Formátování kódu

➢ Vertikální uspořádání

Deklarace a funkce vzájemně oddělujeme prázdným řádkem. Jestliže pak procházíme kód směrem shora-dolů, zastavuje se náš zrak postupně na řádcích, které následují za prázdným řádkem.

```
struct Uzel { int c; Uzel *nasl; };

Uzel *prvni=0;

void pridat(int c) {
  Uzel *u = new Uzel;
    u->c = c;
    u->nasl = prvni;
    prvni = u;
}

Uzel *hledat(Uzel *prvni, int x) {
  for (Uzel *u=prvni; u; u=u->nasl) if (u->c == x) return u;
  return 0;
}
```

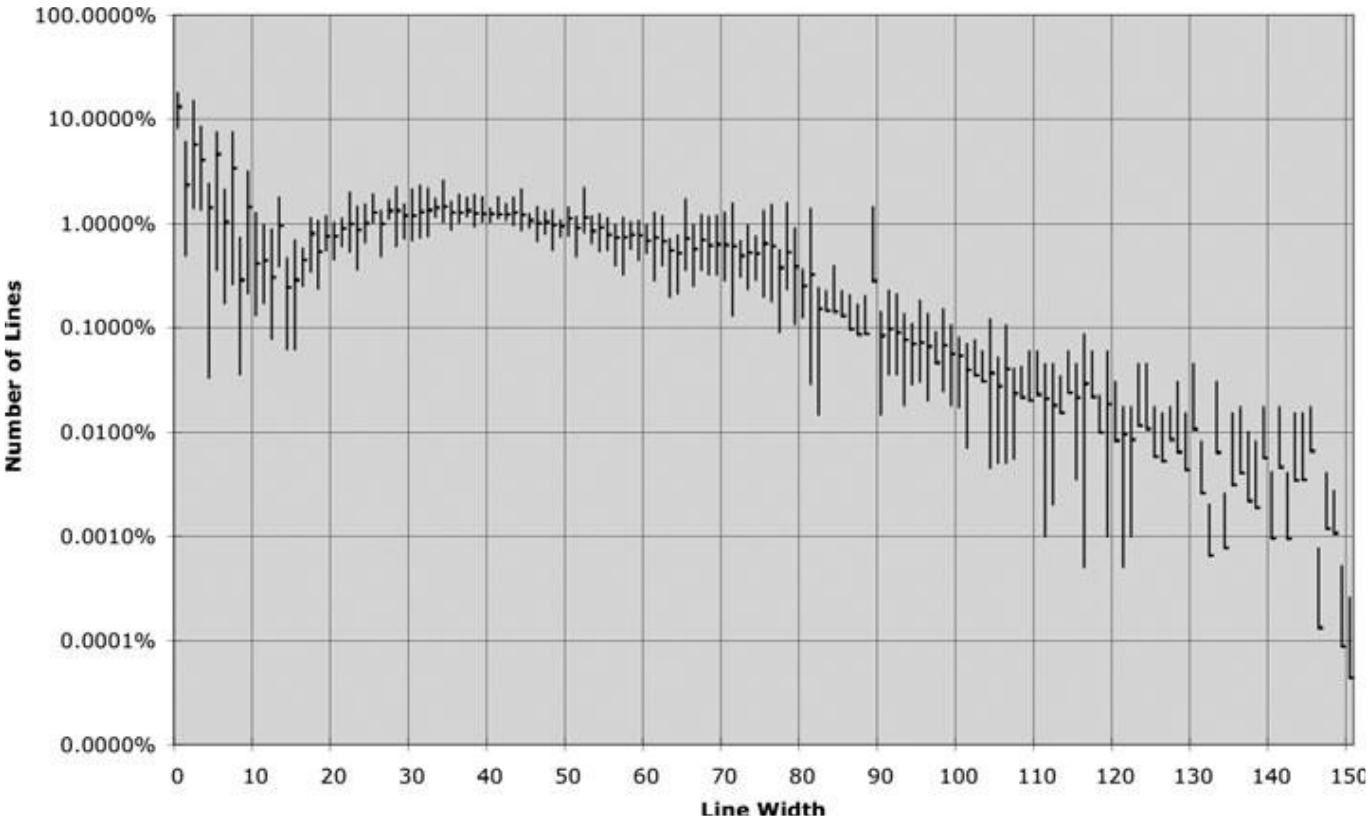
Obecně platí zásada, že části zdrojového kódu, které spolu souvisí (popisují stejný koncept), by měly být psány dostatečně hustě - neměly by být vzájemně odděleny větším počtem prázdných řádků.

```
class Seznam {

  struct Uzel { int c; Uzel *nasl; };

  Uzel *prvni;

  public: Seznam() { prvni = NULL; }

          void pridat(int);

          Uzel *najit(Uzel *, int);
};

class Seznam {
  struct Uzel { int c; Uzel *nasl; };
  Uzel *prvni;

  public: Seznam() { prvni = NULL; }
          void pridat(int);
          Uzel *najit(Uzel *, int);
};
```

Závislost funkcí. Volá-li jedna funkce jinou funkci, měly by obě funkce být blízko sebe a volající funkce by měla být nad funkcí, kterou volá, je-li to možné (například členské funkce ve třídě). To umožňuje snadněji najít volanou funkci a zvyšuje to přehlednost zdrojového kódu.

- **Horizontální uspořádání**

Délka řádku. Řádky zdrojového kódu bývají poměrně krátké – délka do 60 znaků. Maximální délka řádku by měla být do 100 až 120 znaků.



Na horizontální úrovni vkládáme mezery tak, aby se odlišily části, mezi kterými je silný vztah, a části se slabým vzájemným vztahem.

```
int c = u->c;
```

Před operátorem přiřazení a za ním jsou vloženy mezery, aby se zdůraznilo oddělení levé části přiřazení od pravé strany přiřazení.

```
Uzel *u = najit(prvni, 12);
```

Mezi jménem funkce a závorkou omezující parametry není mezera, aby se zdůraznila silný vztah mezi funkcí a jejími parametry. Naopak za čárkou oddělující parametry je mezera, aby se zvýraznilo oddělení parametrů.

```
float diskriminant = b*b - 4*a*c;
float koren1 = (-b + sqrt(diskriminant)) / (2*a),
      koren2 = (-b - sqrt(diskriminant)) / (2*a);
```

- **Odsazení**

Vyjadřuje hierarchii (rozsah platnosti) jednotlivých částí kódu. Části na globální úrovni (deklarace proměnných, definice uživatelských datových typů, hlavičky funkcí) jsou bez

odsazení. Proměnné a funkce uvnitř tříd jsou odsazeny o jednu úroveň. S každým blokem se odsazení vždy o jednu úroveň zvyšuje.

```
template<class T>void Quicksort(T a[],int k,int l)
{for(int i=k;;){T x=a[(k+l)/2];int j=l;do{while(a[i]<x)++i;
while(x<a[j])--j;if(i>j)break;
T w=a[i];a[i]=a[j];a[j]=w;++i;--j;}while(i<=j);
if(k<j)Quicksort(a,k,j);if(i>=l)return;k=i;}
```

```
template<class T>
void Quicksort(T a[], int k, int l)
{ for (int i=k;;) {
    T x = a[(k + l)/2];
    int j = l;
    do {
      while (a[i]<x) ++i;
      while (x<a[j]) --j;
      if (i>j) break;
      T w = a[i];
      a[i] = a[j];
      a[j] = w;
      ++i; --j;
    } while (i<=j);

    if (k<j) Quicksort(a,k,j);
    if (i>=l) return;
    k = i;
  }
```

Odsazení sloupců v zápisu více hodnot zvýší čitelnost.

```
ulozitAdresu("Thámova 15",       "Praha 8",     18600);
ulozitAdresu("Koliště 67a",      "Brno",        60200);
ulozitAdresu("Mlynské Nivy 58", "Bratislava", 82105);
```

Deklarace je užitečné uspořádat do logických skupin.

```
class HasovaciTabulka {
  public:
    HasovaciTabulka();
    bool vlozit(Prvek &);
    Prvek *najit(Klic &);
    Prvek *prvni();
    virtual unsigned hasovaciFunkce(Klic &);
    bool jePosledni();
    Prvek *aktualni();
    virtual unsigned sekundarniFunkce(Klic &);
    bool odebrat(Klic &);
    Prvek *dalsi();
    ~HasovaciTabulka();
};
```

Při uspořádání do logických skupin je třída přehlednější.

```
class HasovaciTabulka {
  public:
    HasovaciTabulka();
    ~HasovaciTabulka();

    virtual unsigned hasovaciFunkce(Klic &);
    virtual unsigned sekundarniFunkce(Klic &);

    bool  vlozit(Prvek &);
    bool  odebrat(Klic &);
    Prvek *najit(Klic &);

    Prvek *prvni();
    Prvek *aktualni();
    bool  jePosledni();
    Prvek *dalsi();
};
```

➢ **Styly formátování**

- ANSI styl

```
void foo()
{
      ...
  while (podm1)
  {
    prikaz1;
    prikaz2;
  }
      ...
  if (podm2)
  {
    prikaz3;
  }
  else
  {
    prikaz4;
  }
      ...
}
```

Závorky asociované s příkazem jsou na samostatných řádcích a mají stejnou úroveň odsazení jako příkaz. Deklarace a příkazy uvnitř závorek jsou odsazeny o 1 úroveň.

- K&R styl (zavedený s jazykem C)

```
void foo()
{
      ...
```

```
  while (podm1) {
    prikaz1;
    prikaz2;
  }
      ...
  if (podm2) {
    prikaz3;
  } else {
    prikaz4;
  }
      ...
}
```

Otevírací závorka asociovaná s příkazem je na stejném řádku jako příkaz. Deklarace a příkazy uvnitř závorek jsou odsazeny o 1 úroveň. Uzavírací závorka má stejné odsazení jako příkaz. Výjimku tvoří závorky uzavírající tělo funkce, zde otevírací závorka je na novém řádku.

- GNU styl

```
void foo()
{
      ...
  while (podm1)
    {
      prikaz1;
      prikaz2;
    }
      ...
  if (podm2)
    {
      prikaz3;
    }
  else
    {
      prikaz4;
    }
      ...
}
```

Závorky asociované s příkazem jsou na samostatných řádcích a jsou odsazeny o 1 úroveň (vyjma závorek uzavírajících tělo funkce).

- Styl, ve kterém je první deklarace nebo příkaz na stejném řádku jako otevírací závorka.

```
void foo()
{
      ...
  while (podm1)
  { prikaz1;
    prikaz2;
```

```
    }
        ...
    if (podm2)
    { prikaz3;
    }
    else
    { prikaz4;
    }
        ...
}
```

- Styl podobný K&R, ale uzavírací závorka má stejné odsazení jako příkazy uvnitř závorky.

```
void foo()
{
        ...
  while (podm1) {
    prikaz1;
    prikaz2;
    }
        ...
  if (podm2) {
    prikaz3;
    } else {
    prikaz4;
    }
        ...
  }
```