Vztah programu a operačního systému

Programovací techniky

doc. Ing. Jiří Rybička, Dr.
Ústav informatiky
PEF MENDELU v Brně
rybicka@mendelu.cz

Standardní soubory

- Reprezentovány objekty `cin`, `cout`, `cerr`, `clog`

Standardní soubory

- Reprezentovány objekty `cin`, `cout`, `cerr`, `clog`
- Není třeba deklarovat, otevírat, resetovat...

Standardní soubory

- Reprezentovány objekty `cin`, `cout`, `cerr`, `clog`
- Není třeba deklarovat, otevírat, resetovat...
- Z hlediska OS: přesměrování, kolona

Standardní soubory

- Reprezentovány objekty `cin`, `cout`, `cerr`, `clog`
- Není třeba deklarovat, otevírat, resetovat...
- Z hlediska OS: přesměrování, kolona
- OS zajistí všechny manipulace se soubory na disku

Standardní soubory

- Reprezentovány objekty `cin`, `cout`, `cerr`, `clog`
- Není třeba deklarovat, otevírat, resetovat...
- Z hlediska OS: přesměrování, kolona
- OS zajistí všechny manipulace se soubory na disku
- Specifické odchylky chování při práci s klávesnicí

Standardní soubory

- Reprezentovány objekty `cin`, `cout`, `cerr`, `clog`
- Není třeba deklarovat, otevírat, resetovat...
- Z hlediska OS: přesměrování, kolona
- OS zajistí všechny manipulace se soubory na disku
- Specifické odchylky chování při práci s klávesnicí
- Univerzální použití – efektivní ladění s ostrými daty

Standardní soubory

- Reprezentovány objekty `cin`, `cout`, `cerr`, `clog`
- Není třeba deklarovat, otevírat, resetovat...
- Z hlediska OS: přesměrování, kolona
- OS zajistí všechny manipulace se soubory na disku
- Specifické odchylky chování při práci s klávesnicí
- Univerzální použití – efektivní ladění s ostrými daty
- Možnost uchování výstupů, další zpracování, editace...

Příkazový řádek

- Spuštění programu: cesta, jméno programu, parametry

Příkazový řádek

- Spuštění programu: cesta, jméno programu, parametry
- `../bin/apps/prog 118 ”b+c d” /soubory/data.bin`

Příkazový řádek

- Spuštění programu: cesta, jméno programu, parametry
- `../bin/apps/prog 118 ”b+c d” /soubory/data.bin`
- Parametry jsou vhodným doplňkovým vstupem

Příkazový řádek

- Spuštění programu: cesta, jméno programu, parametry
- `../bin/apps/prog 118 ”b+c d” /soubory/data.bin`
- Parametry jsou vhodným doplňkovým vstupem
- Informace z příkazového řádku: modifikace činnosti programu, konfigurace, zadání prvku, s nímž se má pracovat

Příkazový řádek

- Spuštění programu: cesta, jméno programu, parametry
- `../bin/apps/prog 118 ”b+c d” /soubory/data.bin`
- Parametry jsou vhodným doplňkovým vstupem
- Informace z příkazového řádku: modifikace činnosti programu, konfigurace, zadání prvku, s nímž se má pracovat
- Zjištění počtu zadaných parametrů: první parametr ve funkci main `int main(int pocet...)`

Příkazový řádek

- Spuštění programu: cesta, jméno programu, parametry
- `../bin/apps/prog 118 ”b+c d” /soubory/data.bin`
- Parametry jsou vhodným doplňkovým vstupem
- Informace z příkazového řádku: modifikace činnosti programu, konfigurace, zadání prvku, s nímž se má pracovat
- Zjištění počtu zadaných parametrů: první parametr ve funkci main `int main(int pocet...)`
- Hodnoty všech parametrů: druhý parametr ve funkci main `int main(... char *polepar[])`

Příkazový řádek

- Spuštění programu: cesta, jméno programu, parametry
- `../bin/apps/prog 118 ”b+c d” /soubory/data.bin`
- Parametry jsou vhodným doplňkovým vstupem
- Informace z příkazového řádku: modifikace činnosti programu, konfigurace, zadání prvku, s nímž se má pracovat
- Zjištění počtu zadaných parametrů: první parametr ve funkci main `int main(int pocet...)`
- Hodnoty všech parametrů: druhý parametr ve funkci main `int main(... char *polepar[])`
- cesta a jméno spouštěného programu je chápáno jako první parametr, index 0 v poli parametrů

Příkazový řádek

- Spuštění programu: cesta, jméno programu, parametry
- `../bin/apps/prog 118 ”b+c d” /soubory/data.bin`
- Parametry jsou vhodným doplňkovým vstupem
- Informace z příkazového řádku: modifikace činnosti programu, konfigurace, zadání prvku, s nímž se má pracovat
- Zjištění počtu zadaných parametrů: první parametr ve funkci main `int main(int pocet...)`
- Hodnoty všech parametrů: druhý parametr ve funkci main `int main(... char *polepar[])`
- cesta a jméno spouštěného programu je chápáno jako první parametr, index 0 v poli parametrů
- Všechny parametry jsou řetězcové (tvar C), deklarovány jako `char *Ret` nebo `char Ret[]`

Příklad

- Procedura získá hodnotu jednoznakového parametru a celočíselného parametru:

```
void ComLine(int &S, char &D,  //výstupy
             int pocet, char *param[]){
             //předání počtu a hodnot parametrů
  if (pocet>1) { //je alespoň 1. parametr
     S = strtoint(param[1]);
     if (pocet>2) { //je i druhý parametr
        D = param[2][0];
     }
  }
}
```

Převod řetězce na číslo

- Pro převod je využita konverze při čtení z textového souboru.

```
int strtoint(char *X){ //konverze na int
  int vysled;
  fstream Soub (”temp.txt”,ios::out);
  Soub<<X;      //výpis řetězce do souboru
  Soub.close();
  Soub.open(”temp.txt”,ios::in);
  Soub>>vysled; //přečtení s konverzí
  Soub.close();
  return vysled;
}
```

Proměnné prostředí

- Běžící proces má k dispozici paměťovou oblast zvanou „prostředí“ (environment)

Proměnné prostředí

- Běžící proces má k dispozici paměťovou oblast zvanou „prostředí“ (environment)
- Oblast se dědí z nadřízeného procesu.

Proměnné prostředí

- Běžící proces má k dispozici paměťovou oblast zvanou „prostředí“ (environment)
- Oblast se dědí z nadřízeného procesu.
- V této oblasti mohou být různé proměnné, jejichž hodnoty lze v OS nastavit. Přík. řádek:

  `export PROMENNA=hodnota` – nastavení hodnoty;

  `export PROMENNA=` – zrušení proměnné

Proměnné prostředí

- Běžící proces má k dispozici paměťovou oblast zvanou „prostředí“ (environment)
- Oblast se dědí z nadřízeného procesu.
- V této oblasti mohou být různé proměnné, jejichž hodnoty lze v OS nastavit. Přík. řádek:

  `export PROMENNA=hodnota` – nastavení hodnoty;

  `export PROMENNA=` – zrušení proměnné
- Hodnoty jsou vždy řetězcového typu.

Proměnné prostředí

- Běžící proces má k dispozici paměťovou oblast zvanou „prostředí“ (environment)
- Oblast se dědí z nadřízeného procesu.
- V této oblasti mohou být různé proměnné, jejichž hodnoty lze v OS nastavit. Přík. řádek:
  `export PROMENNA=hodnota` – nastavení hodnoty;
  `export PROMENNA=` – zrušení proměnné
- Hodnoty jsou vždy řetězcového typu.
- Obsah proměnných dodá funkce `getenv(název)`

Proměnné prostředí

- Běžící proces má k dispozici paměťovou oblast zvanou „prostředí“ (environment)
- Oblast se dědí z nadřízeného procesu.
- V této oblasti mohou být různé proměnné, jejichž hodnoty lze v OS nastavit. Přík. řádek:
  `export PROMENNA=hodnota` – nastavení hodnoty;
  `export PROMENNA=` – zrušení proměnné
- Hodnoty jsou vždy řetězcového typu.
- Obsah proměnných dodá funkce `getenv(název)`
- Funkce nabývá řetězcové hodnoty obsahu proměnné, jejíž název je zadán jako parametr.

Proměnné prostředí

- Běžící proces má k dispozici paměťovou oblast zvanou „prostředí“ (environment)
- Oblast se dědí z nadřízeného procesu.
- V této oblasti mohou být různé proměnné, jejichž hodnoty lze v OS nastavit. Přík. řádek:
  `export PROMENNA=hodnota` – nastavení hodnoty;
  `export PROMENNA=` – zrušení proměnné
- Hodnoty jsou vždy řetězcového typu.
- Obsah proměnných dodá funkce `getenv(název)`
- Funkce nabývá řetězcové hodnoty obsahu proměnné, jejíž název je zadán jako parametr.
- Pokud proměnná neexistuje, nabývá funkce hodnoty prázdného řetězce.

Proměnné prostředí

- Běžící proces má k dispozici paměťovou oblast zvanou „prostředí“ (environment)
- Oblast se dědí z nadřízeného procesu.
- V této oblasti mohou být různé proměnné, jejichž hodnoty lze v OS nastavit. Přík. řádek:
  `export PROMENNA=hodnota` – nastavení hodnoty;
  `export PROMENNA=` – zrušení proměnné
- Hodnoty jsou vždy řetězcového typu.
- Obsah proměnných dodá funkce `getenv(název)`
- Funkce nabývá řetězcové hodnoty obsahu proměnné, jejíž název je zadán jako parametr.
- Pokud proměnná neexistuje, nabývá funkce hodnoty prázdného řetězce.
- Test na existenci proměnné lze provést jako test délky dodaného řetězce (funkce `strlen(R)`)

Příklad získání obsahu proměnných

- Předpokládejme, že v proměnné NUMSTART je číselná hodnota a v proměnné NUMDELIM je řetězcová hodnota, z níž potřebujeme pouze první znak.

```
void Envir(int &S, char &D){
  char *Pom;
  Pom = getenv(”NUMSTART”);
  if (strlen(Pom)!=0) S = strtoint(Pom);
  Pom = getenv(”NUMDELIM”);
  if (strlen(Pom)!=0) D=Pom[0];
}
```

Práce se soubory

- (Opakování:) Soubor na disku lze otevřít, zpracovat, modifikovat apod. Je možným vstupem i výstupem dat programu.

Práce se soubory

- (Opakování:) Soubor na disku lze otevřít, zpracovat, modifikovat apod. Je možným vstupem i výstupem dat programu.
- Pro základní manipulaci je k dispozici objekt `fstream`.

Práce se soubory

- (Opakování:) Soubor na disku lze otevřít, zpracovat, modifikovat apod. Je možným vstupem i výstupem dat programu.
- Pro základní manipulaci je k dispozici objekt `fstream`.
- Soubor lze otevřít jako textový (implicitně) nebo jako binární.

Práce se soubory

- (Opakování:) Soubor na disku lze otevřít, zpracovat, modifikovat apod. Je možným vstupem i výstupem dat programu.
- Pro základní manipulaci je k dispozici objekt `fstream`.
- Soubor lze otevřít jako textový (implicitně) nebo jako binární.
- Metoda `open()` propojí objekt s konkrétním diskovým souborem a otevře soubor v potřebném režimu (textový/binární) a pro potřebnou operaci (vstup, výstup, připojení na konec, vyprázdnění).

Práce se soubory

- (Opakování:) Soubor na disku lze otevřít, zpracovat, modifikovat apod. Je možným vstupem i výstupem dat programu.
- Pro základní manipulaci je k dispozici objekt `fstream`.
- Soubor lze otevřít jako textový (implicitně) nebo jako binární.
- Metoda `open()` propojí objekt s konkrétním diskovým souborem a otevře soubor v potřebném režimu (textový/binární) a pro potřebnou operaci (vstup, výstup, připojení na konec, vyprázdnění).
- Metoda `close()` obslouží buffer a soubor uzavře.

Návratový kód

- Ukončený proces dodává operačnímu systému celočíselnou hodnotu, podle níž OS pozná, jak proces skončil.

Návratový kód

- Ukončený proces dodává operačnímu systému celočíselnou hodnotu, podle níž OS pozná, jak proces skončil.
- Hodnota 0 je chápána jako bezchybné ukončení.

Návratový kód

- Ukončený proces dodává operačnímu systému celočíselnou hodnotu, podle níž OS pozná, jak proces skončil.
- Hodnota 0 je chápána jako bezchybné ukončení.
- Nenulové hodnoty mohou signalizovat různé úrovně problémů, např.
  1 varování – krátká vstupní data
  2 varování – vstupní data byla oříznuta
  4 chyba – vstupní data obsahují chybu
  8 chyba – výsledky nebyly vygenerovány

Návratový kód

- Ukončený proces dodává operačnímu systému celočíselnou hodnotu, podle níž OS pozná, jak proces skončil.
- Hodnota 0 je chápána jako bezchybné ukončení.
- Nenulové hodnoty mohou signalizovat různé úrovně problémů, např.
  - 1 varování – krátká vstupní data
  - 2 varování – vstupní data byla oříznuta
  - 4 chyba – vstupní data obsahují chybu
  - 8 chyba – výsledky nebyly vygenerovány
- Návratový kód je celočíselnou hodnotou funkce `main`.

Příklad práce s návratovým kódem

- Program má sečíst čísla zadaná ze standardního vstupu. Na standardní výstup vypisuje výsledek. Je-li ovšem vstup prázdný, na výstupu se neobjevují validní data (hodnota nula nereprezentuje součet)

Příklad práce s návratovým kódem

- Program má sečíst čísla zadaná ze standardního vstupu. Na standardní výstup vypisuje výsledek. Je-li ovšem vstup prázdný, na výstupu se neobjevují validní data (hodnota nula nereprezentuje součet)
- Následný proces nemůže být spuštěn, je-li výstupní hodnota neplatná

```
int main(){ //program se jmenuje Soucet
   float C, Suma=0;
   int pocet=0;
   while (cin>>C){
       Suma+=C;
       pocet++;
   }
```

Příklad práce s návratovým kódem

- (pokračování kódu programu)

```
    int kod=0; //návratový kód
    if (pocet>0) cout << Suma << endl;
    else {
        cerr << ”chybějí vstupní data”<<endl;
        kod=2;
    }
    return kod;
}
```

Příklad práce s návratovým kódem

- (pokračování kódu programu)

```
    int kod=0; //návratový kód
    if (pocet>0) cout << Suma << endl;
    else {
        cerr << ”chybějí vstupní data”<<endl;
        kod=2;
    }
    return kod;
}
```

- Návratový kód je nula (validní data) nebo 2 (výstup nebyl vygenerován).

Reakce OS

- Výstupní kód je naplněn do proměnné `$?`

Reakce OS

- Výstupní kód je naplněn do proměnné `$?`
- Hodnotu lze testovat příkazem `if` interpretu bash

Reakce OS

- Výstupní kód je naplněn do proměnné `$?`
- Hodnotu lze testovat příkazem `if` interpretu bash
- Skript OS, který řeší celý proces:

```
./Soucet < data.txt > vysled.txt 2>> chyby.log
if test $? -eq 2
# data chybějí -> vygenerují se nová
  then ./GenDat >> data.txt
# jinak se zpracuje součet
  else ./Zpracuj < vysled.txt > uloz.txt
fi
```

Souhrnný příklad

- Základní úloha: program čte řádky ze standardního vstupu a vypisuje je na standardní výstup očíslované pořadovými čísly.

```
int Cislo=1; char Oddel=':'; string Radek;
while (getline(cin, Radek)) {
   cout << setw(5) << Cislo << Oddel
        << " " << Radek << endl;
   Cislo++;
}
```

Souhrnný příklad

- Základní úloha: program čte řádky ze standardního vstupu a vypisuje je na standardní výstup očíslované pořadovými čísly.

```
int Cislo=1; char Oddel=':'; string Radek;
while (getline(cin, Radek)) {
   cout << setw(5) << Cislo << Oddel
        << " " << Radek << endl;
   Cislo++;
}
```

- Chceme tuto základní činnost ovlivnit ve dvou místech:

Souhrnný příklad

- Základní úloha: program čte řádky ze standardního vstupu a vypisuje je na standardní výstup očíslované pořadovými čísly.

```
int Cislo=1; char Oddel=':'; string Radek;
while (getline(cin, Radek)) {
   cout << setw(5) << Cislo << Oddel
        << " " << Radek << endl;
   Cislo++;
}
```

- Chceme tuto základní činnost ovlivnit ve dvou místech:
  - hodnota pořadového čísla prvního řádku;

Souhrnný příklad

- Základní úloha: program čte řádky ze standardního vstupu a vypisuje je na standardní výstup očíslované pořadovými čísly.

```
int Cislo=1; char Oddel=':'; string Radek;
while (getline(cin, Radek)) {
   cout << setw(5) << Cislo << Oddel
        << " " << Radek << endl;
   Cislo++;
}
```

- Chceme tuto základní činnost ovlivnit ve dvou místech:
  - hodnota pořadového čísla prvního řádku;
  - znakový oddělovač pořadového čísla od textu řádku.

Hlavní program

- Možnosti nastavení:
  1. pevné hodnoty uvnitř programu (viz výše);
  2. hodnoty v konfiguračním souboru (systémový, uživatelský); systémový soubor nastavuje správce systému a platí implicitně pro všechny uživatele, uživatelský soubor si nastaví pro sebe daný uživatel a jde o implicitní hodnoty pro veškeré používání programu;
  3. hodnoty v proměnných prostředí; hodnoty jsou platné v rozsahu jednoho sezení (činnosti příkazového interpretu);
  4. hodnoty zadané z příkazového řádku; hodnoty jsou platné pro jedno konkrétní spuštění programu.

Hlavní program

- Možnosti nastavení:
  1. pevné hodnoty uvnitř programu (viz výše);
  2. hodnoty v konfiguračním souboru (systémový, uživatelský); systémový soubor nastavuje správce systému a platí implicitně pro všechny uživatele, uživatelský soubor si nastaví pro sebe daný uživatel a jde o implicitní hodnoty pro veškeré používání programu;
  3. hodnoty v proměnných prostředí; hodnoty jsou platné v rozsahu jednoho sezení (činnosti příkazového interpretu);
  4. hodnoty zadané z příkazového řádku; hodnoty jsou platné pro jedno konkrétní spuštění programu.
- Možnost uvedená jako následující vždy potlačuje všechna předchozí nastavení.

Hlavní program

- Základní činnost obohatíme o možnosti konfigurací:

```
int main(int pocet, char *param[]){
   int Cislo=1; char Oddel=':'; string Radek;
   Config(Cislo, Oddel, param); //záleží na pořadí
   Envir(Cislo, Oddel);
   ComLine(Cislo, Oddel, pocet, param);
   while (getline(cin, Radek)) {
     cout << setw(5) << Cislo << Oddel
          << " " << Radek << endl;
     Cislo++;
   }
   return Cislo;
      //lze využít pro následné číslování
}
```

Čtení konfiguračního souboru

- Předpokládáme, že konfigurační soubor má stejné jméno jako spouštěný program, leží ve stejném adresáři, je textový a má rozšíření `.cfg`

Čtení konfiguračního souboru

- Předpokládáme, že konfigurační soubor má stejné jméno jako spouštěný program, leží ve stejném adresáři, je textový a má rozšíření `.cfg`
- V souboru jsou pouze dvě hodnoty: jedno celé číslo a jeden znak

```
void Config(int &S, char &D, char *param[]){
   char *Pom;
   strcpy(Pom, param[0]); //kopie 1. parametru
   strcat(Pom, ".cfg"); //přidání rozšíření .cfg
   ifstream Cf (Pom); //pokus o otevření souboru
   if (Cf.is_open()) Cf >> S >> D;
      //když soubor existuje a je otevřen, čtení
   Cf.close();
}
```