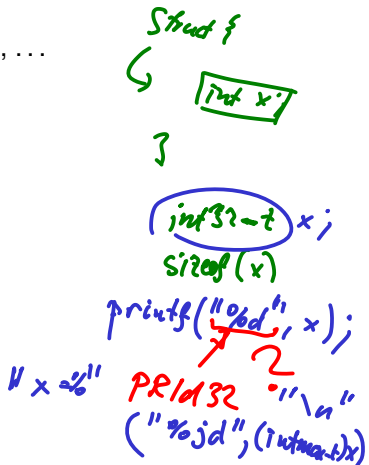
```
#define LEFT -1,0
#define WALK0(dx,dy) x+=dx, y+=dy
#define WALK(a) WALK0(a)
WALK(LEFT); → WALK0(-1,0); → x+=-1, y+=0;
```

Kreslíme obrázky preprocesorem

```
#define X )*2+1
#define _ )*2
#define s ((((((((((((((((0
static const uint16_t stopwatch[] = {

s _ _ _ _ _ X X X X X _ _ _ X X _ ,
s _ _ _ X X X X X X X X X _ X X X ,
s _ _ X X X _ _ _ _ _ X X X _ X X ,
s _ X X _ _ _ _ _ _ _ _ _ X X _ _ ,
s _ X X _ _ _ _ _ _ _ _ _ X X _ _ ,
s X X _ _ _ _ _ _ _ _ _ _ X X _ ,
s X X _ _ _ _ _ _ _ _ _ _ X X _ ,
s X X _ X X X X X _ _ _ _ _ X X _ ,
s X X _ _ _ _ _ X _ _ _ _ _ X X _ ,
s X X _ _ _ _ _ X _ _ _ _ _ X X _ ,
s _ X X _ _ _ _ X _ _ _ _ X X _ _ ,
s _ X X _ _ _ _ X _ _ _ _ X X _ _ ,
s _ _ X X X _ _ _ _ _ X X X _ _ _ ,
s _ _ _ X X X X X X X X X _ _ _ _ ,
s _ _ _ _ _ X X X X X _ _ _ _ _ _ ,
s _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

};
```

Bludiště knihovních typů

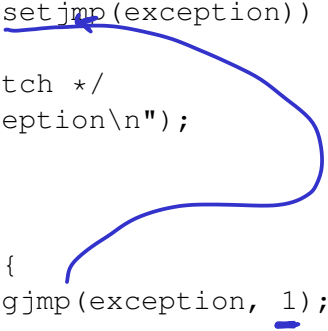
- limits.h:
  - `CHAR_BIT`
  - `CHAR_MAX`, `SHRT_MAX`, `INT_MAX`, `UINT_MAX`, `LONG_MAX`, ...
- float.h: podobně pro floatové typy
- stddef.h:
  - `ptrdiff_t`, `size_t`, `wchar_t`, `NULL`, `offsetof()`
- stdbool.h: `bool` (namísto `_Bool`)
- stdint.h:
  - `(u)int`*N*`_t`, `(u)int_least`*N*`_t`, `(u)int_fast`*N*`_t`
  - `(u)intptr_t`, `(u)intmax_t`
- inttypes.h:
  - `PRI`*f*`N`, `PRI`*f*`MAX`, `SCN`*f*`N`, ... (nebo `%jd`, `%zd`)
  - `strtoimax()`, `strtoumax()`
- signal.h: `sig_atomic_t`
- uchar.h: `char16_t`, `char32_t`



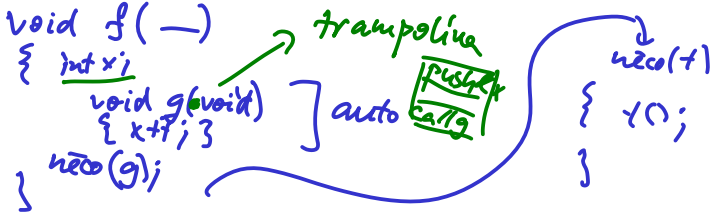
setjmp()

“Goto do dřívějšího bodu běhu programu”, např. k ošetření výjimek:

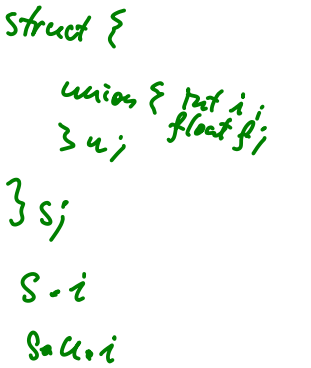
```
#include <setjmp.h>
jmp_buf exception;

int main(void) {
    /* Try */ if (!setjmp(exception)) {
        program();
    } else {  /* Catch */
        printf("exception\n");
    }
}

void program(void) {
    /* Throw */ longjmp(exception, 1);
}
```



GNU rozšíření: Syntaxe



- Vnořené funkce
- a `?:` b a?a:b
- `0b10001010`
- `case 1...5:`
- Anonymní prvky struktur (v C11 jsou)
- `__auto_type` *proměnná* = *hodnota* (GCC 4.9)



GNU rozšíření: Příkazové výrazy

```
#define MIN(a,b) ({ \
  typeof(a) _a=(a), _b=(b); _a < _b ? _a : _b; })

#define stk_strcat(a,b) ({ \
  char *_a=(a), *_b=(b); \
  int la=strlen(_a), lb=strlen(_b); \
  char *z = alloca(la + lb + 1); \
  memcpy(z, _a, la); \
  memcpy(z+la, _b, lb+1); \
  z; })
```

GNU rozšíření: Typy

- `__int128` – 128-bitový celočíselný typ (nemá literály)
- `__float128` – 128-bitový floatový typ (nemá literály)
- Fixed-point typy podle draftu N1169:
  - `_Fract` (rozsah $[-1, 1]$)
  - `_Accum` (alespoň 4 bity celé části)
  - `unsigned _Fract`, `short _Fract`, ...
  - modifikátor `_Sat` (saturace)
  - literály `12.3r`
  - printf: `%r` (`_Fract`), `%R` (`unsigned _Fract`), `%k` (`_Accum`)
  - `<stdfix.h>`
- Decimální floaty podle draftu N1312:
  - `_Decimal32`, `_Decimal64`, `_Decimal128`
  - literály `12.3d`, `12.3dd`, `12.3dl`
  - printf: `%Hf`, `%Df`, `%DDf`
  - `<float.h>`

GNU rozšíření: Inline assembler

```
static inline void cpuid(int op, int *a, int *b,
                                 int *c, int *d)
{
  asm("cpuid"
      : "=a" (*a), "=b" (*b), "=c" (*c), "=d" (*d)
      : "0" (op));
}

static inline uns bit_ffs(uns w)
{
  asm("bsfl %1,%0" : "=r" (w) : "rm" (w));
  return w;
}

#define wmb() asm volatile ("": : :"memory")
```

GNU rozšíření: Atributy funkcí, typů a proměnných

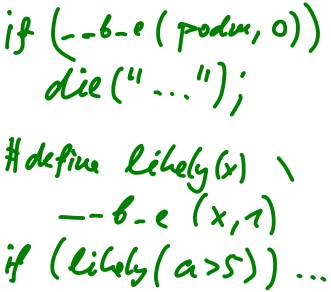
Příklad použití:

```
double sin __attribute__((const)) (double f);
```

Zajímavé atributy:

- Konstruktory: `constructor`
- Pro optimalizaci: `const`, `pure`, `malloc`, `hot`, `cold`, `optimize`, `noreturn` [`_Noreturn`]
- Inlinování: `noinline`, `always_inline`, `flatten`
- Proměnné a typy: `aligned`, `packed`
- Varování: `unused`, `format`, `warn_unused_result`, `warning`, `error`

GNU rozšíření: Zabudované funkce



- `__builtin_constant_p(value)`
- `__builtin_types_compatible_p(type1, type2)`
- `__builtin_choose_expr(condition, if-true, if-false)`
- `__builtin_expect(value, expectation)`
- `__builtin_unreachable()`
- `__builtin_prefetch(address, ...)`
- `__builtin_add_overflow(x, y, &result)`
- `__builtin_ffs(integer)`
- `__builtin_bswap(integer)`
- `__builtin_popcount(integer)` a další strojové instrukce

Viz též `_Generic(expr, type:value, ..., default:value)`.

Konec