Využití maker v jazyce C, hlavičkové soubory

- kontrola ukazatelů předávaných jako parametry funkcím
- co je nutné kontrolovat a proč?
- kontrola za předaný ukazatel nemá hodnotu NULL – v tom případě se s ním nesmí pracovat

Pro hlavičku funkce

```
int Funkce(int *aUk)
{  return *aUk; }   // co se stane v případě, že aUk == NULL?
```

Vytvoření makra pro kontrolu a odchod z funkce

```
#define RETIFNULL(uk,val)    if((uk) == NULL) return val

int Funkce(int *aUk)
{
   RETIFNULL(aUk,1); // pokud je ukazatel NULL vrátí jedničku
   return *aUk;   // co se stane v případě, že aUk == NULL?
}
```

Pro ukazatel na ukazatel s hlavičkou funkce

```
int Alloc(int **aUk)
{
  (*aUk) = calloc(10,sizeof(int)); // co se může pokazit?
  return 0; }
```

Vytvoření makra pro kontrolu a odchod z funkce – při alokaci paměti do ukazatele zkontroluje, zda ukazatel není NULL a že *uk je NULL. Pokud by měl hodnotu, znamená to, že už má paměť přidělenou a opětovným přidělením by se původní hodnota ztratila

```
#define RETIFINIT(uk,val1,val2){   if((uk) == NULL) return val1; \
                                   if(*(uk)!= NULL) return val2; }


int Alloc(int **aUk) // kontrola při alokování
{
  RETIFINIT(aUk,-1,-2); // pokud je ukazatel NULL vrátí -1
                        // pokud má již ukazatel paměť vrátí -2
  (*aUk) = calloc(10,sizeof(int));
  return 0;}
```

Pro ukazatel na ukazatel s hlavičkou funkce

```
int Set(int **aUk)
{
  (*aUk)[0] = 0; // co se může pokazit?
  return 0; }
```

Vytvoření makra pro kontrolu a odchod z funkce – při použití dat přes ukazatel zkontroluje, zda ukazatel není NULL a že *uk není NULL. Pokud by měl hodnotu NULL, znamená to, že už neukazuje na paměť ve které jsou data, která chceme použít

```
#define RETIFINIT(uk,val1,val2)  { if((uk) == NULL) return val1; \
                                   if(*(uk)!= NULL) return val2; }

#define RETIFNOTINIT(uk,val1,val2){if((uk) == NULL) return val1; \
                                   if(*(uk)== NULL) return val2; }

int Set(int **aUk) // kontrola při použití
{
  RETIFNOTINIT(aUk,-1,-2); // pokud je ukazatel NULL vrátí -1
                           // pokud není paměť s daty vrátí -2
  (*aUk)[0] = 0;
  return 0;}
```

Využití maker v jazyce C, hlavičkové soubory

- hlavičkové soubory by měly být ošetřeny proti vícenásobnému načtení.
  Využívá se funkcí preprocesoru pro podmíněný překlad kódu

```
#ifndef JEDNOZNACNY_IDENTIFIKATOR
// umístit na první řádek souboru. Někdy až za komentářem k
// souboru. Při prvním průchodu není definován a proto
// se tato část překládá
#define JEDNOZNACNY_IDENTIFIKATOR
// co nejdříve (na druhém řádku) nadefinujeme, takže při dalším
// načtení je již definován a k překladu nedojde

#endif
```

- do hlavičkového souboru se vkládají části kódu: které netvoří kód (protože při vícenásobném načtení hlavičkového souboru by došlo ke kolizi), a které chceme publikovat (v hlavičkovém souboru jsou pouze ty prototypy funkcí, které chceme používat mimo modul, kde jsou definovány)

Využití maker v jazyce C, hlavičkové soubory

- vytvoření funkce pro daný typ se jménem obsahujícím daný typ
- operátor preprocesoru # pro vytvoření řetězce

```
#define TYPE double
#define TYPE_PRINT "%lf"
// definice typu a jeho tiskového řetězce

#define TO_STR(str) (#str)
// text za # se uzavře do uvozovek



double a = 8;
printf("Vysledek " TYPE_PRINT " pro typ %s ",a, TO_STR(TYPE));
// Řetězce (viz. formátovací řetězec), které jsou vedle sebe,
// překladač spojí do jednoho.
// Pomocí makra TO_STR se z textu v TYPE stane řetězec “double“
// # nejde použít přímo ve zdrojovém textu
```

Využití maker v jazyce C, hlavičkové soubory

- vytvoření funkce pro daný typ se jménem obsahujícím daný typ

- pro tvorbu názvu funkce využijeme maker
- využití operátoru preprocesoru ## pro spojování řetězců
- dosazování maker a použití operátorů preprocesoru má daná pravidla. Proto se může následující konstrukce jevit jako zbytečně složitá

Využití maker v jazyce C, hlavičkové soubory

```
#define TYPE double
#define TYPE_PRINT "%lf"
// definice typu a jeho tiskového řetězce

#define TO_STR(str) (#str)

#define NAME2(fun,suffix) fun ## _ ## suffix
#define NAME1(fun,suffix) NAME2(fun,suffix)
#define NAME(fun) NAME1(fun,TYPE)
// sekvence maker pro vytvoření jména funkce na základě parametrů


TYPE** NAME(Alokuj)(size_t x,size_t y)
// výsledný překlad po postupném dosazování maker bude:
// double ** Alokuj_double(size_t x, size_t y)
{
printf("Vysledek " TYPE_PRINT " pro typ %s", TO_STR(TYPE));
}


#define NAME(fun) NAME1(fun,__COUNTER__)
// __COUNTER__ je proměnná preprocesoru, která se po použití
// inkrementuje. Generují se tedy funkce s různými čísly v názvu
```

Využití maker v jazyce C, hlavičkové soubory

- pro využití s jiným typem je možné použít stejné funkce a z maker použít
- je možné vkládat pomocí include kód? Pokud ano, za jakých podmínek?
  Jaké jsou nevýhody a výhody?

```
#define TYPE int
#define TYPE_PRINT "%d"
// definice typu a jeho tiskového řetězce

#define NAME(fun) NAME1(fun,TYPE)
// protože makro NAME1 se musí generovat s novým typem

TYPE** NAME(Alokuj)(size_t x,size_t y)
// výsledný překlad po postupném dosazování maker bude:
// int ** Alokuj_int(size_t x, size_t y)
{
printf("Vysledek " TYPE_PRINT " pro typ %s", TO_STR(TYPE));
}
 // volání je potom
double ** ukd = Alokuj_double(3,5);
int    ** uki = Alokuj_int   (4,6);
```