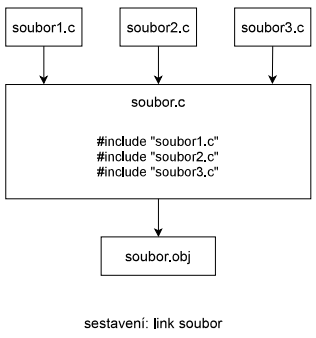
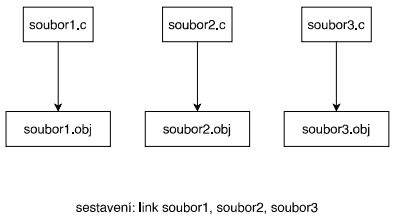
Dělení do souborů

V této kapitole se budeme věnovat dělení programu do více souborů a také oddělenému překladu. Rozdělíme-li program do více souborů, které pak pomocí `#include` vložíme do jednoho souboru, při překladu vznikne pouze jeden `.OBJ` soubor. *Odděleným překladem* souborů rozumíme přeložení každého souboru zvlášť (vznikne více `.OBJ` souborů) a jejich spojení pomoci linkeru do jednoho programu.
Představme si, že máme tři soubory `soubor1.c`, `soubor2.c` a `soubor3.c`. Funkce `main()` je v prvním uvedeném. Všechny soubory „inkludujeme“ do jednoho souboru `soubor.c`, který obsahuje tři příkazy `#include` a nic víc. Tento soubor se přeloží a vznikne jeden `.OBJ` soubor a ten se sestaví viz obrázek 7.
Nevýhodou tohoto přístupu je, že při změně v jednom ze souborů je potřeba přeložit i zbylé dva. Dále vzniká problém s globálními proměnnými, které mají stejný název, nebo které nechceme, aby byly viditelné v ostatních souborech.
Při odděleném překladu se každý soubor přeloží zvlášť, vzniknou tedy tři `.OBJ` soubory viz obrázek 8. Výhodou je, že při změně jednoho souboru, není potřeba znovu překládat všechny ostatní. Navíc při spolupráci více lidí na jednom projektu stačí sdílet jen `.OBJ` soubory. Tím se zabrání nechtěné modifikaci cizích kódů.
Prvním krokem při dělení kódu do více souborů je vždy rozmyslet dělení problému na menší a co nejméně závislé části. Dále je potřeba si rozmyslet, jakým způsobem budou jednotlivé části navzájem komunikovat (je nutné stanovit tzv. *interface*). To je možné buď skrze sdílené globální proměnné, nebo lépe skrze volání funkcí z jednotlivých modulů.

Obrázek 7: Vkládání souborů.



Obrázek 8: Oddělený překlad.



Rozšíření platnosti globálních proměnných

Při použití komunikace přes globální proměnné, je potřeba rozšířit jejich platnost ze souboru, kde byly definovány, do všech souborů pomocí klíčového slova `extern`.
Podívejme se na následující příklad. Vytvoříme dva zdrojové kódy. `pomocny.c` vypadá následovně.

```
int a;
extern int b;

int funkce1(){
    return a + b;
}
```

a `hlavni.c` obsahuje následující zdrojový kód.

```
#include <stdio.h>

extern int a;
extern int funkce1(); /* zde je nutne uvest cely funkcni
    prototyp */
int b;

int main(){
    a = 3;
    b = 3;
    printf("%d\n", funkce1());
    return 1;
}
```

Rozsah platnosti proměnných `a` a `b` je díky `extern` v obou souborech. Pro sestavení programu použijeme následující příkaz.[122]

```
gcc -o program hlavni.c pomocny.c
```

Po spuštění programu `program` se na výstup vypíše číslo 6.
Většina vývojových prostředí provádí oddělený překlad automaticky.[123]

[122] Protože jsou externí deklarace zpracovávány linkerem, je možné, že by mohl vzniknout problém při použití dlouhých identifikátorů. Je zvykem používat identifikátory sdílených proměnných a funkcí s maximální délkou 8 znaků.

[123] Oddělený překlad se provádí tak, že se vytvoří projekt, ve kterém se vytvoří jednotlivé soubory, a překládá se celý projekt.

Statické globální funkce a proměnné

Pokud nechceme, aby z jiných souborů byly viditelné některé globální proměnné či funkce, použijeme pro ně třídu `static`. Tyto proměnné jsou viditelné (platné) jen v souboru, ve kterém byly definovány, a to i když je použito v ostatních souborech klíčové slovo `extern`. Použití `static` globálních proměnných i funkcí je výhodné zejména, pokud na projektu pracuje více lidí, nemusí se pak obávat použití stejného identifikátoru.
Podívejme se opět na příklad. Máme následující dva soubory.
`pomocny.c` vypadá následovně.

```
static int a;
extern int b;
```

```
static int funkce1(int x){
    return x;
}

int funkce2(){
    return funkce1(b) + a;
}
```

`hlavni.c` obsahuje následující zdrojový kód.

```
#include <stdio.h>

extern int a;
extern int funkce2();
int b;

int funkce1(){
    printf("%d\n", funkce2());
}

int main(){
    a = 3;
    b = 3;
    funkce1();
    return 1;
}
```

Překlad proběhne úspěšně, při sestavování se ale objeví chyba, že není definovaná proměnná `a`. V souboru `pomocny.c` byla tato proměnná definovaná jako `static`, tudíž, ani když použijeme v `hlavni.c` `extern int a;`, tuto proměnnou v souboru nevidíme. Smažeme-li klíčové slovo `static`, překlad i sestavení proběhne bez problémů. Nevadí ani, že v obou souborech používáme stejný identifikátor pro funkci `funkce1()`, protože funkce je v souboru `pomocny.c` definovaná jako `static`, nemá tudíž s funkcí definovanou v `hlavni.c` nic společného.

Dělení programu na menší části

V této části uvedeme několik obecně platných pravidel, které je dobré dodržovat při vytváření programů, jež se skládají z více částí (modulů), a to zejména, pokud se na projektu podílí více lidí. Ke každému zdrojovému kódu (soubor s příponou `.c`) vytváříme tzv. *hlavičkový soubor* (soubor s příponou `.h`).
Definice funkcí a definice proměnných, vytváříme v souborech `.c` (například

`program1.c`). V tomto souboru striktně rozlišujeme, které proměnné a funkce budeme dávat k dispozici vně souboru. Snažíme se však, aby jich bylo málo. Spíše vytváříme speciální funkce, které zajišťují manipulaci s nimi.[124] Všechny ostatní funkce a globální proměnné označíme, jako `static`. Funkční prototypy všech funkcí a definice globálních proměnných zkopírujeme do hlavičkového souboru, který nese stejné jméno, jako zdrojový soubor (`program1.h`), a označíme je zde `extern`. Pokud z modulu poskytujeme nějaké symbolické konstanty, definují se v hlavičkovém souboru. Na začátku souboru `.c` vložíme hlavičkový soubor (`#include ''program1.h''`). Tím jsme zaručili, že ve zdrojovém kódu známe funkční prototypy všech funkcí i všechny symbolické konstanty. Deklarace globálních proměnných diky klíčovému slovu `extern` také nezpůsobí žádný problém. Do dalších modulů (například `program2.c`) vkládáme pouze hlavičkové soubory ostatních modulů (`#include ''program1.h''`).

[124] Tomuto přístupu říkáme *autorizovaný přístup*.

Doporučený obsah `.c` souboru

Soubor `.c` by měl obsahovat dokumentační část (jméno souboru a verze, stručný popis modulu, jméno autora a případně datum). Hlavička souboru by mohla vypadat následovně:

```
/*
 * obd_main.c   verze 1.0
 *
 * Program pro pocitani s obdelniky
 * ================================
 *
 * M. Trneckova, zari 2021
 *
 */
```

Poté by mělo následovat vložení všech potřebných hlavičkových souborů. Nejprve se vkládají systémové hlavičkové soubory příkazem `#include < >` a pak vlastní `#include '' ''`. Následují deklarace externích proměnných a funkcí (pokud nejsou v hlavičkovém souboru). Dále se definují globální proměnné, které se budou sdílet s ostatními moduly (nesmíme zapomenout přidat jejich definici označenou `extern` do hlavičkového souboru). Následuje definice symbolických konstant a maker s parametry, které se používají pouze v tomto modulu, definice lokálních typů (`typedef`), definice statických globálních proměnných, funkční prototypy lokálních funkcí (nejlépe úplné). Poté se definuje funkce `main()`, pokud má být součástí modulu, následovaná definicí ostatních funkcí. Nejprve se uvádí globální funkce, tedy funkce, které mohou být volány z ostatních modulů, a poté lokální, které mají modifikátor `static`.

Doporučený obsah hlavičkových souborů `.h`

Pro hlavičkové soubory platí podobná pravidla, jako pro soubory `.c`. Ke každému souboru `.c`, s výjimkou souboru, ve kterém je uvedena funkce `main()`,[125] by měl být vytvořen hlavičkový soubor. Hlavičkové soubory nesmí obsahovat žádné definice a inicializace proměnných. Neměly by obsahovat žádné příkazy `#include` kromě vkládání standardních knihoven. Dále by měly zajišťovat ochranu proti opakovanému vkládání souborů (pomocí podmíněného překladu).
Soubory by měly obsahovat dokumentační část (stejně jako u odpovídajícího `.c` souboru), podmíněný překlad proti opakovanému vkládání, definice symbolických konstant a maker s parametry (o kterých se předpokládá, že se budou používat i v jiných modulech), definice globálních typů pomocí `typedef`. Dále obsahuje deklarace globálních proměnných příslušného modulu a úplné funkční prototypy globálních funkcí (obojí označené jako `extern`).

[125] Tento soubor je pro všechny soubory nadřazený. Nemusí tedy ostatním modulům předávat žádné informace.

Konkrétní příklad

V této části s uvedeme konkrétní (ale ne kompletní) příklad, jak by mohlo vypadat členění programu do více souborů. Program bude počítat obvod a obsah obdélníku. Bude se skládat ze dvou modulů, modul `obd_funkce` (pro modul vytvoříme soubory `.h` a `.c`) a hlavní modul `obd_main`.
Začneme modulem obsahující funkce pro výpočet obvodu a obsahu obdélníku. Zdrojový soubor (`obd_funkce.c`) může vypadat následovně.

```
/*
 * obd_funkce.c   verze 1.0
 *
 * Funkce pro vypocet obsahu a obvodu
 * ==================================
 *
 * M. Trneckova, zari 2021
 *
 */

 /* systemove hlavickove soubory */
 /* zadne nejsou */

 /* vlastni hlavickove soubory */
 #include "obd_funkce.h" /* hlavickovy soubor vlastniho
     modulu */
 /* hlavickove soubory ostatnich modulu - zadne nejsou */

 /* def. globalnich promennych */
```

```
 /* zadne nejsou */

 /* lokalni def. symbolickych konstant a maker */
 /* zadne nejsou */

 /* lokalni def. novych typu */
 /* zadne nejsou */

 /* def. statickych globalnich promennych */
 /* zadne nejsou */

 /* uplne funkcni prototypy lokalnich funkci */
 /* zadne nejsou */

 /* funkce main */
 /* v tomto modulu neni */

 /* def. lokalnich funkci */
 /* zadne nejsou */

 /* def. globalnich funkci */
 double obvod(double a, double b){
    return (2 * (a + b));
 }

 double obsah(double a, double b){
    return (a * b);
 }
```

Příslušný hlavičkový soubor obsahuje následující kód.

```
/*
 * obd_funkce.h   verze 1.0
 *
 * Hlavickovy soubor pro modul obd_funkce.c
 * ========================================
 *
 * M. Trneckova, zari 2021
 *
 */

 /* podmineny preklad proti nasobnemu vkladani */
 #ifndef OBD_FUN
 #define OBD_FUN

 /* def. symbolickych konstant pouzitych v jinych modulech
     */
 /* zadne nejsou */
```

```
/* def. maker s parametry */
/* zadne nejsou */

/* def. globalnich typu */
/* zadne nejsou */

/* def. globalnich promennych modulu obd_funkce.c */
/* zadne nejsou */

/* uplne funkcni prototypy globalnich funkci modulu
    obd_funkce.c */
extern double obvod(double a, double b);
extern double obsah(double a, double b);

#endif /* OBD_FUN */
```

Hlavní zdrojový kód je následující.

```
/*
 * obd_main.c   verze 1.0
 *
 * Program pro praci s obdelniky
 * =================================
 *
 * M. Trneckova, zari 2021
 *
 */

/* systemove hlavickove soubory */
#include <stdio.h>

/* vlastni hlavickove soubory */
/* hlavickovy soubor vlastniho modulu - neni */
#include "obd_funkce.h" /* hlavickove soubory ostatnich
    modulu */

/* def. globalnich promennych */
/* zadne nejsou */

/* lokalni def. symbolickych konstant a maker */
#define CHYBA -1.0
#define kontrola(delka) (((delka) >= 0.0) ? (delka) : CHYBA
    )

/* lokalni def. novych typu */
/* zadne nejsou */

/* def. statickych globalnich promennych */
static double a;
```

```
static double b;

/* uplne funkcni prototypy lokalnich funkci */
static double nacti();
static void vypis(double o, double s);

/* funkce main */
int main(){
   double o, s;
   a = nacti();
   b = nacti();

   if(a == CHYBA || b == CHYBA){
       /* Vypis chybove hlasky */
       return 1;
   }

   o = obvod(a, b);
   s = obsah(a, b);

   vypis(o, s);
   return 0;
}

/* def. lokalnich funkci */
static double nacti(){
   double vstup;
   /* nacteni strany od uzivatele */
   scanf("%lf", &vstup);
   return kontrola(vstup);
}

static void vypis(double o, double s){
   printf("\nObvod = %f, obsah = %f", o, s);
}

/* def. globalnich funkci */
/* zadne nejsou */
```

Kontrola vstupních dat by správně měla být už ve funkci, která data čte, stejně tak by funkce měla obsahovat možnost vstupní data opravit.

Cvičení

Úkol 80
Vytvořte dva soubory `soubor1.c` a `soubor2.c`. Oba budou sdílet proměnnou `a`. Proměnná je definovaná v `soubor1.c` a v `soubor2.c` se vypisuje.

Úkol 81
Změňte předchozí program tak, aby proměnná `a` byla definovaná jako globální `static`. Vyzkoušejte chování programu.

Úkol 82
Vytvořte program `funkce.c`, který bude obsahovat jen funkce s funkčními prototypy `float plus(float a, float b);` a `float minus(float a, float b);` počítající součet (rozdíl) dvou čísel. Dále vytvořte program `main.c`, který bude obsahovat pouze funkci `main()`, ve které budou volány funkce `plus()` a `minus()`.
Pomocí `#include` vložte soubor `funkce.c` do souboru `main.c` a přeložte.

Úkol 83
Vyzkoušejte funkci programu z předchozího cvičení, pokud nebude `funkce.c` do souboru `main.c` vkládaná, ale budou překládány samostatně.

1. Soubory odděleně překládejte a v souboru `main.c` uveďte úplné funkční prototypy funkcí `plus()` a `minus()`.
2. Vytvořte soubor `funkce.h`, do něhož vložte úplný funkční prototypy funkcí `plus()` a `minus()`. Pomocí `#include` zajistěte spojení souborů `funkce.c` a souboru `main.c`.