Jazyk C pro programátory v jazyce Java

Doplňkový text předmětu
„Úvod do počítačových sítí“

Úvod

- Proč ještě C když znám Javu
  - Jazyk použitelný na vyšší i nižší úrovni
  - Lepší kontrola vnitřních mechanizmů
  - Někdy i větší výkonnost než Java
- Java zakrývá řadu detailů
  - Organizace a přidělování paměti
  - Explicitní inicializace a detekce chyb
  - Větší prostor pro chyby
  - Delší zdrojový kód pro řešení srovnatelných problémů

Úvod

- Historie a vlastnosti
  - Pochází z konce 60 let a začátku 70 let
  - Procedurální jazyk pro systémové programování
  - Přímý přístup k systémovým voláním
  - Přenositelnost programů

Odlišnosti

C

- Funkčně orientovaný
- Přetěžované typování
- Omezené přetypování
- Jednoduchý jmenný prostor
- Společná makra
- Víceúrovňový I/O model
- Funkce pro manipulaci s pamětí
- Pointery
- Parametry přenášené hodnotou
- Nastavení kódu chyby, signály

Java

- Objektově orientovaný
- Přísné typování
- Polymorfismus
- Třídy pro jmenný prostor
- Externí, málo používaná
- I/O bytově orientovaný
- Automatická manipulace s pamětí
- Bez pointerů
- Hodnotou nebo odkazem
- Výjimky, zpracování výjimek

Jednoduchý program

- Soubor funkcí, startovací funkce main()
- V programu je celá uživatelský kód slinkovaný s knihovnami, které mohou být i dynamické

Jednoduchý program

```
#include <stdio.h>
int main( void )
{
        puts(„Hello world\n“);
        printf(„Hello World. \n \t Hello\n “);
        exit( 0 );
}
```

Kompilace

```
gcc –o hello hello.c
```

Spuštění

```
hello
```

Některé parametry volání gcc

Parametry volání gcc

-o soubor	výstupní soubor
-Wall	detailní varování
-c	kompilace jednoho modulu
-g	vložení kódu pro debuger
-p	vložení kódu pro profilování
-E	výstup preprocesoru
-l knihovna	knihovna
-L cesta	cesta do adresáře s knihovnami

Překlad a sestavení s více moduly

```
gcc –c prvni.c
gcc –c druhy.c
gcc –o vystup prvni.o druhy.o
```

C preprocesor

C preprocesor - makra

```
#define LIMIT 100
#undef LIMIT
#define maximum(a,b) ((a) > (b) ? (a) : (b))
```

C preprocesor – podmíněný překlad

```
#if logický výraz                    #if (a==b) || (c==d)
    segment 1
#else
    segment 2
#endif

#ifdef LIMIT nebo #if defined( LIMIT )
    segment 1
#else
    segment 2
#endif
```

Komentáře

```
/* komentar */
// komentar do konce radku
```

C preprocesor

C preprocesor – vkládání souborů

```
#include “soubor.h”      relativně k aktuálnímu adresáři
#include <soubor.h>      relativně k /usr/include
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mem.h>
#include <stddef.h>

#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
```

```
#include “soubor.h”
```

soubor.h

```
#ifndef SOUBOR_H
#define SOUBOR_H
#ifndef JINY_SOUBOR_H
        #error "include jiny_soubor.h "
#endif
```

```
#endif /* soubor.h */
```

Numerické datové typy

Numerické datové typy (signed, unsigned)

char	celé číslo 1 byte	**-128 to 127**
short	celé číslo 2 byte	**-32,768 to 32,767**
int, long	celé číslo 4 byte	**-2,147,483,648 to 2,147,483,647**
long long	celé číslo 8 byte	
float	reálné číslo 4 byte	**3.4E+/-38**
double	reálné číslo 8 byte	**1.7E+/-308**
boolean	není, 0 – **false**, 1 (nenulová hodnota) – **true**	

signed/unsigned char, short, int, long

Konverze typů

převod pravé strany na výsledný typ + konverze a přiřazení výsledku na levou stranu

double ← float ← int ← char, short

přetypování: **c = (char) x;**

Definice uživatelských typů a proměnných

Definice uživatelských typů

```
typedef unsigned char BYTE;
typedef enum { pondeli, utery, streda, ctvrtek, patek } tyden;
nebo
#define pondeli 0
#define utery   1
#define streda  2
#define ctvrtek 3
#define patek   4
```

Definice proměnných

```
short x, y;
float pi = 3.1415;
char ch = 'a';
```

Odkazy (pointery)

Definice

typ promenna	definuje jméno proměnné a její typ, rezervuje místo pro hodnotu
typ * promenna	definuje jméno proměnné a její typ je odkazem na proměnnou nějakého typu, rezervuje místo pro adresu

& — operátor adresy
* — operátor dereference

Použití vpravo od rovnítka v přiřazovacím příkazu

&promenna	pracuje se adresou proměnné
promenna	pracuje se s obsahem (hodnotou) proměnné
*promenna	pracuje s obsahem proměnné jako s adresou místa, kde je uložena hodnota
**promenna	obsah promenne se chápe jako adresa místa, ze kterého se vybere obsah, který je chápán jako adresa místa, kde je uložena hodnota

Odkazy (pointery)

Použití vlevo od rovnítka v přiřazovacím příkazu

&promenna	nemá smysl
promenna	promenna je adresou, kam se uloží hodnota
*promenna	promenna obsahuje adresu, kam se uloží hodnota
**promenna	promenna obsahuje adresu, která obsahuje adresu, kam se uloží hodnota

Příklady

int B = 5;	proměnná B typu int s inicializovaná na hodnotu 5
int *A = NULL;	proměnná A může obsahovat adresy (odkaz) na nějakou proměnnou typu int
A = &B;	do proměnné A se dosadí adresa proměnné B
*A = 8;	na adresu uloženou v A (tedy do B) se uloží číslo 8
*A++	hodnota v B se zvýší o 1

Odkazy (pointery)

Příklady

```
short *r, *s, tmp;
short a, b;

r = &a; s = &b;
tmp = *r; *r = *s; *s = tmp;

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

Odkazy (pointer) typu void * mohou být přiřazeny libovolnému jinému typu pointer

```
void *v;
char *s = v;
```

Odkaz na funkci

int fce();	funkce vracející integer
int *fce();	funkce vracející odkaz (pointer) na integer
int (*fce)();	pointer na funkci, vracející integer
int *(*fce)();	pointer na funkci, vracející pointer na integer

```
void fce( int x ) { . . . }
void univerzalni_funkce( void (*f)(int), int p )
    { . . .
        (*f)(p);
    . . .
    }
void main( void ) {  . . .      univerzalni_funkce( fce, 10 );  fce( 10 );      . . . }
```

Řídicí struktury

```
příkaz_1;  příkaz_2; . . . příkaz_n;
```

```
if( podminka) prikazy;
    else if (podminka) prikazy;
else prikazy;
```

```
switch( vyraz ) {
    case konstanta_1: prikazy; break;
    case konstanta_2: prikazy; break;
default: prikazy;
}
```

Řídicí příkazy

while(vyraz) prikazy; i = 0; while(i < 100) a[i] = i++;	opakuje nula nebo vícekrát nejdříve testuje, pak přiřazuje
do { prikazy } while(vyraz); i = 0; do { a[i] = i++; } while(i <100);	opakuje jednou nebo vícekrát nejdříve přiřazuje, pak testuje
for(od; podminka; vyraz) prikazy; for(i = 0, j = 0; i < 10; i++, j++) a[i,j] = i+j;	opakuje od hodnoty pokud platí podmínka a vyhodnotí vyraz
for(;;) prikazy;	nekonečná smyčka

Příkaz break, continue return a exit

Příkaz break

Předčasně ukončí jednu úroveň cyklu

```
for( j = 0; j < 20; j++ ) {
    prikaz_1;
    for( i = 0; i < 10; i++ ) {
        prikaz_2;
        if( i > 5 ) break;
        prikaz_3;
    }
    prikaz_4;
}
```

Příkaz exit

Konec programu

```
exit;
exit( návratová hodnota);
```

Příkaz continue

Přeskočí zbytek cyklu.

```
for( i = 0; i < 10; i++ ) {
    prikaz_1;
    if( i > 5 ) continue;
    prikaz_2;
}
```

Příkaz return

Návrat z podprogramu

return;	nevrací nic (typ návratové hodnoty void)
return a+b;	vrací součet

Strukturovaná data

Homogenní datová struktura – pole

```
int x[100];          celočíselné pole o sto prvcích [0 .. 99]
```

Vícerozměrná pole

```
int x[10] [20];      matice o 10 řádcích a 20 sloupcích
```

Nehomogenní datová struktura – record

```
struct {
        int a;
        char b;
        char c[10];
} xxx;
```

Příklady

```
xxx pokus;
xxx *odkaz;
        pokus.a = 0; pokus.b = 'a'; pokus.c[1] = 'b';

        (*odkaz).a = 0; odkaz->a = 0;
```

Strukturovaná data

Příklad

```
#define PORT 2345
struct sockaddr_in sin;

        sin.sin_family = AF_INET;
        sin.sin_port = htons( (unsigned short)PORT );
        sin.sin_addr.s_addr = INADDR_ANY;
```

Struktury s bitovými poli

```
        struct{
                unsigned int a:1;
                unsigned int b:2;
                unsigned int c:5;
        } bit_mask;
bit_mask x;
        x.a = 1; x.b = 2; x.c = 20;
```

Uniony

Uniony jsou podobné strukturám, ale vyhradí prostor pro nejdelší složku – složky se překrývají.

```
        union{
                short a;
                short b;
                char c;
                char d;
                char e[10];
        } xxx;

xxx pokus;
        pokus.a = 5; je totéž jako pokus.b = 5;
        pokus.a == (pokus.c + 256*pokus.d);
```

Funkce

```
char x;
void fce( char a, char *b )
    {
        *b = a; return;
    }

fce( 'a', x);          // x = 'a';
```

```
char x;
char fce( char a )
{
    char b;          // uloží se do zásobníku,
                     // po ukončení volání zmizí
    return a;
}

x = fce('a');          // x = 'a';
```

```
char fce( char a )
{
    static char b = 0;          // uloží se do statické paměti,
    b++;                        // po ukončení volání zůstane v pameti
}
```

External a řetězce

External

```
extern char fce( char a );
extern char pole[];
```

Řetězce

```
char pole_1[10];          // pole délky 10, lze uložit až 9 znaků, poslední 0
char *pole_2 = "abcdef";  // inicializované pole pro 6 znaků a na konci 0
char pole_3[7] = "abcdef";
```

Výstupy

```
printf( formát, seznam argumentů);
```

formát

%c	jeden znak (char)
%d	celé číslo dekadicky (short, int)
%nd	celé číslo dekadicky, n míst
%x	celé číslo hexadecimálně
%nx	celé číslo hexadecimálně, n míst
%nX	celé číslo hexadecimálně s nevýznamými nulami, n míst
%n.mf	reálné číslo, n míst a m míst za desetinnou tečkou
%l	celé číslo (long)
%g	reálné číslo (float)
%f	reálné číslo (float)
%lf	reálné číslo (double)
%s	řetězec

```
printf("Pocet novych studentu %5d", n);
```

Řídicí znaky

\n	nová řádka
\r	posun na další řádek
\t	horizontální tabulátor
\0	znak s kódem nula

Makefile

```
OBJ = client.o common.o
BIN = Lode
CC = gcc
LINK = gcc

$(BIN): $(OBJ)
        $(LINK) -lpthread $(OBJ) -o client

client.o: client.c client.h
        $(CC) -c client.c

common.o: common.c common.h
        $(CC) -c common.c

clean:
   rm -f *.o client *~ *.bak
```

Makefile

```
.c.o:
        gcc -Wall -c $<

LIBS=-lsocket -lnsl

all:    client server

client: client.o common.o
        gcc -o client client.o common.o ${LIBS}


server: server.o common.o
        gcc -o server server.o common.o ${LIBS}

clean:
        rm –f *~ *.o server client
```