Velké projekty v C

- velký = „100ky“ a více řádek
- udržovatelnost, bezpečnost, přehlednost kódu → rozdělení programu do více souborů
- další prvky – globální proměnné, řízení viditelnosti proměnných, funkcí …

Globální proměnné

- opakování: lokální proměnné
  - definované uvnitř těla funkce (bloku)
  - viditelné pouze uvnitř těla funkce (bloku), ve kterém byly definovány
- Definice mimo tělo funkce
- Viditelné v jakékoliv funkci, jejíž definice se nachází **za** definicí globální proměnné

```
#include <stdio.h>

void TiskGlobal(void);

int GlobalProm;         // globalni promenna

int main(void)
{
      int LokalProm = 10;
      GlobalProm = 2*LokalProm;
      TiskGlobal();     // vytiskne: "Hodnota = 20"
      scanf("%d", &GlobalProm2);          // CHYBA!!!
      return 0;
}

int GlobalProm2;

void TiskGlobal(void)
{
      printf("Hodnota je %d\n", GlobalProm);
}
```

- implicitně inicializovány na 0 (dle typu – `0`, `0.0`, `'\0'`, `{0,0…}`, atd.)
- Používat s rozvahou!!
  - zhoršují udržovatelnost kódu (na PC prvotní hledisko) – funkce nejsou samostatné jednotky (viz. např. `TiskGlobal()` výše)
  - omezují paměťové nároky a mohou zvýšit rychlost aplikace (na PC podružné, u MCU někdy nutnost)
  - pouze pro sdílená data

- Příklad dobrého využití – aplikace pracující se sériovým portem (nefunkční!!!)

```
#include <stdio.h>

// globalni promenne
int Pristupu = 0;
int JePripojen = 0;     // 0 = nelze komunikovat, 1 = port pripraven

// deklarace funkci
void WriteCOM(int co);
int ReadCOM(void);
void InitCOM(void);
void CloseCOM(void);

int main(void)
{
      int temp;

      InitCOM();
      WriteCOM('A');
      WriteCOM('h');
      temp = ReadCOM();
      WriteCOM('o');
      temp = ReadCOM();
      WriteCOM('j');
      temp = ReadCOM();
      temp = ReadCOM();
      CloseCOM();

      printf("Pristupu: %d", Pristupu);
      return 0;
}
void InitCOM(void)
{
      // inicializace komunikace
      if (SePodarila) {
            JePripojen = 1;
      }
}
void CloseCOM(void)
{
      // uzavreni portu
      JePripojen = 0;
}
void WriteCOM(int co)
{
      if (JePripojen) { // ochrana pred komunikaci s neinit portem
            COM = co;   // COM je pouze abstrakce serioveho portu
            Pristupu++;
      }
}
int ReadCOM(void)
{
      if (JePripojen) { // ochrana pred komunikaci s neinit portem
            Pristupu++;
            return COM; // COM je pouze abstrakce serioveho portu
      }
      return 0;         // pokud nejsme pripojeni k portu
}
```

Viditelnost proměnných

- překrývání proměnných

```c
#include <stdio.h>

int x;      // 1

void f(void);

int main(void)
{
      int x;      // 2

      x = 10;     // do 2
      {
            int x;      // 3
            x = 20;     // do 3
            printf("x = %d\n", x);  // 3
      }
      printf("x = %d\n", x);  // 2
      f();
      return 0;
}

void f(void)
{
      x = 33;     // do 1
}
```

Atributy datových objektů

- datový objekt → proměnné, funkce
- datový objekt = identifikátor + atributy
- atribut = upřesnění vlastností datového objektu
  - explicitní → určuje programátor
  - implicitní → určuje překladač (programátor může změnit)
- atributy:
  - datový typ – známe
  - rozsah platnosti – scope
  - doba existence – duration
  - viditelnost z dalších modulů programu – linkage

Paměťové třídy

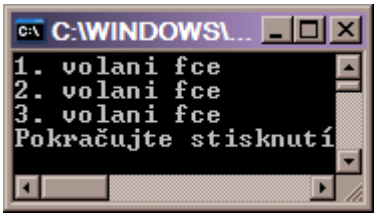
- upravují rozsah platnosti a dobu existence
- určuje umístění datového objektu v paměti

- třída `auto`
  - objekty uloženy na zásobníku
  - implicitně pro lokální proměnné
    - vzniká při vstupu – `{`
    - končí s `}`
    - příklad:

```
void Funkce(void)
{
      auto int lokProm; // auto není nutne
}
```

- třída `static`
  - mění duration lokálních proměnných
  - statická lokální proměnná:
    - vzniká při spuštění programu
    - zachovává hodnotu i po opuštění funkce
    - inicializována na `0`
  - příklad: počet volání funkce

```
#include <stdio.h>
void Funkce(void)
{
      static int pocet; // impl. init na 0
      printf("%d. volani fce\n", ++pocet);
}
int main(void)
{
      Funkce();
      Funkce();
      Funkce();
      return 0;
}
```



  - pozor na `static` u globálních proměnných
  - je uložena v datovém segmentu
- třída `register`
  - překladač umístí proměnnou do registru, nikoli do RAM → rychlejší
- třída `extern`
  - viz. později programy z více modulů

Typové modifikátory

- typový modifikátor `const`
  - definice tzv. konstantních proměnných = po inicializace nelze měnit

```
const int max = 100;
max = 200;  // error kompilátoru
```

  - omezení oproti `#define`:
    - nelze použít do konstantních výrazů (inicializace globálních proměnných, definice velikosti polí atd…)

- nejčastější použití – volání odkazem beze změny parametru uvnitř fce

```
int hledej(const char *str, char co)
```

- Příklad

```
#include <string.h>

// Funkce najde znak c a vrati pozici
int Hledej(const char *str, char c)
{
        // nejaky kod

      strcpy(str,"Cau");  // pokus o zmenu retezce
      /* kompilator vypise error:
      error C2664: 'strcpy' : cannot convert parameter 1 from 'const
      char *' to 'char *'
      */

        // nejaky kod

        return -1;
}
```

- typový modifikátor `volatile`
  - info pro kompilátor: proměnná může být modifikována pomocí nějaké asynchronní události – např. pomocí přerušení
  - úsek programu, ve kterém se tato proměnná nachází, nebude překladačem optimalizován.
  - příklad zápisu:

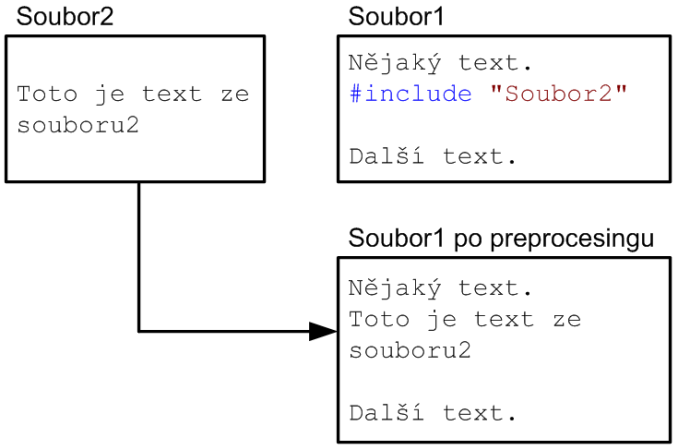
```
volatile int i;
i = 0;
while (i == 0);   /* smycka se ukonci az po prichodu nejake
vnejsi udalosti, ktera zmeni promennou i */
```

Projekt z více souborů

- možnosti:
  1. „include“ do jednoho → překlad
  2. oddělený překlad samostatných jednotek + linker

Ad 1. – include

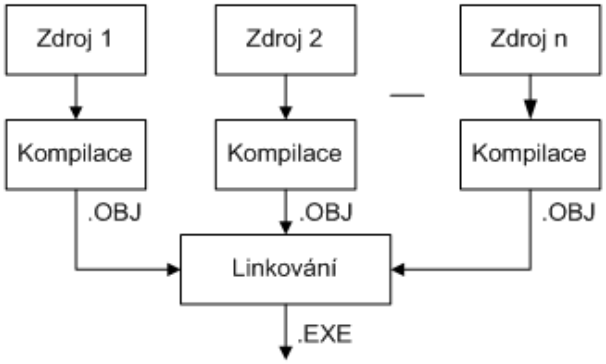
- opakování – `#include soubor`



- nepoužívat

Ad 2. – Samostatný překlad

- každý soubor *.c (*.cpp) → lze přeložit samostatně (*.obj)



- spolupráce modulů (na úrovni ZK) →export, import proměnných a funkcí (hlavičkové soubory)
- viditelnost identifikátorů mimo soubor se ZK → globální proměnné + označené funkce (viz. dále další podmínky)
- `extern` deklarace
  - = export import funkcí
  - import globálních proměnných
  - může sloužit i jako interní deklarace

- hlavičkové soubory = protředek pro sdílení informací mezi moduly
  - XXX.c: definice všech funkcí, neexportované – deklarace funkcí, `#define`, `typedef` ...
  - XXX.h: deklarace exportovaných funkcí, exportované – `#define`, `typedef` ...
  - Ostatní.c → pokud nutné něco z XXX.C → `#include "XXX.h"`
  - XXX.c doplnění deklarací exportovaných funkcí → taktéž `#include "XXX.h"`
  - nutno zabránit vícenásobným vložením (možné problémy)→ ochrana přes definici makra `JMENOSOUBORU_H` → viz. příklad níže.

- **Příklad 1**
  - předpoklady: 2 moduly + main()
    - modul 1 – interní f2(), exportovaná f1()
  - výpis funkce1.c

```
#include <stdio.h>      // nutne - funkce volaji printf()
#include "modul1.h"
#include "modul2.h"

void f2(void);          // deklarace f2 - nebude videt zvenku

int f1(void)            // definice f1
{
      int a;
      a = f3();         // volani importovane funkce
      f2();             // volani lokalni funkce
      printf("f1\n");
      return a;
}

void f2(void)           // definice f2
{
      printf("f2\n");
}
```

  - výpis funkce1.h

```
#ifndef MODUL1_H
#define MODUL1_H
// symbol plati až do konce souboru (kam byl h soubor vložen!)

extern int f1(void);    // export f1

#endif
```

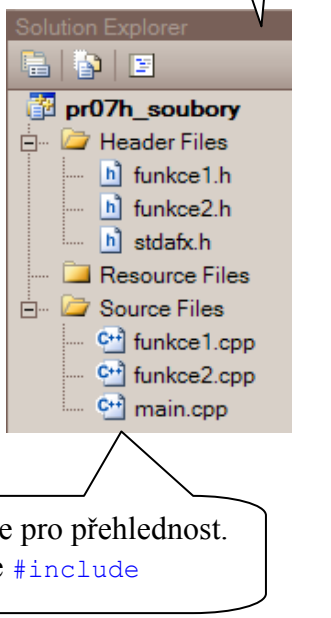
  - výpis funkce2.c

```
#include <stdio.h>      // volaji printf()
#include " modul2.h"

// deklarace
void f4(void);          // deklarace f4

// import
extern int gP1;         // deklarace gP1

int f3(void)            // definice f3
{
      f4();
      printf("f3\n");
      return gP1 + 10;
}
// lokalni pomocna funkce
void f4(void)           // definice f4
{
      printf("f4\n");
}
```



- výpis funkce2.h

```
#ifndef MODUL2_H
#define MODUL2_H

extern int f3(void);    // export f3

#endif
```

- výpis main.c

```
#include "modul1.h"
#include "modul2.h"

// definice GP
int gP1;    // možno dát jako extern int gP1 do dalšího h

int main(void)
{
      int a, b;
      gP1 = 5;
      a = f1();
      b = f3();
      return 0;
}
```

Static u globálních proměnných a funkcí

- velké projekty → možné kolize identifikátorů
- problém: lze importovat neexportované globální proměnná nebo funkce
- řešení: všechny lokální globální proměnné nebo funkce označit `static`
  - zcela něco jiného než u lokálních proměnných!!!
- **Příklad:**
  - výpis main.cpp

```
static int prom;        // chci lokalni glob. prom
static void f1(void);   // chci jako lokalni funkci


int main(int argc, char* argv[])
{
      return 0;
}

void f1(void)
{
      printf("f1()");
}
```

  - výpis funkce.cpp

```
extern int prom;        // ERROR: unresolved ext. symbol
extern void f1(void);   // ERROR: unresolved ext. symbol

void f2(void)
{
      prom = 10;
      f1();
}
```

- pokus o použití neexportovaného symbolu – chyba linkeru – neoznačí místo chyby!!!

```
funkce2.obj : error LNK2019: unresolved external symbol "int __cdecl f2(void)" (?f2@@YAHXZ)
referenced in function "void __cdecl f4(void)" (?f4@@YAXXZ)
        C:\dokumenty\ucim\iujce\přednášky\pr07\pr07pokusy\Debug\pr07pokusy.exe : fatal error
        LNK1120: 1 unresolved externals
```

Velké projekty – shrnutí

- globální proměnné – používat co nejméně
- projekt rozdělit do funkcí → spolu související = 1 soubor
    - lokální funkce a lokální globální proměnné v jednom souboru označit `static`
    - vše, co se exportuje → deklarace do hlavičkového souboru
    - *.h – zamezit vícenásobnému vložení
- pokud v X.c, potřebuji funkci z Y.c → `#include "X.h"`