13 Základy programovaní v C, charakteristika jazyka, model kompilace, struktura programu, makra, podmíněný překlad, syntaxe jazyka, struktury, uniony, výčtové typy, preprocesor, základní knihovny, základní vstup a výstup, pointery, dynamická správa paměti, pole a ukazatelé, funkce a pointery. (A0B36PR2)

13.1 C - Obecná charakteristika

C je nízkoúrovňový, kompilovaný, relativně minimalistický programovací jazyk.

- Univerzální programovací jazyk nižší až střední úrovně
  - Strukturovaný (funkce + data)
  - Zdrojový kód přenositelný (portable), nutno ctít podmínky přenositelnosti, překladač ne (je závislý na platformě)
  - Rychlý, efektivní, kompaktní kód, mnohdy nepřehledný
- Pružný a výkonný, ale nestabilní
- Podpora
  - konstrukcí jazyka vysoké úrovně (funkce, datové struktury)
  - operací blízkých assembleru (ukazatele, bitové operace,...)
    - Slabá typová kontrola
    - Málo odolný programátorovým chybám
- Dává velkou volnost programátorovi v zápisu programu

  - Výhoda: dobrý programátor vytvoří efektivní, rychlý a kompaktní kód
  - Nevýhoda: špatný nebo unavený programátor, pak nepřehledný program náchylný k chybám
- Nutná vlastní správa paměti
- silná vazba na HW, využití jeho možností (inline assembler), špatná nebo žádná podpora národních zvyklostí
- Použití:
  - operační systémy,
  - řídicí systémy
  - grafika
  - databáze
  - programování ovladačů grafických, zvukových a dalších karet
  - programování vestavěných - embeded systémů
  - číslicové zpracování signálů (DSP),
  - ...

13.2 Podobnost C a JAVA

- Program začíná funkcí main(), je základní funkcí programu, každý program musí obsahovat právě jednu tuto funkci
- Stavba funkcí / metod, – Jméno funkce, formální parametry, návratová hodnota, vymezení těla funkce, vymezení bloku, vlastnosti lokálních proměnných (jsou v zásobníku), předávání primitivních typů parametrů hodnotou, return.
- Množina znaků pro konstrukci identifikátorů
- Primitivní typy proměnných se znaménkem (Java nezná proměnné bez znam.)
  - char, short, int, long, float, double
- Aritmetické, logické, relační, bitové operátory
- Podmíněný příkaz if() / if() else
- Příkazy cyklů while() , do while(), for(;;), break, continue
- Programový přepínač switch(), case, default, break

```
int main (int argc, char** argv) {
```

```
    printf("I-copy-and-paste-all-the-time Policy \n");
    return (0);

}
//
//
// delsi ukazka vseho mozneho, je syntakticky spravne, nektere veci pridany jen pro u
// cyklus for, stdin, osetreni vstupu, konstanty...
#include <stdio.h>  /* hlavičkový soubor, přípona h */
#include <stdlib.h>
#include ‘‘konstanty.h’’ // uživatelské soubory se vkládají takto
#define NASOBITEL 5 /* symbolicka konstanta */
#include <math.h> // obsahuje funkce jako sqrt()...
// Zpracovani posloupnosti
// argc     počet parametrů na příkazovém řádku
// *argv[]  pole ukazatelů na příkazy příkazového řádku
int main(int argc, char** argv) {

    int i, suma, dalsi;
    const double PI = 3.14; // unused, jen ukazka -> v math.h neni PI
    printf("Zadejte 5 cisel \n");
    suma = 0;
    for(i = 1; i <= 5; i++){
        scanf("%d", &dalsi);
        // osetreni vstupu
        if(dalsi < 1){

            printf("\n n = %d neni prirozene cislo \n\n", n);
            exit(EXIT_FAILURE);
        }
        suma = suma + dalsi;
    }
    printf("suma = %d \n\n", suma);
    return (EXIT_SUCCESS);

}
```

13.3 Co C nemá

- Interpret kódu (JVM) (C je kompilovaný)
- Objektovou podporu
  - Třídy, objekty, zapouzdření, dědičnost, polymorfismus

- Jednotnou metodiku vytváření a použití strukturovaných proměnných
  - referenční proměnná, new()
- Automatickou správu paměti
  - Garbage collector
- Velikost proměnných nezávislou na platformě
- Způsob uložení proměnných nezávislý na OS (Little-endian, Big endian,...)
- Ošetření výjimek metodikou chráněných bloků
  - try, catch, finally
- Standardní podporu grafického uživatelského rozhraní GUI
- Standardní podporu řízení událostí (events, listeners)
- Standardní podporu webovských aplikací (aplety, síťové připojení)
- Standardní podporu (semi)paralelního zpracování úloh
  - threads, multitasking, multithreading

13.4 Co C nemá, nebo je jinak

- C je kompilovaný jazyk
  - Zdrojový kód je nezávislý (portable) na platformě (málo závislý)
  - Spustitelný kód je závislý na platformě
- Členění programu na moduly, určení jejich vazeb (interface)
- Import knihoven (systémových i uživatelských)
- Určení doby života proměnných, vlastní správa paměti
- Definování konstant a maker
- Vytváření strukturovaných proměnných (mohou být i statické)
- Určení viditelnosti proměnných, modifikátory přístupu
- Ošetření běhových chyb (run-time errors)
  - odpovědný programátor, překladač nevynucuje
- Správa paměti (heap management)
  - odpovědný programátor, malloc / free
- Práce s booleovskými proměnnými a řetězci (boolean a String není)

13.5 Co je v C navíc

- Preprocessor
  - Vkládání hlavičkových souborů (header file) do zdrojového kódu
  - Podmíněný překlad
  - Makra
  - #pragma – doplňující příkazy závislé na platformě
- Linker – spojování přeložených modulů a knihoven do spustitelného kódu
- x (už NEPLATÍ) enum – výčtové typy (množina číslovaných pojmenovaných konstant)
- **Ukazatele (pointer) jako prostředek nepřímého adresování proměnných**
- struct a bitová pole (strukturované proměnné z různých prvků)
- union – překrytí proměnných různého typu (sdílení společné paměti)
- typedef – zavedení nových typů pomocí již známých typů
- sizeof – určení velikosti proměnné (i strukturovaného typu)
- Jiné názvy i parametry funkcí ze standardních knihoven (práce se soubory, znaky, řetězci, matematické funkce, ...)
- příkaz goto navesti;

```
int hledejMax(int *p) {
    for(. . .) {
        for(. . .) {
            if(. . .)
              goto error; // Ven z vnitrniho bloku
        }
    return(. . .);
    error: // Cil skoku uvnitr funkce
        return(. . .);
    }
```

13.6 Program C obsahuje

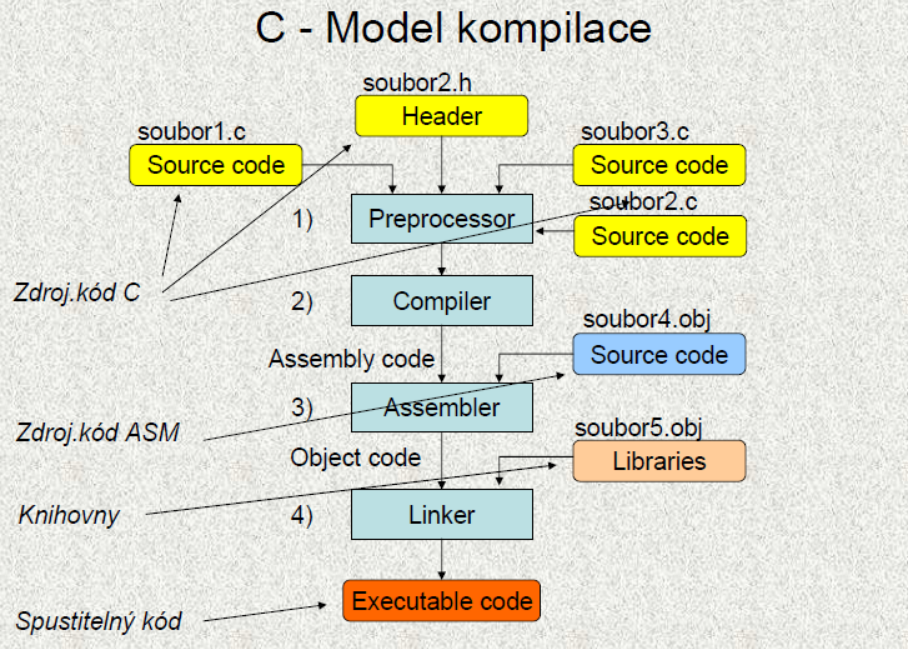
- Příkazy preprocesoru (Preprocessor commands)
- Definice typů (type definitions)
- Prototypy funkcí (function prototypes) kde je uvedena deklarace:
  - Jména funkce
  - Vstupních parametrů
  - Návratové hodnoty funkce
- Proměnné (variables)
- Funkce (functions) (procedura v C je funkce bez návratové hodnoty nebo void)
- Komentáře: //, /* */, nesmí být vnořené

Uvnitř funkce nelze definovat lokální funkce (definice funkci nesmí být vnořené).

13.7 C kompilace

1. Preprocesor:
   a) Čte zdrojový kód v C
   b) Odstraní komentáře (nahradí každý komentář jednou mezerou, pozor: některé překladače nepodporují // komentáře, pouze /* */)
   c) Upraví zdrojový text podle direktiv preprocesoru (řádky začínající #)
      i. Vloží do textu obsah jiného souboru #include ....
      ii. Odebere text vymezený direktivami podmíněného překladu
      iii. Expanduje makra
2. Překladač C:
   a) Čte výstup z preprocesoru
   b) Kontroluje syntaktickou správnost textu
   c) Hlásí chyby a varování
   d) Generuje text v assembleru (když nejsou chyby)
3. Assembler:
   a) Čte výstup z překladače C
   b) Generuje relokovatelný object kód (kód s nevyřešenými odkazy mezi moduly)
   c) Přeloží případné moduly zapsané přímo v assembleru (mix programovacích jazyků)

4. Linker (spojovací program):
   a) Čte object kód všech zúčastněných modulů programu
   b) Připojí knihovní object moduly (přeložené dříve nebo dodané)
   c) Vyřeší odkazy mezi moduly
   d) Generuje spustitelný kód (zjednodušené)



13.8 Celočíselné typy

- Rozsahy celočíselných typů v C nejsou dány normou, ale implementací (pro 16ti a 64 bitové prostředí jsou jiné než je uvedeno v následující tabulce - ta je pro 32 bitové prostředí)
- limits.h, float.h
- Norma pouze garantuje
  - short <= int <= long unsigned short <= unsigned <= unsigned long
- Celočíselné literály (zápisy čísel):
  - dekadický 123 456789
  - hexadecimální 0x12 0xFFFF (začíná 0x nebo 0X)
  - oktalový 0123 0567 (začíná 0)
  - unsigned 123456U (přípona U nebo u)

  - long 123456L (přípona L nebo l)
  - unsigned long 123456UL (přípona UL nebu ul)
- Není-li uvedena přípona, jde o literál typu int

C – Celočíselné datové typy

Typ	Velikost [byte]	Rozsah	Použití
char	1	-128 až +127 nebo 0 až 255	Znaky
unsigned char	1	0 až 255	Malá čísla
signed char	1	-128 až +127	Malá čísla
int	2 nebo 4	-32.768 až +32.767 nebo -2.147.483.648 až +2.147.483.647	Celá čísla
unsigned int	2 nebo 4	0 až 65.535 nebo 0 až 4.294.967.295	Kladná celá čísla
short	2	-32.768 až +32.767	Celá čísla
unsigned short	2	0 až 65.535	Kladná celá čísla
long	4	-2.147.483.648 až +2.147.483.647	Velká celá čísla
unsigned long	4	0 až 4.294.967.295	Kladná celá čísla

•Typ boolean není (až ANSI C99), =0 ->false, !=0 -> true

- C - racionální čísla (neceločíselné datové typy):
  - Velikost reálných čísel určená implementací
  - Většina překladačů se řídí standardem IEEE-754-1985, potom jsou rozsahy reálných čísel dány následující tabulkou:

Typ	Velikost [byte]	Rozsah (uveden pro kladná č.)	Přesnost
float	4	1.2E-38 až 3.4E+38	6 desítkových číslic
double	8	2.3E-308 až 1.7E+308	15 desítkových číslic
long double	10	3.4E-4932 až 1.1E+4932	19 desítkových číslic

- C - typ void: void značí prázdnou hodnotu nebo proměnnou bez typu (jen ukazatelé)
  - void funkce1 (....) - fukce bez návratové hodnoty (procedura)
  - int funkce2 (void) - funkce bez vstupních parametrů
  - void *ptr; - ukazatel bez určeného typu (viz dále)
- #define konstanta
  - #define CERVENA 0 /* Zde muze byt komentar */

- je to makro bez parametrů, každé #define musí být na samostatné řádku)
- Preprocesor provede textovou náhradu všech výskytů slova CERVENA znakem 0
- může být i vnořená: #define MAX_2 MAX_1+30

13.9 Funkce

C je modulární jazyk = funkce je jeho základním stavebním blokem.

- Každý program v C obsahuje minimálně funkci main()

  ```
  int main(int argc, char** argv) { ... }
  ```

- Běh programu začíná na začátku funkce main()
- Definice funkce obsahuje hlavicku funkce a její tělo
- C používá prototypu funkce k deklaraci informací nutných pro překladač, aby mohl správně přeložit volání funkcí i v případě, že definice funkce je umístěna dále v kódu modulu nebo je jiném modulu

  ```
  int max(int a,int b);
  ```

- Deklarace se skládá pouze z hlavičky funkce, (odpovídá interface v Javě)
- Parametry se do funkce předávají hodnotou (call by value), parametrem může být i ukazatel (pointer). Ten pak dovolí předávat parametry i odkazem.
- C nepovoluje funkce vnořené do jiných funkcí (lokální funkce ve funkci)
- Jména funkcí jsou implicitně extern, a mohou se exportovat do ostatních modulů (samostatně překládaných souborů)
- Specifikátor static před jménem funkce omezí viditelnost jejího jména pouze na daný modul (lokální funkce modulu)
- Formální parametry funkce jsou lokální proměnné inicializované skutečnými parametry při volání funkce
- C dovoluje rekurzi, lokální proměnné jsou pro každé jednotlivé volání zakládány znovu (v zásobníku). Kód funkce v C reentrantní (reentrant = Reentrantní provádění bloků znamená, že je možné provádět několikanásobně volaný blok paralelně.).
- Funkce nemusí mít žádné vstupní parametry, zapisuje se funkceX(void)
- Funkce nemusí vracet žádnou funkční hodnotu, pak je návratový typ void (je to procedura)

- Pokud v definici parametrů funkce je klíčové slovo const - tento parametr (předaný odkazem, např. pole) nelze uvnitř funkce měnit
  - př. int fce (const char *src) { ... } , z pole lze pouze číst, jeho prvky nelze uvnitř funkce měnit

13.9.1 Specifikátory paměťové třídy

C - Specifikátory paměťové_třídy (Storage Class Specifiers - SCS):

SCS	Význam
auto (lokální)	Definuje proměnnou jako dočasnou (automatickou). Lze použít jen pro lokální proměnné deklarované uvnitř funkce. Implicitní nastavení je auto, **její platnost 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, **v datové oblasti**
extern	Rozšiřuje viditelnost statických proměnných z modulu na celý program, globální proměnné, extern tam, kde se použije, definice bez **v datové oblasti**

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

x JAVA: globální proměnné nejsou, obchází se také deklarací static (ale je třeba si uvědomit odlišný význam, static v JAVE způsobí, že proměnná existuje pouze jednou pro celou třídu, všechny instance ji sdílí (konstanty v JAVE - final))

13.9.2 Stdin

Načtení hodnoty ze stdin: scanf("%d", &x);

- do funkce scanf vstupuje jako parametr adresa (resp. reference) paměťového místa, kam se má načtená hodnota uložit
- jde o předání parametru odkazem, jde tedy o parametr, který může být využit pro vstup i výstup hodnoty (x JAVA - parametry pouze hodnotou (primitivni typy))
- funkce v C může takto „vracet “ více hodnot

13.9.2.1 Bezpečné načtení double v C

```
#include <stdio.h>
#include <stdlib.h>
int nextDouble(double *cislo){

    // === Bezpecne pro libovolny zadany pocet znaku ===
    // Navratova hodnota:
```

```
    // TRUE - zadano realne cislo
    // FALSE - neplatny vstup
    enum boolean {FALSE,TRUE}; // v ANSI C99 uz existuje true a false, zde výčty
    const int BUF_SIZE = 80;
    char vstup[BUF_SIZE],smeti[BUF_SIZE];
    // fgets precte az sizeof(vstup)-1 znaků ze stdin (standardní vstup) a pushne je
    fgets(vstup,sizeof(vstup),stdin);
    // sscanf je jako scanf, ale cteno z bufferu
    if(sscanf(vstup,"%lf%[^\n]",cislo,smeti) != 1)
        return(FALSE); // Input error
    return(TRUE);
}
```

13.10 Dynamická správa paměti - Alokace paměti

Pole lze alokovat normálně staticky, musí se ale předem zadat jeho velikost.
Způsob dynamické alokace:

```
(int*)malloc(count*sizeof(int)) //zde je typ proměnné int*, ne int !
```

Komplexnější příklad použití:

```
int* ctiPole1 (int *delka, int max_delka){

    // Navratovou hodnotou funkce je ukazatel na pridelene pole
    int i, *p;
    printf(" Zadejte pocet cisel = ");
    // funkce nextInt je temer totozna, jako funkce nextDouble predstavena vyse
    if (!nextInt(delka)) {

        printf("\n Chyba - Zadany udaj neni cele cislo\n\n");
        exit(EXIT_FAILURE);
    }
    if(*delka < 1 || *delka > max_delka){

        printf("\n Chyba - pocet cisel = <1,%d> \n\n",MAX_DELKA);
        exit(EXIT_FAILURE);
    }
    // Alokace pameti (prideleni pameti z "heapu")
    if((p=(int*)malloc((*delka)*sizeof(int))) == NULL){
```

```
            printf("\n Chyba - neni dostatek volne pameti \n\n");
            exit(EXIT_FAILURE);
        }
        printf("\n Zadejte cela cisla (kazde ukoncit ENTER)\n\n");
        for (i = 0; i < *delka; i++) {
            if (!nextInt(p+i)) {

                printf("\n Chyba - Zadany udaj neni cele cislo\n\n");
                exit(EXIT_FAILURE);
            }
        }
        return(p); // p - ukazatel na pridelene a naplnene pole

    }
```

Dealokace probíhá pomocí funkce free, předává se jí ukazatel na začátek alokované paměti:

```
void free(void *ptr)
```

13.11 Operátory

Jako JAVA. Výraz může být operandem, výraz má typ a hodnotu (x void hodnotu nemá). Priority operátorů:

C - Priorita a asociativita operátorů

- Pořadí vyhodnocování výrazu se řídí prioritou a asociativitou
- 18/2*3+11 je 38 .... (18/2)*3 +11

Priorita	Operátor	Asociativita		Priorita	Operátor	Asociativita
1	() [] ->	L->R		9	^	L->R
2	! ~ ++ -- + - (type) * & sizeof	R->L		10	|	L->R
3	* / %	L->R		11	&&	L->R
4	+ -	L->R		12	||	L->R
5	<< >>	L->R		13	?:	R->L
6	< <= > >=	L->R		14	= += -= *= /= %= &= ^= |= <<= >>=	R->L
7	== !=	L->R		15	,	L->R
8	&	L->R				

- Ve výrazu: funkce1() + funkce2(), není definováno, která funkce se provede jako první.

- Máme (2 příkazy): int a = 3; a+=a++ + ++a * a++;
  - v JAVE: 31
  - v C: 26, 31, ... není definováno pořadí, programátor není upozorněn, že jde o nejednoznačný zápis
- Zkrácené vyhodnocování logických operátorů (viz JAVA)
- Operandy musí být stejného aritmetického typu, nebo oba struct nebo union stejného typu, nebo oba pointery stejného typu (pravý může být NULL)

Přístup do paměti - operátory:

C - Operátory - přístupu do paměti

Operátor	Význam	Příklad	Výsledek
&	Adresa proměnné	&x	Konstantní pointer na x
*	Nepřímá adresa	*p	Proměnná nebo funkce adresovaná ukazatelem p
[ ]	Prvek pole	x [i]	*(x+i), prvek pole x s indexem i
.	Prvek struct / union	s.x	Prvek x struktury / unionu s
->	Prvek struct / union	p->x	Prvek x struktury / unionu s adresovaný ukazatelem p

- Operandem operátoru & *nesmí být* - *bitové pole* a proměnná třídy *register*
- Operátor *nepřímé adresy* * - umožňuje přístup pomocí *ukazatele (pointer)*

```
př: int a,*pa; // proměnná int a ukazatel na int
    pa=&a;     // adresa a do pa
    *pa=45;    // totez jako a=45
př: double a[10],*pa; // proměnná int a ukazatel na int
    pa=a;      // adresa pole a[] do pa (neni &)
    *(pa+3)=12; // totez jako a[3] nebo pa[3]
```

Přehled 55

13.12 Ukazatelé - Pointery

- Motivace
  - Predávání parametru odkazem
  - Práce s poli, rízení pruchodem polem
  - Pointer na funkci
  - Využívání pole funkcí
  - Vytvárení seznamových struktur
  - Hešování
- Na rozdíl od Javy je možné s ukazatelem provádet aritmetické operace

- Ukazatel v C je prímo implementován pametí procesoru a je možné prímo adresovat, vcetne požadavku na registry
  - Mocný nástroj pro implementaci strojove orientovaných aplikací
- Ukazatel (pointer) je promenná jejíž hodnotou je „ukazatel“na jinou promennou (analogie neprímé adresy ve strojovém kódu ci v assembleru)
- Ukazatel má též typ promenné na kterou muže ukazovat
  - ukazatel na char, int,..
  - „ukazatel na pole“
  - ukazatel na funkci
  - ukazatel na ukazatel
  - atd.
- Ukazatel muže být též bez typu (void), pak muže obsahovat adresu libovolné promenné. Její velikost pak nelze z vlastností ukazatele urcit
- Neplatná adresa, ale definovaná v ukazateli má hodnotu konstanty NULL (kterémukoliv pointeru lze přiřadit hodnotu NULL)
- C za behu programu nekontroluje zda adresa v ukazateli je platná •
- Specialitou C je pointer na funkci!
  - ukazatel umožní, aby funkce byla parametrem funkce
  - ukázka:

```
void funkce1(int x); // Prototyp funkce
void (*pFnc)(int x); // Ukazatel na funkci s parametrem int
int max=200;
pFnc=funkce1; // Adresa funkce1 do ukazatele pFnc
(*pFnc)(max); // volani funkce ‘‘funkce1’’ s parametrem max
```

- Pomocí ukazatele lze predávat parametry funkci odkazem (call by reference) – základní využití
- Adresa promenné se zjistí adresovým operátorem & (ampersand), tzv. referencní operátor (promenná >>> adresa této promenné) int x;
- K promenné na kterou ukazatel ukazuje se pristoupí operátorem neprímé adresy * (hvezdicka), tzv. dereferencní operátor (promenná ukazatel >> hodnota z adresy, kam ukazuje) int *px; px=&x;

13.12.1 Ukázka pro lepší pochopení

```
int x=30; // promenná typu int,  &x - adresa promenné x
int *px; // *px promenná typu int, px promenná typu pointer na int
px=&x; // do promenné typu pointer na int se uloží adresa promenné x
printf(" %d " " %d \n", x, px); // 30 2280564
printf(" %d " " %d \n", &x, *px); // 2280564 30
printf(" %d " " %d \n", *(&x), &(*px)); // 30 2280564
```

13.12.2 Další ukázka

```
int x;
int *px;
int **ppx;
x=1;
px=NULL;
ppx=NULL;
px=&x;
ppx=&px;
**ppx=6;
*px=10;
x = 55;
// *ppx = 20 // CHYBA, konverze int na int* nelze
// px = 20 // CHYBA, konverze int na int* nelze

printf(" %d " " %d " " %d" " \n", x, *px, **ppx); // 55  55  55
printf(" %d %d %d %d " , &x, px, ppx, *ppx); // 2293527  2293527  2293527  (pozn. poi
```

13.12.3 Operace s pointery

- Povolené aritmetické operace s ukazateli:
  - –pointer + integer
  - pointer - integer
  - pointer1 - pointer2 (musí být stejného typu)
- Povolené operandy relace:
  - dva ukazatele (pointers) shodného typu nebo jeden z nich NULL nebo typu void
- Jednoduché prirazení - povolené operandy
  - dva operandy typu - pointer (stejného typu) nebo pravý operand=NULL nebo jeden pointer typu void
- Aritmetické operace jsou užitecné když ukazatel ukazuje na pole

13.13 Pole

- Pole je množina prvku (promenných) stejného typu
- K prvkum pole se pristupuje pomocí poradového císla prvku (indexu) •
- Index musí být celé císlo (konstanta, promenná, výraz)
- Index prvního prvku je vždy roven 0
- Pole se funkcím předává odkazem: void funkcePole (int p[])
- Prvky pole mohou být promenné libovolného typu (i strukturované)
- Pole muže být jednorozmerné i vícerozmerné (prvky pole jsou opet pole)
- Definice pole urcuje:
  - jméno pole
  - typ prvku pole
  - pocet prvku pole
- Prvky pole je možné inicializovat
- Pocet prvku statického pole musí být znám v dobe prekladu
- Prvky pole zabírají v pameti souvislou oblast!
- Velikost pole (byte) = pocet prvku pole * sizeof (prvek pole)
- C - nemá promennou typu String, nahrazuje se jednorozmerným polem z prvku typu char. Poslední prvek takového pole je vždy '\0' (null char)
  - lze také zapsat char *c; c = "ja jsem string"; // c je ukazatel na prvni prvek pole
- C - nekontroluje za behu programu, zda vypocítaný index je platný!

C - Deklarace pole (array):

```
char poleA [9]; // jednorozmerne pole z prvku char
int poleB[3][3]; // dvourozmerne pole z prvku int
Uložení v paměti
```

poleB [0][0], [0][1], [0][2], [1][0], [1][1], [1][2], [2][0], [2][1], [2][2],

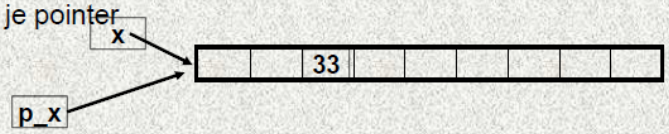
C - Inicializace pole:

```
double x[]={0.1, 0.4, 0.5};
char s[]="abc";
char s1[]={'a','b','c','\0'}; // totez jako "abc"
int ai[3][3]={{1,2,3}, {4,5,6}, {7,8,9}};
char cmd[][10]={"Load","Save","Exit"};
                                // druhy rozmer nutny
```

C - Přístup k prvkům pole:

```
ai[1][3]=15*2;
```

Identifikátor pole je pointer




```
int x[9];
x[2]=33;
x[i] ~~ obsah prvku pole i
// x ~~ pointer na počátek pole
//&x[i] ~~ adresa prvku pole - "adresa x" + i * sizeof(int)
//x[i] ~~ obsah prvku pole - *(x + i)
int *p_x;

p_x = x; // ~~ p_x = &x[0];

for (i=0;i<9;i++)x[i]=0;
for (i=0;i<9;i++)(p_x + i)=0;
```

- **Jméno pole je konstantní ukazatel na počátek pole (na prvek x[0])**

13.14 Hlavičkové soubory

- 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 make

- „obdoba interface“

- Příklad soubor xxx.h:

```
/* podmineny preklad proti opakovanemu vkládání ,,include‘‘ */
#ifndef XXX  // čti: if not defined = proti duplicitě = podmíněný překlad
#define XXX
/* definice symb. konstant vyuzivanych i v jinych modulech */
#define CHYBA -1.0
/* definice maker s parametry */
#define je_velke(c) ((c) >= ‘A’ && (c) <= ‘Z’)
/* definice globalnich typu */
typedef struct{

    int vyska;
    int vaha;

} MIRY;
/* deklarace globalnich promennych modulu xxx.c */
extern MIRY m; // v jiném modulu bude definice MIRY m;
/* uplne funkcni prototypy globalnich funkci modulu xxx.c */
extern double vstup_dat(void);
extern void vystup_dat(double obsah);
#endif
```

- Příklad soubor xxx.c (includuje xxx.h)

```
#include <stdio.h> /* standardni vklad*/
#include ,,xxx.h" /* natazeni konstant,prototypu funkci a globalnich typu vlastniho mo
/* deklarace globalnich promennych */
extern int z; /*ktere nebyly definovány v hlavičkovém soubor */
/* definice globalnich promennych */
int y;/* které nejsou definovány v hlavičkovém soubor */
/* lokalni definice symbolickych konstant a maker */
#define kontrola(x) ( ((x) >= 0.0) ? (x) : CHYBA_DAT )
/* lokalni definice novych typu */
typedef struct{} OSOBA;
```

```
/* definice statickych globalnich promennych */
static MIRY m;
/* uplne funkcni prototypy lokalnich funkci */
int nextDouble(double *cislo);
/* funkce main() */
int main(int argc, char** argv){}
/* definice globalnich funkci - to, ze je glob., bylo definovano v xxx.h */
double vstup_dat(void){ ...return ();}
/*funkční prototypy v.h souboru*/
void vystup_dat(double obsah){ ... }
/* definice lokalnich funkci */
int nextDouble(double *cislo){... }
```

13.15 Typedef

Umožňuje vytvářet nové datové typy:

```
typedef double *PF;
typedef int CELE;
PF x,y;
CELE i,j;
```

13.16 Struktury

- Struktura je konečná množina prvků (proměnných), které nemusí být stejného typu
  - Obdoba třídy bez metod v Javě
  - Record v jiných jazycích
- 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 (jak se to řeší v Javě? odpověď: Patrně kompozicí.)
- Pro struktury stejného typu je definována operace přiřazení struct1=struct2
  - (pro proměnné typu pole přímé přiřazení není definováno, jen po prvcích) – co z toho vyplývá???
- Struktury (jako celek) nelze porovnávat relačním operátorem ==
  - co z toho vyplývá???

- Struktura může být do funkce předávána hodnotou i odkazem
- Struktura může být návratovou hodnotou funkce

```
typedef struct {   // <==== Pomoci Typedef
char jmeno[20];    // Prvky struktury, pole
char adresa[50];   // - ‘‘ - pole
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;
```

13.17 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
nx.n=123456789L; // do n hodnota long
nx.x=2.1456; // do x hodnota double (prekryva n)
```

13.18 Podmíněný překlad

```
#if VERSE_CITACE == 1
    do {
        ...
    } while(TRUE);
#endif
```

13.19 Definice vs. deklarace

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

13.20 Standardní knihovny

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.

Příklad:

- 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
- 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
- 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