Přetypování

je jednou z nejdůležitějších vlastností C

- `getchar` vrací `int`, ale my chceme `char`.

Přetypování

je jednou z nejdůležitějších vlastností C

- `getchar` vrací `int`, ale my chceme `char`.
- `char a=getchar();` může udělat warning.

Přetypování

je jednou z nejdůležitějších vlastností C

- `getchar` vrací `int`, ale my chceme `char`.
- `char a=getchar();` může udělat warning.
- `char a=(char)getchar();` už warning neudělá.

Přetypování

je jednou z nejdůležitějších vlastností C

- `getchar` vrací `int`, ale my chceme `char`.
- `char a=getchar();` může udělat warning.
- `char a=(char)getchar();` už warning neudělá.
- `cil=(cilovy_typ)puvodni_promenna;`

Přetypování

je jednou z nejdůležitějších vlastností C

- `getchar` vrací `int`, ale my chceme `char`.
- `char a=getchar();` může udělat warning.
- `char a=(char)getchar();` už warning neudělá.
- `cil=(cilovy_typ)puvodni_promenna;`
- `int x=(int)"ahoj"; x*=2;`

Přetypování

je jednou z nejdůležitějších vlastností C

- `getchar` vrací `int`, ale my chceme `char`.
- `char a=getchar();` může udělat warning.
- `char a=(char)getchar();` už warning neudělá.
- `cil=(cilovy_typ)puvodni_promenna;`
- `int x=(int)"ahoj"; x*=2;`
- Přetypování je dobrý sluha, ale špatný pán.

Pole

a pointery a vztahy mezi poli a pointery

- Vždy indexujeme od nuly!

Pole

a pointery a vztahy mezi poli a pointery

- Vždy indexujeme od nuly!
- Tudíž stačí říct počet prvků (jedno číslo):
  `int a[10];`

Pole

a pointery a vztahy mezi poli a pointery

- Vždy indexujeme od nuly!
- Tudíž stačí říct počet prvků (jedno číslo):
  `int a[10];`
- S polem pracujeme jako v Pythonu:

Pole

a pointery a vztahy mezi poli a pointery

- Vždy indexujeme od nuly!
- Tudíž stačí říct počet prvků (jedno číslo):
  `int a[10];`
- S polem pracujeme jako v Pythonu:
- `for(int i=0;i++<10;a[i-1]=i);`

Reprezentace pole

a jeho souvislost s pointery

- Pole je reprezentováno ukazatelem na začátek,

Reprezentace pole

a jeho souvislost s pointery

- Pole je reprezentováno ukazatelem na začátek,
- proto lze s polem pracovat jako s pointerem (a obráceně).

Reprezentace pole

a jeho souvislost s pointery

- Pole je reprezentováno ukazatelem na začátek,
- proto lze s polem pracovat jako s pointerem (a obráceně).
- Ukazatele v C jsou manuálně obsluhovaný ekvivalent referencí v Pythonu.

Reprezentace pole

a jeho souvislost s pointery

- Pole je reprezentováno ukazatelem na začátek,
- proto lze s polem pracovat jako s pointerem (a obráceně).
- Ukazatele v C jsou manuálně obsluhovaný ekvivalent referencí v Pythonu.
- Python: `prvek.next.....`

Reprezentace pole

a jeho souvislost s pointery

- Pole je reprezentováno ukazatelem na začátek,
- proto lze s polem pracovat jako s pointerem (a obráceně).
- Ukazatele v C jsou manuálně obsluhovaný ekvivalent referencí v Pythonu.
- Python: `prvek.next.....`
- Dereference: unární prefixní hvězdička (koukni, co je pod pointerem)

Reprezentace pole

a jeho souvislost s pointery

- Pole je reprezentováno ukazatelem na začátek,
- proto lze s polem pracovat jako s pointerem (a obráceně).
- Ukazatele v C jsou manuálně obsluhovaný ekvivalent referencí v Pythonu.
- Python: `prvek.next.....`
- Dereference: unární prefixní hvězdička (koukni, co je pod pointerem)
- Mimochodem operátor přistoupení do struktury je stejný (ale o tom později).

Reprezentace pole

a jeho souvislost s pointery

- Pole je reprezentováno ukazatelem na začátek,
- proto lze s polem pracovat jako s pointerem (a obráceně).
- Ukazatele v C jsou manuálně obsluhovaný ekvivalent referencí v Pythonu.
- Python: `prvek.next.....`
- Dereference: unární prefixní hvězdička (koukni, co je pod pointerem)
- Mimochodem operátor přistoupení do struktury je stejný (ale o tom později).
- Definice pointeru také unární pref. hvězdičkou: `char*a="ahoj";`

Reprezentace pole

a jeho souvislost s pointery

- Pole je reprezentováno ukazatelem na začátek,
- proto lze s polem pracovat jako s pointerem (a obráceně).
- Ukazatele v C jsou manuálně obsluhovaný ekvivalent referencí v Pythonu.
- Python: `prvek.next.....`
- Dereference: unární prefixní hvězdička (koukni, co je pod pointerem)
- Mimochodem operátor přistoupení do struktury je stejný (ale o tom později).
- Definice pointeru také unární pref. hvězdičkou: `char*a="ahoj";`
- V C nejsou stringy, místo nich používáme (`char*`), tedy řetězce se zde chovají jako pole znaků.

Specifika pointerů

- Lze s nimi pracovat jako s poli:

```
char*a="ahoj";
putchar(a[2]); // vypise o (proc?)
```

Specifika pointerů

- Lze s nimi pracovat jako s poli:

```
char*a="ahoj";
putchar(a[2]); // vypise o (proc?)
```

- Parametry funkcí se v C předávají vždy hodnotou.

Specifika pointerů

- Lze s nimi pracovat jako s poli:

```
char*a="ahoj";
putchar(a[2]); // vypise o (proc?)
```

- Parametry funkcí se v C předávají vždy hodnotou.
- Chceme-li v C předání referencí, použijeme pointer.

Specifika pointerů

- Lze s nimi pracovat jako s poli:
  ```
  char*a="ahoj";
  putchar(a[2]); // vypise o (proc?)
  ```
- Parametry funkcí se v C předávají vždy hodnotou.
- Chceme-li v C předání referencí, použijeme pointer.
- Vzetí pointeru – operátor & (unární prefixní):
  ```
  int j=0; int*i=&j;
  ```

Specifika pointerů

- Lze s nimi pracovat jako s poli:
  ```
  char*a="ahoj";
  putchar(a[2]); // vypise o (proc?)
  ```
- Parametry funkcí se v C předávají vždy hodnotou.
- Chceme-li v C předání referencí, použijeme pointer.
- Vzetí pointeru – operátor & (unární prefixní):
  ```
  int j=0; int*i=&j;
  ```
- V C++ jsou i reference.

Specifika pointerů

- Lze s nimi pracovat jako s poli:
  ```
  char*a="ahoj";
  putchar(a[2]); // vypise o (proc?)
  ```
- Parametry funkcí se v C předávají vždy hodnotou.
- Chceme-li v C předání referencí, použijeme pointer.
- Vzetí pointeru – operátor & (unární prefixní):
  ```
  int j=0; int*i=&j;
  ```
- V C++ jsou i reference.
- ((char*)2)[(int)a] == a[2]

Specifika pointerů

a jejich netriviální využití

- Pointery lze využít třeba k návrhu spojových seznamů, nebo také jinak.

Specifika pointerů

a jejich netriviální využití

- Pointery lze využít třeba k návrhu spojových seznamů, nebo také jinak.
- Jak okopírovat řetězec?

Specifika pointerů

a jejich netriviální využití

- Pointery lze využít třeba k návrhu spojových seznamů, nebo také jinak.
- Jak okopírovat řetězec?
- V C `a=b` okopíruje pointer (jako v Pythonu).

Specifika pointerů

a jejich netriviální využití

- Pointery lze využít třeba k návrhu spojových seznamů, nebo také jinak.
- Jak okopírovat řetězec?
- V C `a=b` okopíruje pointer (jako v Pythonu).
- `while(*a++=*b++);`

Specifika pointerů

a jejich netriviální využití

- Pointery lze využít třeba k návrhu spojových seznamů, nebo také jinak.
- Jak okopírovat řetězec?
- V C `a=b` okopíruje pointer (jako v Pythonu).
- `while(*a++=*b++);`
- Proměnná `a` musí být vhodně naalokovaná.

Specifika pointerů

a jejich netriviální využití

- Pointery lze využít třeba k návrhu spojových seznamů, nebo také jinak.
- Jak okopírovat řetězec?
- V C `a=b` okopíruje pointer (jako v Pythonu).
- `while(*a++=*b++);`
- Proměnná `a` musí být vhodně naalokovaná.
- `char *strcpy(char *dest, const char *src);`

Allokace, deallokace

pro účely práce se stringy

- Řízeno souborem `stdlib.h`

Allokace, deallokace

pro účely práce se stringy

- Řízeno souborem `stdlib.h`
- `void* malloc(size_t kolik);` naalokuje,

Allokace, deallokace

pro účely práce se stringy

- Řízeno souborem `stdlib.h`
- `void* malloc(size_t kolik);` naalokuje,
- musíme říct velikost v bytech. My ale nevíme mnoho o velikostech datových typů.

Allokace, deallokace

pro účely práce se stringy

- Řízeno souborem `stdlib.h`
- `void* malloc(size_t kolik);` naalokuje,
- musíme říct velikost v bytech. My ale nevíme mnoho o velikostech datových typů.
- Proto makro `sizeof(typ)`, nicméně velikost charu je dle normy 1.

Allokace, deallokace

pro účely práce se stringy

- Řízeno souborem `stdlib.h`
- `void* malloc(size_t kolik);` naalokuje,
- musíme říct velikost v bytech. My ale nevíme mnoho o velikostech datových typů.
- Proto makro `sizeof(typ)`, nicméně velikost charu je dle normy 1.
- `char* retezec=malloc(10);` vytvoří řetězec délky 9,...

Allokace, deallokace

pro účely práce se stringy

- Řízeno souborem `stdlib.h`
- `void* malloc(size_t kolik);` naalokuje,
- musíme říct velikost v bytech. My ale nevíme mnoho o velikostech datových typů.
- Proto makro `sizeof(typ)`, nicméně velikost charu je dle normy 1.
- `char* retezec=malloc(10);` vytvoří řetězec délky 9,...
- protože řetězce jsou ukončeny znakem 0: `retezec[9]=0;//abychom nepretekli`

Allokace, deallokace

pro účely práce se stringy

- Řízeno souborem `stdlib.h`
- `void* malloc(size_t kolik);` naalokuje,
- musíme říct velikost v bytech. My ale nevíme mnoho o velikostech datových typů.
- Proto makro `sizeof(typ)`, nicméně velikost charu je dle normy 1.
- `char* retezec=malloc(10);` vytvoří řetězec délky 9,...
- protože řetězce jsou ukončeny znakem 0: `retezec[9]=0;//abychom nepretekli`
- Funkce `malloc` paměť neinicializuje.

Allokace, deallokace

pro účely práce se stringy

- Řízeno souborem `stdlib.h`
- `void* malloc(size_t kolik);` naalokuje,
- musíme říct velikost v bytech. My ale nevíme mnoho o velikostech datových typů.
- Proto makro `sizeof(typ)`, nicméně velikost charu je dle normy 1.
- `char* retezec=malloc(10);` vytvoří řetězec délky 9,...
- protože řetězce jsou ukončeny znakem 0: `retezec[9]=0;//abychom nepretekli`
- Funkce `malloc` paměť neinicializuje.
- Dealokace: `void free (void*);`

Allokace, deallokace

pro účely práce se stringy

- Řízeno souborem `stdlib.h`
- `void* malloc(size_t kolik);` naalokuje,
- musíme říct velikost v bytech. My ale nevíme mnoho o velikostech datových typů.
- Proto makro `sizeof(typ)`, nicméně velikost charu je dle normy 1.
- `char* retezec=malloc(10);` vytvoří řetězec délky 9,...
- protože řetězce jsou ukončeny znakem 0: `retezec[9]=0;//abychom nepretekli`
- Funkce `malloc` paměť neinicializuje.
- Dealokace: `void free (void*);`
- Další funkce `realloc, calloc`.

Znovu kopírování stringů

jako by snad bylo zakleté

- Jak naalokovat místo pro kopírovaný string?

Znovu kopírování stringů

jako by snad bylo zakleté

- Jak naalokovat místo pro kopírovaný string?
- Funkce `strlen` určí délku.

Znovu kopírování stringů

jako by snad bylo zakleté

- Jak naalokovat místo pro kopírovaný string?
- Funkce `strlen` určí délku.
- Na konec musíme přidat znak číslo 0!

Znovu kopírování stringů

jako by snad bylo zakleté

- Jak naalokovat místo pro kopírovaný string?
- Funkce `strlen` určí délku.
- Na konec musíme přidat znak číslo 0!
- Tedy: `char*b=malloc(strlen(a)+1);`...

Znovu kopírování stringů

jako by snad bylo zakleté

- Jak naalokovat místo pro kopírovaný string?
- Funkce `strlen` určí délku.
- Na konec musíme přidat znak číslo 0!
- Tedy: `char*b=malloc(strlen(a)+1);`...
- pak ještě musíme udělat `char*ret=b;`...

Znovu kopírování stringů

jako by snad bylo zakleté

- Jak naalokovat místo pro kopírovaný string?
- Funkce `strlen` určí délku.
- Na konec musíme přidat znak číslo 0!
- Tedy: `char*b=malloc(strlen(a)+1);`...
- pak ještě musíme udělat `char*ret=b;`...
- a až pak můžeme spustit teror z předchozího slidu.

Znovu kopírování stringů

jako by snad bylo zakleté

- Jak naalokovat místo pro kopírovaný string?
- Funkce `strlen` určí délku.
- Na konec musíme přidat znak číslo 0!
- Tedy: `char*b=malloc(strlen(a)+1);`...
- pak ještě musíme udělat `char*ret=b;`...
- a až pak můžeme spustit teror z předchozího slidu.
- v `string.h` plno funkcí jako `strcmp`, `strncmp`, `strncpy`...

Funkce printf

bere nepevný počet argumentů

- `int printf(char format[],...);`

Funkce printf

bere nepevný počet argumentů

- `int printf(char format[],...);`
- Ve formátovacím řetězci mohou být escapové sekvence (`\n`, `\r`, `\t`),

Funkce printf

bere nepevný počet argumentů

- `int printf(char format[],...);`
- Ve formátovacím řetězci mohou být escapové sekvence (`\n`, `\r`, `\t`),
- odkazy k dalším argumentům pomocí procentítka

Funkce printf

bere nepevný počet argumentů

- `int printf(char format[],...);`
- Ve formátovacím řetězci mohou být escapové sekvence (`\n`, `\r`, `\t`),
- odkazy k dalším argumentům pomocí procentítka
- například `printf("%d %s %f", 10,"ahoj",3.14);`

Funkce printf

bere nepevný počet argumentů

- `int printf(char format[],...);`
- Ve formátovacím řetězci mohou být escapové sekvence (`\n`, `\r`, `\t`),
- odkazy k dalším argumentům pomocí procentítka
- například `printf("%d %s %f", 10,"ahoj",3.14);`
- viz `http://www.cplusplus.com/reference/cstdio/printf/`

Struktury a unie

další věc částečně známá

- V Pythonu se používaly objekty, které od sebe mohly dědit.

Struktury a unie

další věc částečně známá

- V Pythonu se používaly objekty, které od sebe mohly dědit.
- V C jsou struktury `struct jmeno {obsah}`

Struktury a unie

další věc částečně známá

- V Pythonu se používaly objekty, které od sebe mohly dědit.
- V C jsou struktury `struct jmeno {obsah}`
- a unie `union nazev{vnitrek}`.

Struktury a unie

další věc částečně známá

- V Pythonu se používaly objekty, které od sebe mohly dědit.
- V C jsou struktury `struct jmeno {obsah}`
- a unie `union nazev{vnitrek}`.
- Použití (definované) struktury:

```
struct spojak * hlava;
```

Struktury a unie

další věc částečně známá

- V Pythonu se používaly objekty, které od sebe mohly dědit.
- V C jsou struktury `struct jmeno {obsah}`
- a unie `union nazev{vnitrek}`.
- Použití (definované) struktury:
  `struct spojak * hlava;`
- Prvky struktury jsou v paměti reprezentované za sebou, prvky unie přes sebe.

Struktury a unie

další věc částečně známá

- V Pythonu se používaly objekty, které od sebe mohly dědit.
- V C jsou struktury `struct jmeno {obsah}`
- a unie `union nazev{vnitrek}`.
- Použití (definované) struktury:
  `struct spojak * hlava;`
- Prvky struktury jsou v paměti reprezentované za sebou, prvky unie přes sebe.
- Protože je otrava pořád psát `struct spojak`, můžeme definovat vlastní typ:
  `typedef int integer;`

Struktury a unie

další věc částečně známá

- V Pythonu se používaly objekty, které od sebe mohly dědit.
- V C jsou struktury `struct jmeno {obsah}`
- a unie `union nazev{vnitrek}`.
- Použití (definované) struktury:
  `struct spojak * hlava;`
- Prvky struktury jsou v paměti reprezentované za sebou, prvky unie přes sebe.
- Protože je otrava pořád psát `struct spojak`, můžeme definovat vlastní typ:
  `typedef int integer;`
- anebo `typedef struct pom_spojak{int hod; struct pom_spojak*next;} spojak;`

Struktury a unie

další věc částečně známá

- V Pythonu se používaly objekty, které od sebe mohly dědit.
- V C jsou struktury `struct jmeno {obsah}`
- a unie `union nazev{vnitrek}`.
- Použití (definované) struktury:
  `struct spojak * hlava;`
- Prvky struktury jsou v paměti reprezentované za sebou, prvky unie přes sebe.
- Protože je otrava pořád psát `struct spojak`, můžeme definovat vlastní typ:
  `typedef int integer;`
- anebo `typedef struct pom_spojak{int hod; struct pom_spojak*next;} spojak;`
- A pak: `spojak hlava;`

Přístup do spojáku

a zatracené priority operátorů

- Do struktury se přistupuje jako v Pythonu (operátor tečky).

Přístup do spojáku

a zatracené priority operátorů

- Do struktury se přistupuje jako v Pythonu (operátor tečky).
- Pointer se dereferencuje (unární prefixní) hvězdičkou.

Přístup do spojáku

a zatracené priority operátorů

- Do struktury se přistupuje jako v Pythonu (operátor tečky).
- Pointer se dereferencuje (unární prefixní) hvězdičkou.
- ale ve výrazu `*a.hod` má vyšší prioritu tečka (než hvězdička).

Přístup do spojáku

a zatracené priority operátorů

- Do struktury se přistupuje jako v Pythonu (operátor tečky).
- Pointer se dereferencuje (unární prefixní) hvězdičkou.
- ale ve výrazu `*a.hod` má vyšší prioritu tečka (než hvězdička).
- Takže `(*a).hod`...

Přístup do spojáku

a zatracené priority operátorů

- Do struktury se přistupuje jako v Pythonu (operátor tečky).
- Pointer se dereferencuje (unární prefixní) hvězdičkou.
- ale ve výrazu `*a.hod` má vyšší prioritu tečka (než hvězdička).
- Takže `(*a).hod`...
- ... nebo `a->hod`.

Zde udělat příklad na spoják

Definovat strukturu a napsat funkce `pridej` a `uber`.

Práce se soubory

je také podobná jako v Pythonu, jenom to není objektové a ty funkce se jmenují jinak

- Pomocí `stdio.h`

Práce se soubory

je také podobná jako v Pythonu, jenom to není objektové a ty funkce se jmenují jinak

- Pomocí `stdio.h`
- Místo proměnné typu `file` použijme:
  `FILE * soubor;`

Práce se soubory

je také podobná jako v Pythonu, jenom to není objektové a ty funkce se jmenují jinak

- Pomocí `stdio.h`
- Místo proměnné typu `file` použijme:
  `FILE * soubor;`
- Soubor rovnou otevřeme:
  `soubor=fopen("jmeno","rezim");`

Práce se soubory

je také podobná jako v Pythonu, jenom to není objektové a ty funkce se jmenují jinak

- Pomocí `stdio.h`
- Místo proměnné typu `file` použijme:
  `FILE * soubor;`
- Soubor rovnou otevřeme:
  `soubor=fopen("jmeno","rezim");`
- Režim může být zejména: `r`, `w`, `a`, `r+`, `w+`, `a+`

Práce se soubory

je také podobná jako v Pythonu, jenom to není objektové a ty funkce se jmenují jinak

- Pomocí `stdio.h`
- Místo proměnné typu `file` použijme:
  `FILE * soubor;`
- Soubor rovnou otevřeme:
  `soubor=fopen("jmeno","rezim");`
- Režim může být zejména: `r, w, a, r+, w+, a+`
- Použijeme "a" nebo "a+" chceme-li volat `fseek, fsetpos, rewind`.

Práce se soubory

je také podobná jako v Pythonu, jenom to není objektové a ty funkce se jmenují jinak

- Pomocí `stdio.h`
- Místo proměnné typu `file` použijme:
  `FILE * soubor;`
- Soubor rovnou otevřeme:
  `soubor=fopen("jmeno","rezim");`
- Režim může být zejména: `r, w, a, r+, w+, a+`
- Použijeme "a" nebo "a+" chceme-li volat `fseek, fsetpos, rewind`.
- Chceme-li soubor v binárním režimu, přidáme znak `b`:
  `"rb", "r+b", "rb+"`– binární režim se stará o konce řádků.

Práce se soubory

je také podobná jako v Pythonu, jenom to není objektové a ty funkce se jmenují jinak

- Pomocí `stdio.h`
- Místo proměnné typu `file` použijme:
  `FILE * soubor;`
- Soubor rovnou otevřeme:
  `soubor=fopen("jmeno","rezim");`
- Režim může být zejména: `r, w, a, r+, w+, a+`
- Použijeme "a" nebo "a+" chceme-li volat `fseek, fsetpos, rewind`.
- Chceme-li soubor v binárním režimu, přidáme znak `b`:
  `"rb", "r+b", "rb+"`– binární režim se stará o konce řádků.
- C11 zavádí ještě "x" k režimu "w"...

Práce se soubory

je také podobná jako v Pythonu, jenom to není objektové a ty funkce se jmenují jinak

- Pomocí `stdio.h`
- Místo proměnné typu `file` použijme:
  `FILE * soubor;`
- Soubor rovnou otevřeme:
  `soubor=fopen("jmeno","rezim");`
- Režim může být zejména: `r, w, a, r+, w+, a+`
- Použijeme "a" nebo "a+" chceme-li volat `fseek, fsetpos, rewind`.
- Chceme-li soubor v binárním režimu, přidáme znak `b`:
  `"rb", "r+b", "rb+"`– binární režim se stará o konce řádků.
- C11 zavádí ještě "x" k režimu "w"...
- ... vybouchni, pokud soubor už existuje.

Soubory II

zavřít soubor jde rychleji než otevřít

- `int fclose(FILE*);`
- `int feof(FILE*);`
- `int fgetc(FILE*)`
  `char*fgets(char*s,int pocet,FILE*)`
- `fgetc, fputc, fputs, fprintf`
- `fscanf` formátované načítání (je zrádné).