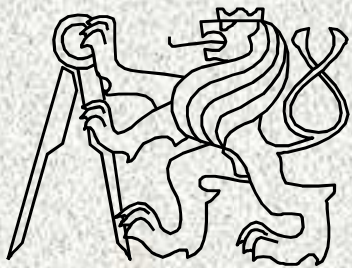


Jazyk C - struktury



A0B36PR2-Programování 2

Fakulta elektrotechnická

České vysoké učení technické

C - Obsah

Přehled

- C - proměnné, struktura (struct)
- C - proměnné, sdílená paměť (union)
- C – funkce – parametry main()
- C - příkazy preprocesoru
- C - standardní knihovny

Obsah

- Porovnání JAVA vs „C“
 - Globální proměnné
 - Hlavičkové soubory
 - Struktura velkých programů
 - Struktura
 - Union
 - Struktura v unionu
 - Ukazatel na funkce
 - Pole ukazatelů na funkce
 - Podmíněný překlad

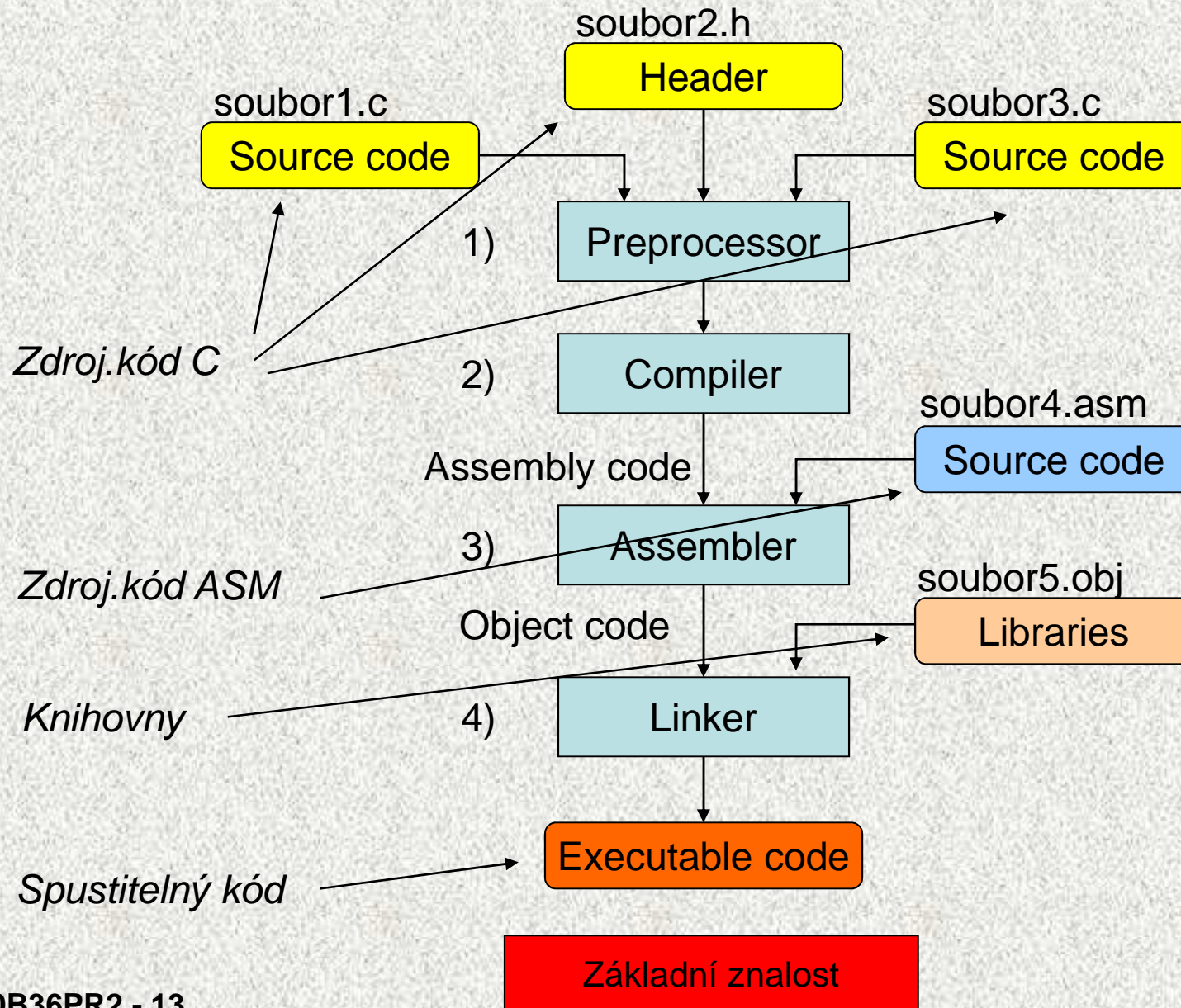
Globální proměnné - `extern`

- Proměnné platné ve více modulech téhož programu
- `extern` = je definováno jinde, externě

```
// hlavní.c
#include <stdio.h>
extern int x; /* deklarace*/
extern int fun();
        /* funkční prototyp*/
double f; /* definice*/
int main(void){
x = 3;
f = 3.5;
printf("%d\n", fun());
return (EXIT_SUCCESS);
}
```

```
// pomocny.c
int x; /* definice*/
extern double f;
        /* deklarace*/
int fun(void){
return (x + (int) f);
}
```

C - Model kompilace



Hlavičkové soubory

- Vkládají se pomocí příkazy `#include`, přípona `.h`
`#include "konstanty.h"` – uživatelské soubory
`#include <stdio.h>` - standardní hlavičkové soubory
- Mechanismus pro udržení čitelnosti rozsáhlých programů (chudá „analogie“ rozhraní z Javy)
 - Uložení definice konstant
 - Uložení prototypů funkcí
 - Uložení prototypů datových typů
 - Jedno místo definice, zabezpečení duplicit

Soubor `konstanty.h`

```
#ifndef TERMINAL_H
```

```
#define TERMINAL_H
```

```
#define POCET_RADEK 60
```

```
#define POCET_SLOUPCU 30
```

```
...
```

```
#endif
```

Hlavičkové soubory, vlastní

- Důvody:
 - Deklarace funkčního prototypy před použitím
 - Prostředek pro zpřehlednění struktury programu
 - Ukrytí definice funkce, možnost vytváření knihoven
 - Předání souboru **.h + .obj**
 - Vlastní definice funkce v souborech s relativním kódem **.obj**
- Obsahují
 - Hlavičky funkcí (funkční prototypy) – deklarace "funkce"
 - Deklarace globálních proměnných
 - Definice datových typů
 - Definice symbolických konstant
 - Definice maker
- „obdoba interface“

Struktura hlavičkového souboru `xxx.h`

```
/* podmíněny preklad proti opakovanému inkludování */
#ifndef XXX
#define XXX
/* definice symb. konstant vyuzivanych i v jiných modulech */
#define CHYBA -1.0
/* definice maker s parametry */
#define je_velke(c) ((c) >= 'A' && (c) <= 'Z')
/* definice globalních typu */
typedef struct{
    int vyska;
    int vaha;
} MIRY;
/* deklarace globalních promenných modulu xxx.c */
extern MIRY m; // v jiném modulu bude definice MIRY m;

/* úplné funkční prototypy globalních funkcí modulu xxx.c */
extern double vstup_dat(void);
extern void vystup_dat(double obsah);

#endif
```


Struktura souboru `xxx.c`

```
#include <stdio.h> /* standardni vklad*/
#include „xxx.h“    /* natazeni konstant, prototypu funkci
                   a globalnich typu vlastniho modulu */
/* definice globalnich promennych */
extern int z; /* kdyby sloužily pro jiné moduly */
/* lokální definice symbolických konstant a maker */
#define kontrola(x) ((x) >= 0.0) ? (x) : CHYBA_DAT )
/* lokální definice nových typu */
typedef struct{} OSOBA;
/* definice statických globalních promenných */
static MIRY m;
/* úplně funkční prototypy lokálních funkcí */
int nextDouble(double *cislo);
/* funkce main() */
int main(int argc, char** argv){
/* definice globalních funkcí */
double vstup_dat(void){ ...return ();}
void vystup_dat(double obsah){ ... }
/* definice lokálních funkcí */
int nextDouble(double *cislo){... }
```

C - Struktura (struct)

C - Struktura (struct):

- Struktura je *množina prvků* (proměnných), které *nemusí* být *stejného typu*
- *Skladba* struktury je *definovaná uživatelem* jako *nový typ* sestavený z již definovaných typů
- K prvkům struktury *se přistupuje tečkovou notací*
- K prvkům struktury je *možné přistupovat i pomocí ukazatele* na strukturu operátorem *->*
- Struktury *mohou být vnořené*
- Pro struktury *stejného typu* je definována operace *přiřazení* *struct1=struct2* (pro proměnné typu pole přímé přiřazení není definováno, jen po prvcích)
- Struktury (jako celek) *nelze porovnávat* relačním operátorem *==*
- Struktura *může být do funkce předávána hodnotou i odkazem*
- Struktura *může být návratovou hodnotou funkce*

C - Struktura (struct)

C - Struktura (struct) - sablona (tag):

```
př: struct Tid{ // <==== Tid=jmeno sablony (tag)
    char jmeno[20]; // Prvky struktury, pole
    char adresa[50]; // - " - pole
    long int telefon; // - " - int
};

struct Tid sk1,skAvt[20]; // struktura, pole struktur
struct Tid *pid; // ukazatel na strukturu
sk1.jmeno="Jan Novak"; // teckova notace
sk1.telefon=123456;
skAvt[0].jmeno="Jan Novak"; // prvek pole
skAvt[3].telefon=123456;
pid=&sk1; // do pid adresa struktury
pid->jmeno="Jan Novak"; // odkaz pomoci ->
pid->telefon=123456;
(*pid).jmeno="Jan Novak"; // odkaz pomoci *
(*pid).telefon=123456;
```

C - Struktura (struct)

C - Struktura (struct) - nový typ (typedef):

```
př: typedef struct {           // <==== Pomoci Typedef
    char jmeno[20];           // Prvky struktury, pole
    char adresa[50];        // - " -           pole
    long int telefon;       // - " -           int
    }Tid,*Tidp;

Tid sk1,skAvt[20];         // struktura, pole struktur
Tidp pid;                 // ukazatel na strukturu
sk1.jmeno="Jan Novak";    // teckova notace
sk1.telefon=123456;
skAvt[0].jmeno="Jan Novak"; // prvek pole
skAvt[3].telefon=123456;
pid=&sk1;                 // do pid adresa struktury
pid->jmeno="Jan Novak";    // odkaz pomoci ->
pid->telefon=123456;
(*pid).jmeno="Jan Novak"; // odkaz pomoci *
(*pid).telefon=123456;
```


C - Struktura (struct)

C - Struktura (struct) - inicializace:

př:

```
struct Tid{           // <==== Tid=jmeno sablony (tag)
    char jmeno[20];    // Prvky struktury, pole
    char adresa[50];  // - " -           pole
    long int telefon; // - " -           int
};
```

```
struct Tid sk1={"Jan Novak",
               "Na Kopecku 23",
               123456};
```

C – struktura – kvadraticka rovnice /jiné/ (10)

```
#include <stdio.h>
#include <stdlib.h>
typedef struct{
    double re1; double im1; double re2; double im2;
}Tcomplex;
int kvadratickaRovnice (double a, double b, double c, Tcomplex
    *koreny);
void vypisKorenuKvadratRovnice (char jmenoPromenne[], Tcomplex s);
void vypisKomplexCisla (double re, double im);
int nextDouble(double *cislo);
int main(int argc, char** argv) {
    Tcomplex s; double a,b,c;
    printf(" Zadejte koeficienty a, b, c \n\n");
    if(!nextDouble(&a) || !nextDouble(&b) || !nextDouble(&c)){
        printf("\n Chyba - zadany udaj neni cislo\n\n");
        return(EXIT_FAILURE);
    }printf("\n");
    if(!kvadratickaRovnice(a,b,c,&s)){
        printf(" Neplatne zadani (a=0)\n\n");
    }else{
        vypisKorenuKvadratRovnice("Koren x",s);
    }
    return (EXIT_SUCCESS);
}
```

C – struktura – kvadraticka rovnice /jiné/ (10)

```
int kvadratickaRovnice(double a, double b, double c, Tcomplex *koreny)
{
// Navratova hodnota: TRUE - vypocet platny, FALSE - neplatny vstup
    const int TRUE = 1;
    const int FALSE = 0;
    double disk = b * b - 4 * a * c;
    double odmocDisk;
    if(a == 0)
        return(FALSE);
    if (disk >= 0) {
        odmocDisk = sqrt(disk);
        koreny->re1 = (-b + odmocDisk) / (2 * a);
        koreny->im1 = 0;
        koreny->re2 = (-b - odmocDisk) / (2 * a);
        koreny->im2 = 0;
    } else {
        odmocDisk = sqrt(-disk);
        koreny->re1 = -b / (2 * a);
        koreny->im1 = odmocDisk / (2 * a);
        koreny->re2 = koreny->re1;
        koreny->im2 = -odmocDisk / (2 * a);
    }
    return(TRUE);
}
```

5

6

C – struktura – kvadraticka rovnice /jiné/ (10)

```
void vypisKorenuKvadratRovni(char jmenoPromenne[], Tcomplex s){  
    printf(" %s1 = ", jmenoPromenne);  
    vypisKomplexCisla(s.re1, s.im1);  
    printf("\n\n");  
    printf(" %s2 = ", jmenoPromenne);  
    vypisKomplexCisla(s.re2, s.im2);  
    printf("\n\n");  
}
```

Diagram illustrating function calls:

- Call site 7 (yellow circle) points to the function definition `vypisKomplexCisla(s.re1, s.im1);`.
- Call site 8 (yellow circle) points to the function call `vypisKomplexCisla(s.re2, s.im2);`.

```
void vypisKomplexCisla (double re, double im){  
    if(im >= 0){  
        printf("%.2f + j%.2f", re, im);  
    }else{  
        printf("%.2f - j%.2f", re, -im);  
    }  
}
```


C – struktura – kvadraticka rovnice /jiné/ (10)

```
int nextDouble(double *cislo){
    // Stav: odladeno
    // === Bezpecne pro libovolny zadany pocet znaku ===
    // Navratova hodnota:
    // TRUE - zadano realne cislo
    // FALSE - neplatny vstup
    enum boolean {FALSE,TRUE};
    const int BUF_SIZE = 80;
    char vstup[BUF_SIZE],smeti[BUF_SIZE];
    fgets(vstup,sizeof(vstup),stdin);
    if(sscanf(vstup,"%lf%[^\n]",cislo,smeti) != 1)
        return(FALSE); // Input error
    return(TRUE);
}
```

C – union a struktura v unionu /jiné/ (11)

```
// Union, struktura v unionu (anonymni a pojmenovana)
// Union - prekryta pamet promennych
// vyhradí se prostor pro nejdelší
typedef unsigned char byte;
union {
    int a;
    int b;
}uab;
```

1

```
// Anonymni struktura v unionu
// Pristup: jmeno_unionu.jmeno_prom, napr. u1.a
union {
    struct{
        int a;
        int b;
    };
    struct{
        byte b1;
        byte b2;
        byte b3;
        byte b4;
    };
};
```

2

C – union a struktura v unionu /jiné/ (11)

```
typedef struct{  
    int x1;  
    int x2;  
}Tintx12;
```

```
typedef struct{  
    byte bx1;  
    byte bx2;  
    byte bx3;  
    byte bx4;  
    byte by1;  
    byte by2;  
    byte by3;  
    byte by4;  
}Tbytexy14;
```

3



C - union a struktura v unionu /jiné/ (11)

```
// Pojmenovaná struktura v unionu
```

```
// Prístup: jmeno_unionu.jmeno_struct.jmeno_promenne, napr.
```

```
u1.intx.x1
```

```
union{  
    Tintx12 intx;  
    Tbytexy14 bytex;
```

```
}u2;
```

```
union{  
    int *px1;  
    int *px2;  
    int *px3;
```

```
}u3;
```

```
typedef union{
```

```
    int x1;
```

```
    int x2;
```

```
    int x3;
```

```
}Tu4;
```

```
Tu4 u4, *pu4;
```

4



C - union a struktura v unionu /jiné/ (11)

```
int main(int argc, char** argv) {
    int x, y;
    byte b;
    printf("=== C10a.c ===\n\n");
    printf(" union, structure \n\n");

    printf(" == union ab == \n\n");
    uab.a = 1;
    printf(" a = %d \n\n", uab.a);
    printf(" b = %d \n\n\n", uab.b);

    printf(" == union u1 == \n\n");
    u1.a = 0x12345678;
    b = u1.b1;
    printf(" x1 = %xh \n\n", u1.a);
    printf(" b4,3,2,1 = %xh %xh %xh %xh \n\n\n", u1.b4,
    u1.b3, u1.b2, u1.b1);

    // Pokrac.na dalsi strane
```

5

6

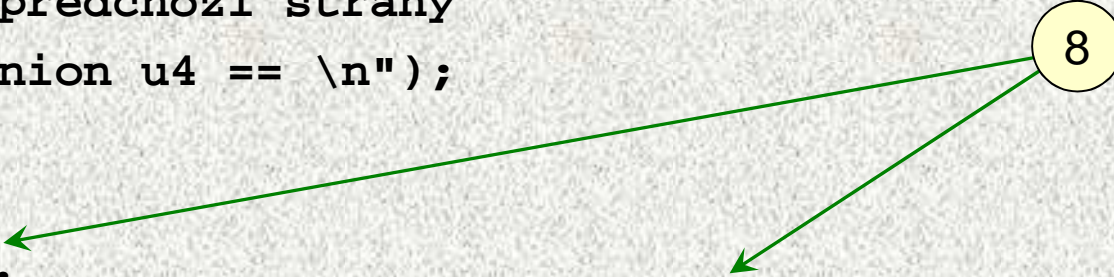
C - union a struktura v unionu /jiné/ (11)

```
// Pokrac. z predchozi strany
printf(" == union u2 == \n\n");
u2.intx.x1 = 0x87654321;
u2.intx.x2 = 0x00AB00CD;
printf(" x1 = %08Xh \n\n", u2.intx.x1);
printf(" bx4,3,2,1 = %Xh %Xh %Xh %Xh \n\n",
       u2.bytex.bx4, u2.bytex.bx3, u2.bytex.bx2, u2.bytex.bx1);
printf(" x2 = %08Xh \n\n", u2.intx.x2);
printf(" by4,3,2,1 = %Xh %Xh %Xh %Xh \n\n\n",
       u2.bytex.by4, u2.bytex.by3, u2.bytex.by2, u2.bytex.by1);
printf(" == union u3 == \n\n");
u3.px1 = &x;
*u3.px1 = 10;
printf(" x = %d, *px1 = %d, *px2 = %d, px3 = %d \n\n\n",
       x, *u3.px1, *u3.px2, *u3.px3);
// Pokrac. na dalsi strane
```

C - union a struktura v unionu /jiné/ (11)

```
// Pokrac. z predchozi strany
printf(" == union u4 == \n");
pu4 = &u4;
u4.x1 = 33;
pu4->x3 = 55;
printf("\n u4.x1 = %d, (*pu4).x2 = %d, pu4->x3 = %d
\n\n\n", u4.x1, (*pu4).x2, pu4->x3);

printf(" Konec programu \n\n");
return (EXIT_SUCCESS);
} // main() END
```



C – podmíněný překlad, ukazatel na funkce

/jiné/ (12)

```
// Programovací styly - Citac /proceduralni styl - reseni 4
// struktura, ukazatel na funkci, pole ukazatelu na funkci
// Podmineny preklad
#define VERSE_CITACE 3 // Platne: 1,2,3
int konec (void); // Function prototypes
int zvetsi (void);
int zmensi (void);
int nastav (void);
int hodnota (void);

struct {
    int hodnota ;
    int (*operace[5])(void); // array of function pointers
} citac = {0,&konec,&zvetsi,&zmensi,&nastav,&hodnota};

typedef struct {
    int value ;
    int (*operation[5])(void); // array of function pointers
}Tcounter;

Tcounter counter =
    {0,&konec,&zvetsi,&zmensi,&nastav,&hodnota};
```

1

Pro zájemce

C – podmíněný překlad, ukazatel na funkce

/jiné/ (12)

```
int main(int argc, char** argv) {
    enum boolean {FALSE,TRUE}; int volba, hodnota;
    #if VERSE_CITACE == 1
    do {
        // == Rozfazovany zapis cinnosti ==
        volba = menu();
        hodnota = (*citac.operace[volba})();
        printf("\n Hodnota = %d \n\n", hodnota);
    } while(TRUE);
    #endif
    #if VERSE_CITACE == 2
    do {
        // == Kompaktni zapis cinnosti ==
        printf("\n Hodnota = %d \n\n",(*citac.operace[menu()]));
    } while(TRUE);
    #endif
    #if VERSE_CITACE == 3
    do {
        // == Kompaktni zapis cinnosti == pomoci typedef Tconter
        printf("\n Hodnota = %d \n\n",(*counter.operation[menu()]));
    } while(TRUE);
    #endif
    return (EXIT_SUCCESS);
}
```

2

C – podmíněný překlad, ukazatel na funkce /jiné/ (12)

```
int zvetsi (void){  
    citac.hodnota++;  
    return(citac.hodnota);  
}
```

```
int zmensi (void){  
    citac.hodnota--;  
    return(citac.hodnota);  
}
```

```
int nastav (void){  
    citac.hodnota=0;  
    return(citac.hodnota);  
}
```

```
int hodnota (void){  
    return(citac.hodnota);  
}
```

C – podmíněný překlad, ukazatel na funkce /jiné/ (12)

```
int menu (void) {
    enum boolean {FALSE,TRUE};
    int volba;
    do {
        printf(" 0. Konec \n");
        printf(" 1. Zvetsi \n");
        printf(" 2. Zmensi \n");
        printf(" 3. Nastav \n");
        printf(" 4. Hodnota \n");
        printf("\n Vase volba: ");
        if(!nextInt(&volba) || volba < 0 || volba > 4){
            printf("\n Nepovolena volba \n\n");
        }else{
            return(volba);
        }
    } while (TRUE);
}
```


C - proměnné, deklarace, definice

C - Deklarace vs. Definice:

- *Deklarace* určuje interpretaci a vlastnosti identifikátoru(ů)
- *Definice* je *deklarace včetně přidělení paměti* (memory allocation) proměnným, konstantám nebo funkcím

C - Syntaxe deklarace:

- *[specifikator_pametove_tridy] typ D1 [,D2,...];*
kde:
 - *specifikator_pametove_tridy* - *extern, static, auto, register*
 - *typ* - *primitivní typy* (char,int,...), *void, enum, struct, union, typedef name*
(+ *const, volatile, short, long, unsigned*)
 - *D1 [,D2,...]* - seznam deklarátorů
- jednoduchý deklarátor:
identifikátor + počáteční_hodnota (nepovinná)
- složený deklarátor
identifikátor + počáteční_hodnota+symboly_, [,], ()*(pointer, array, function)

C - proměnné, deklaráce, definice

C - Příklady deklaráce proměnných:

- jednoduché deklaráce

```
char znak;  
int i,j,k;  
unsigned int suma=0;           // +inicializace  
const int k1=250;             // +konstanta  
static double rychlost,zrychleni;  
extern unsigned char status;
```

- složené deklaráce

```
int pole[30];                  // jednorozm.pole  
unsigned int poleA[]={10,20,30,40}; // +inicializace  
char s1[]="Ahoj";              // +inicializace  
char s2[5]={'A','h','o','j','\0'}; // +inicializace  
double da[2][3];               // dvourozm.pole  
char *sPtr;                     // ukazatel (pointer)
```

C - proměnné, paměťová třída

C - Deklarace, definice - umístění ve zdrojovém kódu:

- Mimo těla funkcí (tj mimo všechny bloky)
- Na začátek bloku {...}

C - Proměnná musí být deklarována dříve než se použije:

C - Paměťová třída proměnné (Storage Class):

- Paměťová třída definuje:
 - Životnost proměnné (*storage duration*) během provádění programu
 - Viditelnost proměnné (*scope*) z různých míst *daného modulu* programu
 - Viditelnost proměnné (*linkage*) z *ostatních modulů* programu
 - Pozn: zde modul programu=samostatně překládaný zdrojový soubor
- Paměťová třída *je definována*:
 - *Polohou* deklarace proměnné *v modulu* programu
 - *Specifikátorem_paměťové_třídy* (*storage class specifier*) (*auto, register, static, extern*)

C - proměnné, paměťová třída

C - Životnost proměnné (storage duration):

- *statická*
 - vytvoření - *jednou na začátku* běhu programu
 - zrušení - *po ukončení* programu
 - zajistí - překladač + [static]
 - inicializace - *explicitní* nebo *automatická na 0 / NULL*.
- *dočasná*
 - vytvoření - *automaticky vždy při vstupu* programu do bloku {...}
 - zrušení - *automaticky vždy po ukončení* bloku
 - zajistí - překladač + [auto]
 - inicializace - *explicitní* nebo *nedefinovaná !!*

Pozn: proměnné se nazývají *automatické* a zakládají se v zásobníku.

- *volitelná*
 - vytvoření - *na žádost* programátora za běhu programu
 - zrušení - *na žádost* programátora za běhu programu
 - zajistí - programátor voláním funkcí pro správu paměti
 - inicializace - *explicitní* nebo *nedefinovaná !!*

Pozn: proměnné se nazývají *dynamické* a přidělují se z haldy (heap)

C - proměnné, paměťová třída

C - Viditelnost proměnné (scope) v modulu (souboru):

- Globální v modulu
 - zajistí se - polohou deklarace v modulu + static
- Lokální v bloku {...}
 - zajistí se - polohou deklarace v bloku {...}

C - Viditelnost proměnné (linkage) v ostatních modulech (celý program):

- Globální v programu
 - zajistí se - polohou deklarace v bloku + [extern]
- Lokální v modulu
 - zajistí se - polohou deklarace v bloku + static

Pozn: viz následující příklady

C - proměnné, paměťová třída

C - Specifikátory paměťové třídy (Storage Class Specifiers - SCS):

SCS	Význam
Auto	Definuje proměnnou jako dočasnou (automatickou). Lze použít jen pro lokální proměnné deklarované uvnitř funkce. Implicitní nastavení je auto, její planost je omezena na život bloku, je v zásobníku
Register	Doporučuje překladači umístit proměnnou do registru procesoru (rychlost přístupu). Ten nemusí vyhovět (nemá-li volné registry). Jinak jako proměnné auto.
static	Deklaruje proměnnou jako statickou uvnitř bloku {...}. Vně bloku (kde je proměnná implicitně statická) omezuje její viditelnost na modul. Ponechává si hodnotu při opuštění bloku, existuje po celou dobu chodu programu, je v heapu
extern	Rozšiřuje viditelnost statických proměnných z modulu na celý program, globální proměnné, extern tam, kde se použije, definice bez

- Pozn: v deklaraci proměnné lze uvést vždy jen jeden SCS

C - proměnné, paměťová třída

C - Životnost a viditelnost proměnných:

LOKÁLNÍ V BLOKU

LOKÁLNÍ V MODULU

GLOBALNÍ V PROGRAMU

```
int i;  
static int max1=500;  
extern int j;  
  
int funkce1(int x, int y)  
{  
    int a1;  
    int a2=30;  
    static int a3=50;  
    //nelze  
    {  
        int b1;  
        b1=funkce2(4);  
    }  
}  
  
static int funkce2(int m)  
{. . .}
```

STATICKÁ

```
extern int i;  
static int max1=200;  
int j;  
int funkce1(int,int);  
void main(void)  
{  
    int test;  
        test=funkce1(4,max1);  
        //test=funkce2(I);  
  
    funkce3();  
}  
  
static void funkce3(void)  
{  
    char c='A';  
        . . .  
}
```

C - proměnné, struktura (struct)

C - Struktura (struct):

- Struktura je *množina prvků* (proměnných), které *nemusí* být *stejného typu*
- *Skladba* struktury je *definovaná uživatelem* jako *nový typ* sestavený z již definovaných typů
- K prvkům struktury *se přistupuje tečkovou notací*
- K prvkům struktury je *možné přistupovat i pomocí ukazatele* na strukturu operátorem *->*
- Struktury *mohou být vnořené*
- Pro struktury *stejného typu* je definována operace *přiřazení* *struct1=struct2* (pro proměnné typu pole přímé přiřazení není definováno, jen po prvcích)
- Struktury (jako celek) *nelze porovnávat* relačním operátorem *==*
- Struktura *může být do funkce předávána hodnotou i odkazem*
- Struktura *může být návratovou hodnotou funkce*

C - proměnné, struktura (struct)

C - Struktura (struct) - sablona (tag):

```
př: struct Tid{ // <==== Tid=jmeno sablony (tag)  
    char jmeno[20]; // Prvky struktury, pole  
    char adresa[50]; // - " - pole  
    long int telefon; // - " - int  
};  
  
struct Tid sk1,skAvt[20]; // struktura, pole struktur  
struct Tid *pid; // ukazatel na strukturu  
sk1.jmeno="Jan Novak"; // teckova notace  
sk1.telefon=123456;  
skAvt[0].jmeno="Jan Novak"; // prvek pole  
skAvt[3].telefon=123456;  
pid=&sk1; // do pid adresa struktury  
pid->jmeno="Jan Novak"; // odkaz pomoci ->  
pid->telefon=123456;  
(*pid).jmeno="Jan Novak"; // odkaz pomoci *  
(*pid).telefon=123456;
```


C - proměnné, struktura (struct)

C - Struktura (struct) - nový typ (typedef):

```
př: typedef struct {           // <==== Pomoci Typedef
    char jmeno[20];           // Prvky struktury, pole
    char adresa[50];         // - " -                pole
    long int telefon;        // - " -                int
}Tid,*Tidp;

Tid sk1,skAvt[20];          // struktura, pole struktur
Tidp pid;                  // ukazatel na strukturu
sk1.jmeno="Jan Novak";     // teckova notace
sk1.telefon=123456;
skAvt[0].jmeno="Jan Novak"; // prvek pole
skAvt[3].telefon=123456;
pid=&sk1;                  // do pid adresa struktury
pid->jmeno="Jan Novak";    // odkaz pomoci ->
pid->telefon=123456;
(*pid).jmeno="Jan Novak"; // odkaz pomoci *
(*pid).telefon=123456;
```

C - proměnné, struktura (struct)

C - Struktura (struct) - inicializace:

př:

```
    struct Tid{                               // <==== Tid=jmeno sablony
(tag)
        char jmeno[20];                       // Prvky struktury, pole
        char adresa[50];                     // - " -           pole
        long int telefon;                    // - " -           int
    };

    struct Tid sk1={"Jan Novak",
                   "Na kopecku 23",
                   123456};
```

C - proměnné, sdílená paměť (union)

C - Sdílená paměť (union):

- Union je *množina prvků* (proměnných), které *nemusí být stejného typu*
- *Prvky unionu sdílejí společně stejná paměťová místa* (překrývají se)
- *Velikost unionu je dána velikostí největšího z jeho prvků*
- Skladba unionu je *definovaná uživatelem jako nový typ* sestavený z již definovaných typů
- K prvkům unionu se *přistupuje tečkovou notací*

př:

```
union Tnum{                // <==== Tnum=jmeno sablony (tag)
    long n;
    double x
};

union Tnum nx;             // nx - promenna typu union
Tnum nx.n=123456789L;     // do n hodnota long
nx.x=2.1456;              // do x hodnota double (prekryva n)
```


C – funkce – parametry main()

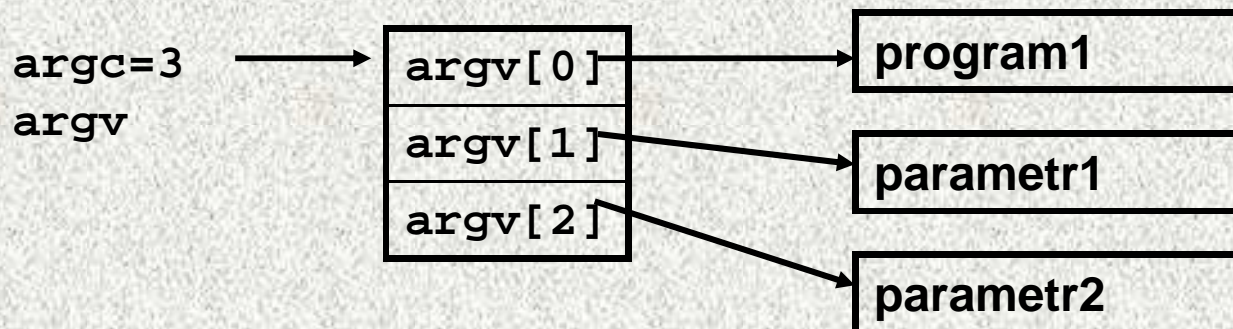
C - Funkce (function) – vstupní parametry funkce main():

- Funkce *main()* přebírá parametry z *příkazového řádku*
- *main()* má *dva vstupní* parametry:
 - argc* - počet parametrů na příkazovém řádku
 - *argv[]* - pole ukazatelů na příkazy příkazového řádku

```
int main(int argc, char **argv);  
int main(int argc, char *argv); // Totez
```

př:

příkazový radek ve tvaru `c:\>program1 parametr1 parametr2`



C - příkazy preprocesoru

C - Preprocesor (Preprocessor):

- *Zdrojový text* programu v C je před vlastní překladem *předzpracován*
- Předzpracování *provede Preprocesor* takto:
 - Odstraní komentáře
 - Nahradí makra jejich definicí
 - Vykoná ostatní direktivy preprocesoru
- Každá *direktiva* preprocesoru *začíná na samostatném řádku znakem #*
- Příliš *dlouhou* příkazovou řádku je možné *rozdělit znakem *

- *Direktivy* preprocesoru umožňují:
 - *Definovat makra a rušit jejich definice*
 - #define, #undef
 - *Testovat zda je makro definováno*
 - defined
 - *Vkládat do zdrojového textu hlavičkové soubory*
 - #include
 - *Řídit podmíněný překlad*
 - #if, #elif, #else, #endif, #ifdef, #ifndef
 - *Definovat nové příkazy preprocesoru závislé na implementaci*
 - #pragma

C - standardní knihovny

C – Standardní knihovny (ANSI C library – Standard Library) :

- Vlastní jazyk *C* *neobsahuje* žádné prostředky pro vstup a výstup dat, složitější matematické operace, práci s řetězci, třídění, blokové přesuny dat v paměti, práci s datem a časem, komunikaci s operačním systémem, správu paměti pro dynamické přidělování, vyhodnocení běhových chyb (run-time errors) apod.
- Tyto a další funkce jsou však *obsaženy ve standardních knihovnách (ANSI C Library)* dodávaných s překladači jazyka C.
- Uživatel *dostává* k dispozici *přeložený kód knihoven* (který se připojuje – linkuje k uživatelovu kódu) a *hlavičkové soubory* (headers) s *prototypy funkcí, novými typy, makry a konstantami*
- *Hlavičkové soubory* (obdoba interface v Javě) se *připojují k uživatelovu kódu* direktivou preprocesoru *#include <...>*.
- *Je zvykem, že hlavičkové soubory mají rozšíření *.h*, např. stdio.h

C - standardní knihovny

C – Standardní knihovny (ANSI C library – Standard Library) pokrač:

- Standardní knihovny jsou *rozděleny* do následujících částí:
(uvedeno pro ANSI C95):
- Vstup a výstup (formátovaný i neformátovaný)
 - stdin.h
- Rozsahy čísel jednotlivých typů
 - limits.h
- Matematické funkce
 - stdlib.h
 - math.h
- Zpracování běhových chyb (run-time errors)
 - errno.h
 - assert.h

C - standardní knihovny

C – Standardní knihovny (ANSI C library – Standard Library) pokrač:

- Standardní knihovny jsou rozděleny do následujících částí:
(uvedeno pro ANSI C95) pokrač:
- Klasifikace znaků (typ char)
 - ctype.h
- Práce s řetězci (string handling)
 - string.h
- Internacionalizace (adaptace pro různé jazykové mutace)
 - locale.h
- Vyhledávání a třídění
 - stdlib.h
- Blokové přenosy dat v paměti
 - string.h

C - standardní knihovny

C – Standardní knihovny (ANSI C library – Standard Library) pokrač:

- Standardní knihovny jsou rozděleny do následujících částí:
(uvedeno pro ANSI C95) pokrač:
- Správa paměti (Dynamic Memory Management)
 - `stdlib.h`
- Datum a čas
 - `time.h`
- Komunikace s operačním systémem
 - `stdlib.h`
 - `signal.h`
- Nelokální skok (lokální je součástí jazyka, viz `goto`)
 - `setjump.h`