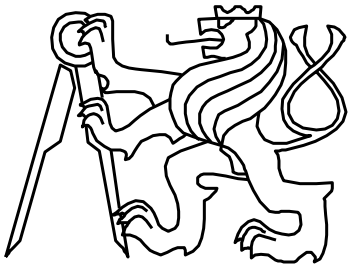
Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.
Praha & EU: Investujeme do vaší budoucnosti

Ukazatele



BI-PA1 Programování a algoritmizace 1, ZS 2012-2013
Katedra teoretické informatiky
© Miroslav Balík
Fakulta informačních technologií
České vysoké učení technické

Ukazatele

- proměnná typu ukazatel
- operace reference
- operace dereference
- kompatibilita ukazatelů
- ukazatelová aritmetika
- pole a ukazatele
- dynamické proměnné
- dealokace dynamických proměnných

Paměť počítače

- Víte, jak velkou paměť má váš počítač?
  - *Ovládací panely/Systém*
- Víte, kolik je 1KiB,1kB, 1MB, 1GiB(ČSN IEC 60027-2, 1998)?

Jednotka	Značka	B	kB	KiB
Kilobyte	kB	1000	1	~0,9766
Kibibyte	KiB	1024	1,024	1
Megabyte	MB	1 000 000	1000	~976,6
Mebibyte	MiB	1 048 576	~1048,6	1024
Gigabyte	GB	10^9	1 000 000	976 562,5
Gibibyte	GiB	2^{30} bytů	~1 073 742	1 048 576

prog8-mocniny2.c

- Jak vypsat adresy proměnných?
  - prog8-adresy.c

Ukazatele

- Obsah proměnné typu *ukazatel na T* se chápe jako adresa proměnné typu *T* (známe již parametry typu *ukazatel*)
- Proměnnou *p* typu *ukazatel na T* zavedeme deklarací
  T *p;
- Tuto proměnnou můžeme inicializovat ukazatelem na proměnnou *x* typu *int* (adresou proměnné *x*)
  T *p = &x;

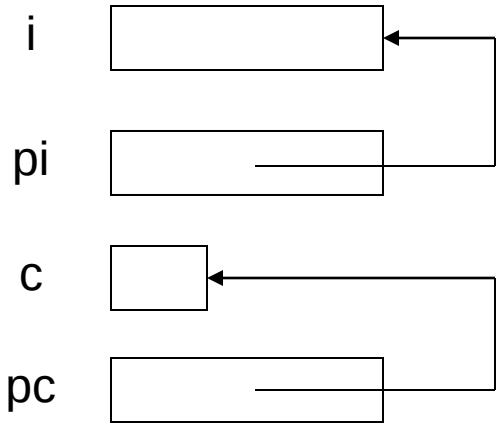
  nebo ji přiřadit ukazatel na proměnnou *x* typu *int* (adresu proměnné *x*)
  p = &x;
- Unární prefixový operátor & označuje operaci *reference*
- Je-li *X* proměnná typu *T*, pak výsledkem operace &*X* je ukazatel na proměnnou *X* (adresa proměnné *X*) a je typu *T**, tzn. ukazatel na *T*
- Příklad (prog8-ukazatele1.c)

```
int i, *pi = &i;
char c, *pc;
pc = &c;
```

Ukazatele

- Nebudeme se zabývat číselnými adresami
- Skutečnost, že proměnná *p* typu ukazatel obsahuje adresu proměnné *x* vyjádříme šipkou z *p* na *x*
- Předchozí příklad ještě jednou:

```
int i, *pi = &i;
char c, *pc;
pc = &c;
```

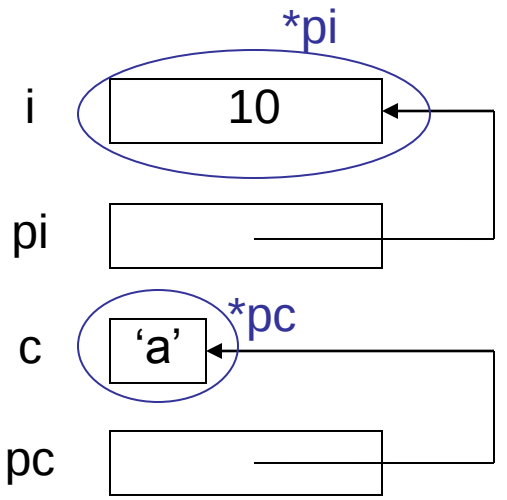



```
int i, *pi = &i;
char c, *pc;
*pc = &c;
```

Dereference

- Chceme-li použít proměnnou, na kterou ukazuje ukazatel, vyjádříme to pomocí unárního operátoru *dereference* *
- Je-li *X* ukazatel typu *T**, pak **X* označuje proměnnou typu *T,* na kterou ukazuje ukazatel *X*
- Příklad (prog8-ukazatele2.c)

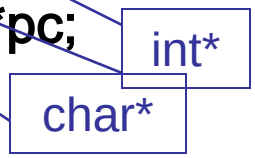
```
int i, *pi = &i;
char c, *pc;
pc = &c;

*pi = 10;

*pc = 'a';
```



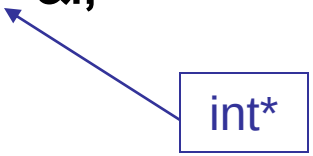
Přiřazení ukazatelů

- Pro typ ukazatel na *T* (*T**) typ *T* nazýváme *doménovým typem ukazatele*
- Kompatibilními typy ukazatel jsou takové, které mají stejný doménový typ
- Proměnné typu ukazatel je třeba přiřadit kompatibilní ukazatel

```
int i, *pi;
char c, *pc;
```




```
pi = &i;                /* kompatibilni typy */
```




```
pi = &c;                /* nekompatibilni typy */
```

- Přiřazení nekompatibilního ukazatele je v C pouze varovné hlášení

Přiřazení ukazatelů

- Co může způsobit, přiřadíme-li proměnné nekompatibilní ukazatel

```
int i, *pi;
char c, *pc;

pi = &c; /* promenna typu int* ukazuje na
            promennou typu char */
pc = &i; /* promenna typu char* ukazuje na
            promennou typu int */

*pi = 0; /* vynuluji se 4 byty pocinaje adresou ulozenou
            v pi */

*pc = 0; /* vynuluje se byte na adrese ulozene v pc */
```

viz příklad prog8-ukazatele3.c

Přiřazení ukazatelů, poznámka

- Uložení čísla int **0x4A 3B 2C 1D** v paměti počítače:

Little endian (Intel)

1D 2C 3B 4A (adresy rostou doprava)

Big endian (Motorola, SPARC, IBM)

4A 3B 2C 1D

Middle-endian (Mixed endian)

3B 4A 1D 2C

nebo

2C 1D 4A 3B

Bi-endian (umí přepínat)

ARM, PowerPC, Alpha, SPARC V9, MIPS, PA-RISC and IA64)

NULL ukazatel

- Jako „prázdný, neplatný“ ukazatel, který neukazuje na žádnou proměnnou, slouží v jazyku C hodnota 0
- Symbolickým označením tohoto ukazatele je *NULL*
- Dereference prázdného ukazatele způsobí chybu při běhu programu
- Příklad prog8-ukazatele4.c:

```
int main(void) {
int *p = NULL;
  *p = 10;
  system(”PAUSE”);
  return 0;
}
```

Ukazatelová aritmetika

- Ukazatelé mohou být operandy sčítání, odčítání a všech relačních operátorů
- Dovolené kombinace a typ výsledku:

  *T** + int -> *T**

  *T** - int -> *T**

  *T** - *T** -> int

  *T** *relop* *T** -> int

- Přičtení *n* k ukazateli typu *T** znamená jeho změnu o *n*-násobek délky typu *T*, podobně odečtení a rozdíl ukazatelů

```
int a[10], *p = &a[0];
*(p + 3) = 10;          /* do a[3] se uloží 10 */
/* vynulování pole a */
for (p = &a[0]; p <= &a[9]; p++) *p = 0;
/* nebo */
for (p = &a[0]; p <= &a[9]; *p++ = 0);
```

Pole a ukazatele

- Jméno pole prvků typu *T* = konstantní ukazatel typu *T** ukazující na prvek s indexem 0

```
int a[10], *pa, i;
pa = a;              totéž co  pa = &a[0]
*(a + 2) = 3;        totéž co  a[2] = 3
*(a + i) = 4;        totéž co  a[i] = 4;

for (pa = a; pa <= a + 9; *pa++ = 0);
a++;          Chyba (proč?)
```

- Upřesnění indexace:

  X [*Y*] ,

  kde první výraz je typu ukazatel na *T* a druhý typu *int*, výsledek je typu *T*

- Výraz *X* [*Y*] je ekvivalentní s *((*X*) + (*Y*))

```
pa[3] = 10;          totéž co *(pa + 3) = 10;
```

Pole a ukazatele

- Podívejte se na příklad prog8-vektory1.c
  - skalární součin dvou vektorů, parametry funkcí specifikovány jako ukazatele

```
...
void ctiVektor(int *v, int n) {
  int i;
  printf “zadejte %d celych cisel\n”, n);
  for (i=0; i<n; i++)
     scanf(” %d” , &v[i]);
}
```

Pole a ukazatele

- Podívejte se na příklad prog8-vektory2.c
  - skalární součin dvou vektorů, průchody polem pomocí ukazatelové aritmetiky

```
void ctiVektor(int v[], int n) {
  int *p, *pn = v+n;
  printf("zadejte %d celych cisel\n", n);
  for (p=v; p<pn; p++)
     scanf("%d", p);
}
```

```
void ctiVektor(int *v, int n) {
 int i;
 printf("zadejte %d celych cisel\n", n);
 for (i=0; i<n; i++)
     scanf("%d", &v[i]);
}
```

Řetězce a ukazatele - PALINDROM

- Palindrom je alespoň dvojznakové slovo, které vychází stejně čteno odpředu i odzadu, např. kajak
- Pro zjištění, zda slovo je palindrom, vyvedeme funkci

```
#define MAXDELKA 100
int jePalindrom(char *s);
int main(void) {
  char slovo[MAXDELKA+1];
  printf("zadejte slovo obsahujici nanejvys %d znaku: ",  MAXDELKA);
  scanf("%s", slovo);
  printf("zadali jste %s\n", slovo);
  printf("toto slovo");
  if (jePalindrom(slovo)) printf("je");
  else printf("neni");
  printf(" palindrom\n");
  ...
}
```

nepochopen
nepotopen
Dne moto: Palindrom i spáchá psí mord, Nil a potom End.

Řetězce a ukazatele

- Jak zjistit, že slovo je palindrom
  - slovo délky *d* bude uloženo v poli *s* počínaje indexem 0 a konče indexem *d*-1 (hodnotou prvku s indexem *d* bude závěrečná nula)
  - aby slovo bylo palindrom, musí platit:

    s[0] == s[d-1]

    s[1] == s[d-2]

    ...
  - test na rovnost skončíme, až dosáhneme poloviny délky slova
- Řešení (prog8-palindrom1.c):

```
int jePalindrom(char *s) {
  int delka = strlen(s), polovina = delka/2, i;
  for (i=0; i<polovina; i++)
      if (s[i]!=s[delka-i-1]) return 0;
  return 1;
}
```

Řetězce a ukazatele

- Testy rovnosti znaků můžeme provádět pomocí dvou proměnných typu ukazatel na *char*, z nichž první nastavíme na první znak a budeme ji zvětšovat a druhou nastavíme na poslední znak a budeme ji zmenšovat
- Test budeme opakovat, pokud první ukazatel bude menší než druhý ukazatel
- Řešení (prog8-palindrom2.c):

```
int jePalindrom(char *s) {
  char *p, *q;
  for (p=s, q=s+strlen(s)-1; p<q; p++,q--)
     if (*p != *q) return 0;
  return 1;
}
```

```
int jePalindrom(char *s) {
  char p=s, q=s+strlen(s)-1;
  while (p<q)
     if (*p++ != *q--) return 0;
  return 1;
}
```

Řetězce a ukazatele

- A něco na závěr řetězců a ukazatelů
- Literál tvořený posloupností znaků, která je uzavřena do uvozovek, je typu *char** (připomeňme, že v paměti je reprezentován posloupností znaků zakončenou nulovým bytem)
- Podívejte se na příklad prog8-palindrom3.c

```
#define  SLOVO "kajak"
int jePalindrom(char *s);

int main(void) {
  printf("slovo %s", SLOVO);
  if (jePalindrom(SLOVO)) printf("je");
  else printf("neni");
  printf(" palindrom\n");
  jePalindrom("abcd");
  return 0;
}
```

Dynamické proměnné

- Ukazatele v jazyku C slouží především pro:
  - předávání výstupních parametrů procedurám a funkcím
  - předávání pole jako parametru procedurám a funkcím
  - přístup k dynamickým proměnným
- Dynamická proměnná není zavedena deklarací, ale je vytvořena speciální funkcí (operátorem, příkazem)
- Dynamická proměnná nemá jméno, a proto k ní můžeme přistupovat pouze pomocí ukazatele (adresy)
- V jazyku C vytvoříme dynamickou proměnnou pomocí funkce *malloc*
  - parametrem funkce je počet bytů, které budou vnitřní reprezentací proměnné
  - funkce „zařídí“, že ve volné paměti je rezervováno místo pro proměnnou s danou velikostí vnitřní reprezentace a jejím výsledkem je adresa tohoto místa (tzn. ukazatel na dynamicky vytvořenou proměnnou)
  - funkce je deklarována tak, že vrací výsledek typu *void**; volání funkce je proto třeba přetypovat

Dynamické proměnné

- Příklad (prog8-dynprom.c)

```
int main(void) {
  int *p;

  p = (int *)malloc(sizeof(int));
 *p = 10;
  printf("*p=%d\n", *p);
  system("PAUSE");
  return 0;
}
```

přetypování na typ *int**

- Jak je to v paměti počítače

Dynamicky vytvořené pole

- Dynamicky vytvořené proměnné jednoduchých typů obvykle nejsou třeba
- Užitečné je dynamicky vytvořené pole
- Příklad: skalární součin dvou vektorů, počet složek je dán vstupními daty
  - funkce *ctiVektor* dynamicky vytvoří pole, jehož délka je dána parametrem, a přečte jeho prvky z klávesnice

```
int *ctiVektor(int n) {
  int i, *p;
  p = (int *)malloc(sizeof(int)*n);
  printf("zadejte %d celych cisel\n", n);
  for (i=0; i<n; i++)
    scanf("%d", &p[i]);
  return p;
}
```

- Ve funkci *main* nebudou deklarována pole, ale proměnné typu ukazatel, do nichž se uloží adresy polí dynamicky vytvořených funkcí *ctiVektor*

Dynamicky vytvořené pole

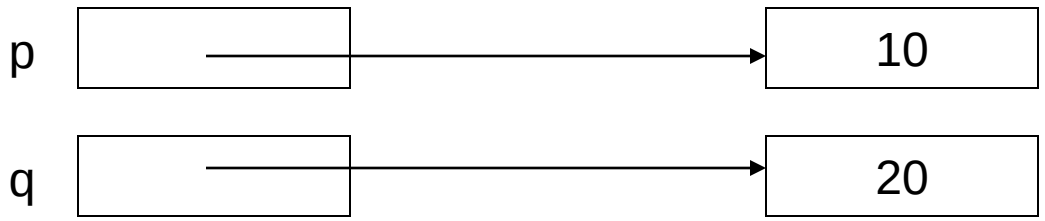
- prog8-dynpole.c

```
int ctiInt(int min, int max) {...}
int *ctiVektor(int n) {...}
int skalarniSoucin(int x[], int y[], int n) {...}
int main(void) {
  int *x, *y, n;
  printf("zadejte pocet slozek vektoru: ");
  n = ctiInt(1, INT_MAX);
  printf("vektor x\n");
  x = ctiVektor(n);
  printf("vektor y\n");
  y = ctiVektor(n);
  printf("skalarni soucin vektoru x a y je %d\n",
            skalarniSoucin(x, y, n));
  return 0;
}
```

Dealokace

- Příklad:

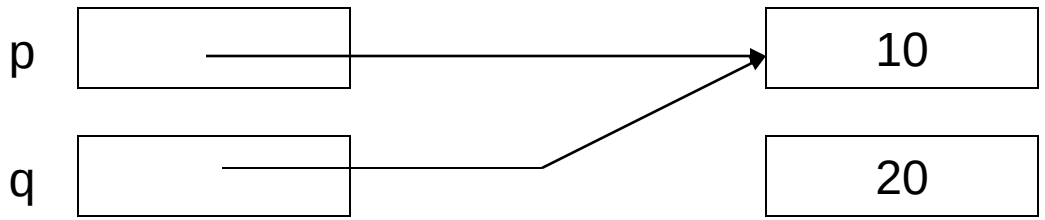
```
int *p, *q;
p = (int *)malloc(sizeof(int));
q = (int *)malloc(sizeof(int));
*p = 10;
*q = 20;
```



Dealokace

- Příklad:

```
int *p, *q;
p = (int *)malloc(sizeof(int));
q = (int *)malloc(sizeof(int));
*p = 10;
*q = 20;
q = p;
```

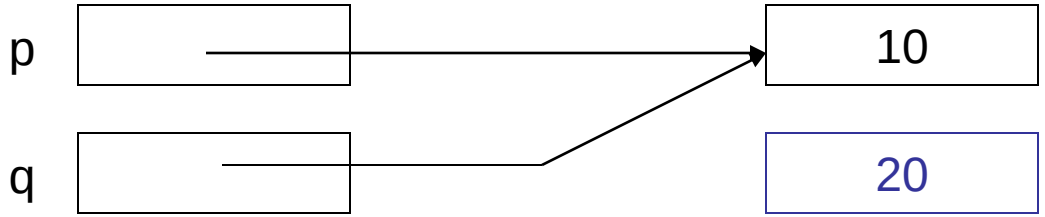


- Co s dynamicky vytvořenou proměnnou s hodnotpu 20, na kterou ukazovala proměnná q (není přístupná)?
- Před ztrátou ukazatele na ni je třeba ji *dealokovat* (vrátit paměť, kterou zabírá, zpět do volné paměti)

Dealokace

- Dealokaci provádí funkce *free*

```
int *p, *q;
p = (int*)malloc(sizeof(int));
q = (int*)malloc(sizeof(int));
*p = 10;
*q = 20;
free(q);
q = p;
```



- Paměť přidělená modře zarámované proměnné se uvolní a je k dispozici pro další *malloc*

Pamět počítače

- program vytvořený překladačem jazyka C využívá pěti úseků paměti:
  - paměť kódu
  - paměť konstant
  - paměť globálnich proměnných
  - zásobník pro přidělení paměti parametrům a lokálním proměnným funkcí
  - paměť pro dynamické proměnné

- Podívejte se na program prog8-pamet.c, který vypíše adresy z těchto úseků