Preprocesor

- Zpracovává ZK ještě před překladačem, výsledek = ZK v jazyce C
- Zpracování ZK preprocesorem = preprocesing
- Provádí prostou záměnu textů
- Příkaz preprocesoru = direktiva
  - začíná `#`
  - v ZK vždy na samostatné řádce
- Seznam direktiv

```
#define     #elif       #else       #endif
#error      #if         #ifdef      #ifndef
#include    #line       #pragma     #undef
```

- lze používat základní výrazy – výsledkem musí být konstatní výraz (hodnota známa v době překladu), součástí může být:
  - operandy pouze konstanty (všech typů) nebo pojmenované konstanty
  - závorky `()`
  - aritmetické operátory: `+ - * / %` (`+ -` unární i binární)
  - bitové operátory: `& | ^ ~`
  - logické operátory: `&& || !`
  - relační operátory: `== != < <= > >=`

Makra

```
#define vyraz hodnota
```

Bez parametrů

- příklady:

```
#define PI 3.14
#define ZNAKU 100
// priklad vyuziti operatoru v prepocesoru
#define DVEPI 2*PI
#define MAX_DELKA (ZNAKU+1)%5

Rad = PI*Stupne /180;   // po preprocesingu Rad = 3.14*Stupne /180
```

- makro v uzavřené v `""` se nerozvine:
  - chybně

```
#define CHYBA Stala se chyba
printf("CHYBA");  // printf("CHYBA");
```

  - správně

```
#define CHYBA "Stala se chyba"
printf (CHYBA);   // printf ("Stala se chyba");
```

- nejčastější chyby `#define`

```
#define WIDTH = 640
#define HEIGHT 480;

x = WIDTH;
y = HEIGHT * 8;
```

Po rozvinutí
`size_x == WIDTH;`

Po rozvinutí
`y = HEIGHT; * 8;`

- makro se nevejde na řádek

```
#include <stdio.h>

#define MAKRO     printf("Ahoj\n"); \
                  printf("jak se mas\n")

int main(void)
{
      printf("pred makrem\n");
      MAKRO;
      printf("po makru\n");
      return 0;
}
```

není ;

- pozor:

```
#define MAKRO printf("Ahoj\n"); \
#define MAKRO1 printf("jak se mas\n")
```

makro po \
preprocesor ignoruje

- mezery, tabelátory v makrech

```
#include <stdio.h>

#define MAKRO S MEZERAMI      printf("ahoj")
#define SOUCET          a + b

int main(void)
{
      int a = 5, b = 6, c;
      MAKRO S MEZERAMI;
      c = SOUCET;
      return 0;
}
```

OK

nelze

S parametry

- Krátká funkce (pár řádek ZK) → administrativa > „výkonný kód“
- syntaxe

```
#define MAKRO(par1, par2, … parN) rozvoj
```

- Příklad I

```
#define add(x,y) ((x)+(y))

return add(a + 3, b);
```

```
return ((a + 3)+(b));
```

- Závorkovat!!!

```
#define add(x,y) x + y

promenna = 5 * add(a, b);
```

```
promenna = 5 * x + y;
```

- špatně se hledají chyby!!!
- makra nelze ladit!!!

Předdefinovaná makra ANSI C

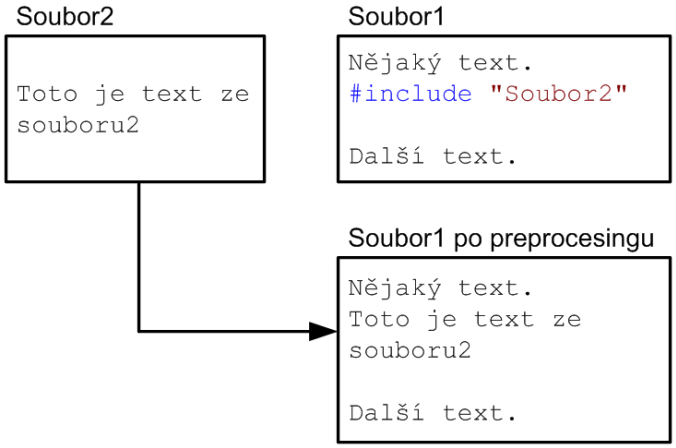
- __LINE__
  číslo právě zpracovávaného řádku programu (desítková celočíselná konstanta)
- __FILE__
  jméno právě překládaného souboru (řetězcová konstanta)
- __DATE__
  datum překladu souboru (řetězcová konstanta)
- __TIME__
  čas překladu (řetězcová konstanta)
- __STDC__
  Identifikace překladače ANSI C (ANSI C = 1, nebo nedefinována)
- příklad

```
#include <stdio.h>

int main(void)
{
      printf("Date: %s\n", __DATE__);
      printf("Time: %s\n", __TIME__);
      printf("File: %s\n", __FILE__);
      printf("Line No.: %d\n", __LINE__);
      printf("ANSI C? ");
#ifndef __STDC__
      printf("No");
#else
      printf("Yes");
#endif
      return 0;
}
```

Vkládání souborů

- `#include soubor`
- vloží text souboru `soubor` do souboru s direktivou



- `#include <soubor>`
  - `soubor` je hledán ve standardním adresáři pro `#include` (cestu zná překladač), pokud není nalezen = chybové hlášení
  - vkládání systémových hlavičkových souborů (`stdio.h`, `math.h` apod.)
- `#include "soubor"`
  - `soubor` je hledán ve aktuálním (pracovním) adresáři, nenalezen = postup jako u `<soubor>`
  - vkládání uživatelských souborů

Podmíněný překlad

- direktivy
  `#if`, `#else`, `#elif`, `#endif`, `#ifdef`, `#ifndef`
- = řízené vynechávání částí ZK
- využití při ladění

Dle konstantního výrazu

- syntaxe:

```
prikazy;
#if VYRAZ
      prikaz2;
#endif
prikazy;
```

```
prikazy;
#if VYRAZ
      prikazy;
#else
      prikazy;
#endif
prikazy;
```

```
prikazy;
#if VYRAZ
      prikazy;
#elif
      prikazy;
#elif
      prikazy;
#else
      prikazy;
#endif
prikazy;
```

`VYRAZ`: celočíselné konstanty, znakové konstanty, operátor `defined`

- příklad: Napište program s funkcí pro součet vektorů, vektory se budou zadávat z klávesnice. Program bude obsahovat ladící verzi, která umožní několik úrovní ladění:
  - bez zadávání údajů z klávesnice
  - se zadávání údajů z klávesnice

```c
#include <stdio.h>

#define DEBUG      0    // 0 = release version, 1 = debug version
#define KLAVESNICE 0    // 1 = pole se plni z klavesnice
#if DEBUG
	#define N 5         // at mam kratsi vypisy
#else
	#define N 200       // musim si udelat pole velke
#endif

void Secti(int PoleVysl[], int Pole1[], int Pole2[], int DelkaPole);

int main(void)
{
	// deklarace promennych
	int i;
	int SumaVektor[N];
	#if DEBUG
		int Vektor1[N] = {1,2,3,4,5};
		int Vektor2[N] = {1,2,3,4,5};
		int Delka = N;
	#else
		int Vektor1[N];
		int Vektor2[N];
		int Delka;
	#endif
	#if (DEBUG != 1)
		// nacteni dat z klavesnice
		printf("Zadej delku vektoru: ");
		scanf("%d", &Delka);
	#endif
	#if ((KLAVESNICE == 1) || (DEBUG != 1))
		for(i=0;i<Delka;i++) {
			scanf("%d", &Vektor1[i]);
			scanf("%d", &Vektor2[i]);
		}
	#endif
	Secti(SumaVektor, Vektor1, Vektor2, Delka);
	return 0;
}
void Secti(int PoleVysl[], int Pole1[], int Pole2[], int DelkaPole)
{
	int i;
	for(i=0;i<DelkaPole;i++) {
		PoleVysl[i] = Pole1[i] + Pole2[i];
	}
	#if DEBUG
		printf("Ladici vypis - sectene pole:\n");
		for(i=0;i<DelkaPole;i++) {
			printf("[%d] = %d\n", i, PoleVysl[i]);
		}
	#endif
}
```

Dle Existence konstanty

```
#include <stdio.h>

#define DEBUG           // debug version
// #define DEBUG        // release version
#define KLAVESNICE      // pole se plni z klavesnice
// #define KLAVESNICE   // pole jsou inic primo v definici

#ifdef DEBUG
      #define N 5       // at mam kratsi vypisy
#else
      #define N 200     // musim si udelat pole velke
#endif

void Secti(int PoleVysl[], int Pole1[], int Pole2[], int DelkaPole);

int main(void)
{
      int i;
      int SumaVektor[N];
      #ifdef DEBUG
            int Vektor1[N] = {1,2,3,4,5};
            int Vektor2[N] = {1,2,3,4,5};
            int Delka = N;
      #else
            int Vektor1[N]
            int Vektor2[N]
            int Delka;
      #endif
      #ifndef DEBUG
            // nacteni dat z klavesnice
            printf("Zadej delku vektoru: ");
            scanf("%d", &Delka);
      #endif
      #if defined(KLAVESNICE) || !defined(DEBUG)
            for(i=0;i<Delka;i++) {
                  scanf("%d", &Vektor1[i]);
                  scanf("%d", &Vektor2[i]);
            }
      #endif
      Secti(SumaVektor, Vektor1, Vektor2, Delka);
      return 0;
}
```

- lze využít parametr /D kompilátorů

#undef

- zrušení definice makra

```
#define add(x, y) ((x) + (y))

int main(void)
{
	int a = 5, b = 10, c;
	c = add(a,b);
	#undef add
	c = add(a,b);
	return 0;
}
```

chyba

#pragma

- pro přidání funkčnosti preprocesoru, předávání informací překladači
- silně implementačně závislé!!!

Příklady pro Microsoft C kompilátor

- změna stupně optimalizace na části kódu

```
#pragma optimize( "", off )
void Funkce(void)
{
	// telo
}
#pragma optimize( "", on )
```

- zpracování každého souboru překladačem pouze jednou

```
#pragma once
```

  - pro hlavičkové soubory, náhrada za

```
#ifndef SOUBOR_H
#define SOUBOR_H
// telo souboru
#endif
```

- Výpis informací do výstupu překladače

```
#include <stdio.h>

#ifndef F_CPU
	#pragma message("F_CPU musi byt definovano!!!")
#endif

int main(void)
{
	return 0;
}
```

#error

- zastavení překladu a výpis informace do výstupu překladače

```
#if defined(WIN32) && defined(LINUX)
      #error definovány obě platformy!
#endif
```

Operátor

- Převede atom na řetězec

```
#include <stdio.h>

#define test(x,y) printf(#x " a " #y)

int main(void)
{
      test(-3, 8);
      return 0;
}
```

```
printf("-3 a 8");
```