Warsztaty C cz. 2

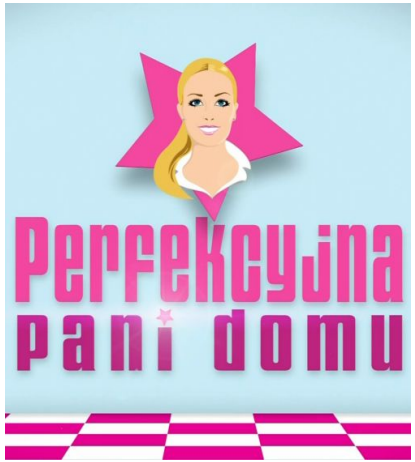
Dobre praktyki i styl / Makra

Zuzanna Krawczyk
Bartosz Chaber
Mariusz Zaborski

O czym dzisiaj powiemy?

- Rady dotyczące dobrych praktyk (ogólnie!).
- Garść porad dotyczących utrzymania dobrego stylu z kernela FreeBSD:
  - dobry styl jako dobra praktyka :)
- Makra - kiedy używać i czego się bać:
  - makro do tworzenia generycznych funkcji,
  - do {...} while(...)
  - gcc -E
  - funkcje inline?

Dobre praktyki



Dobre praktyki

- Używanie nazw które coś mówią
  - r/w ???
- Nie za długie nazwy funkcji
  - funkcja_zczytujaca_dane_z_pliku_w_formacie_csv
- Nie za długie nazwy zmiennych
  - zmienna_uzywana_jako_licznik
- Nie za długie linie
  - for (i = 0; i < n; i++) if (i % 2 == 0) puts(“parzysta); else if(i % 2 == 1) puts(“nieparzysta”);

Dobre praktyki

- Używanie wcięć

```
double
foo(double a, double b, double d, double (*f)(double)) {
double result, x, da, db;
result = 0;
for(x=a; x<b; x+=d) {
da=f(i); db=f(i + d);
result+=da+db; }
return (result*d)/2;
}
```

Dobre praktyki

- Ok będę używał wcięć!!!

```
double
foo(double a, double b, double d, double (*f)(double)) {
        double result, x, da, db;
                result = 0;
        for(x=a; x<b; x+=d) {
                                da=f(i); db=f(i + d);
result+=da+db; }
                                return (result*d)/2;
}
```

Dobre praktyki

- Rozsądne używanie wcięć !!!

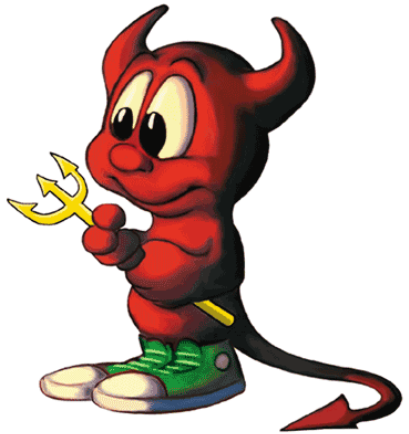
```
double
foo(double a, double b, double d, double (*f)(double)) {
        double result, x, da, db;
        result = 0;
        for(x=a; x<b; x+=d) {
                da=f(i);
                db=f(i + d);
                result+=da+db;
        }
        return (result*d)/2;
}
```

Dobre praktyki

- Używanie dobrze zdefiniowanego stylu (np. K&R)

```
int main(int argc, char *argv[])
{
    ...
    while (x == y) {
        something();
        somethingelse();

        if (some_error) {
            do_correct();
        } else
            continue_as_usual();
    }
    ...
}
```

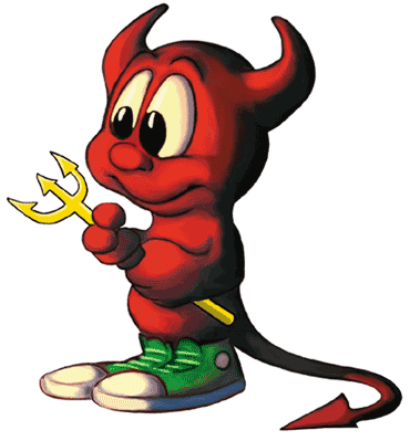
Garść porad dotyczących utrzymania dobrego stylu z kernela FreeBSD



Styl

- Derektywy include posortowoane alfabetycznie:

```
#include <dirent.h>
#include <err.h>
#include <errno.h>
#include <langinfo.h>
#include <locale.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <timeconv.h>
#include <unistd.h>
```



Styl

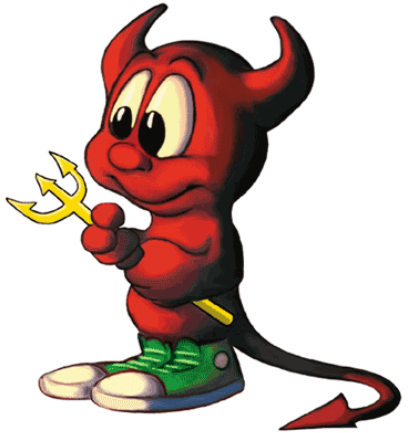
- Nazwy funckji
- nazwy zmiennych
- warunki mają wartość logiczną
- deklaracje zmiennych

```
void
sth_foo(int maxlen)
{
        int a,b,c;

        a = 5;
        ...
}

void
fancy_sth(int maxlen)
{

        while (maxlen >= 0) {
                ...
        }
}

void
sth(int a)
{

        if (a > 0)
}
```

Styl

- I jeszcze więcej wcięć...

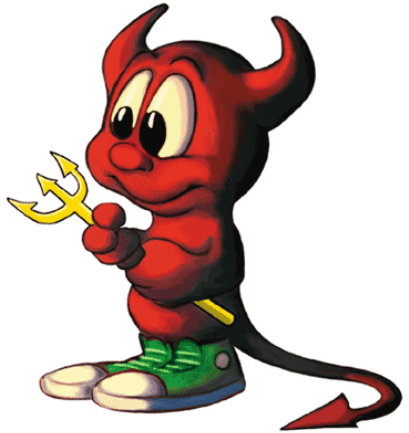
```
if (this != 0 && that != 100 && 3 > x &&
    this != that) {
        do_something_nasty();
}
```



Styl

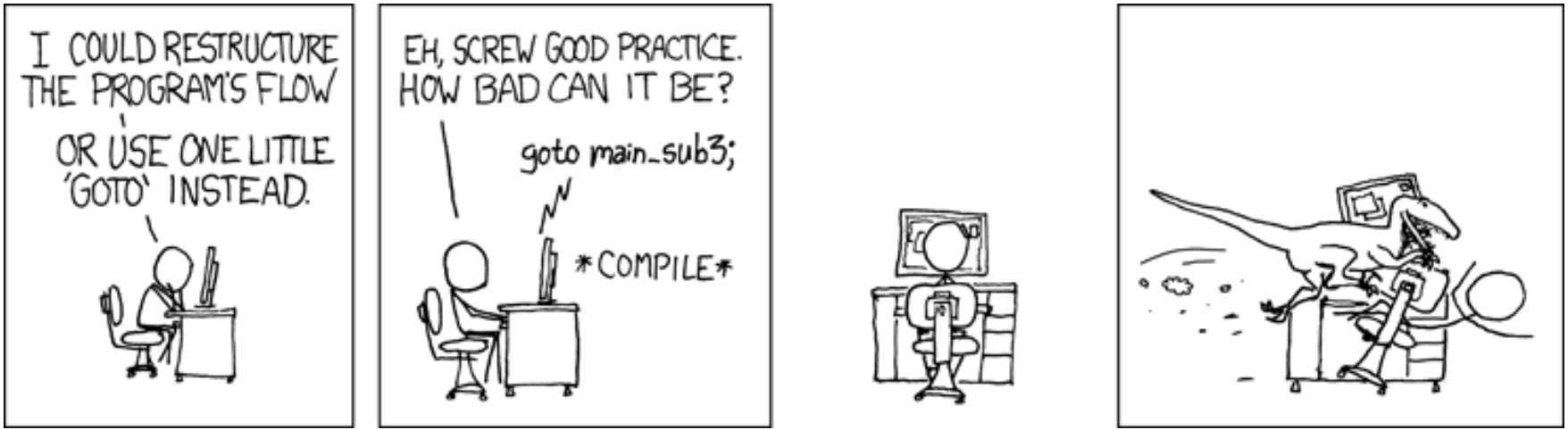
- return

```
int
sth()
{
        return (0);
}
```



Styl

- goto


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

Styl

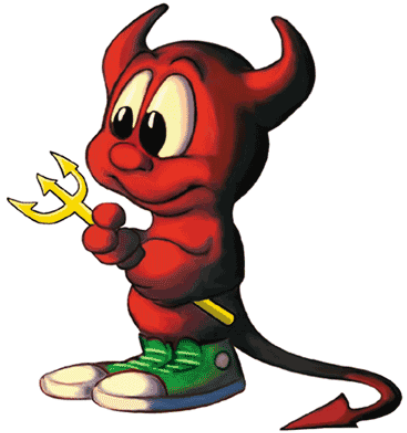
- goto

```
int**
foo(int a, int b)
{
        int **tab;
        int i;

        i = 0;
        if (a < 0 || b < 0)
                return (NULL);

        tab = malloc(a * sizeof(*tab));
        if (tab == NULL)
                goto error;
        for (i = 0; i < a; i++) {
                tab[i] = malloc(b * sizeof(**tab));
                if (tab[i] == NULL)
                        goto error;
        }

        return (tab);
error:
        for (; i > 0; i--)
                free(tab[i]);
        free(tab);
        return (NULL);
}
```

Styl

style(9)

man style



Czym są makra?

Makra w C

- pozwalają wyodrębnić powtarzalne wzorce w kodzie.
- przed kompilacją są “rozwijane” przez kompilator.
- makra nie mają swojego własnego zasięgu.

Makra - zaglądamy pod maskę

```c
#include <stdio.h>
#define MAX(a, b) (((a) > (b))? (a) : (b))

int
main(int argc, char **argv) {
   return MAX(3, 14);
}
```

gcc -E main.c > main_preprocessed.c

Makra - zaglądamy pod maskę

```
# 2 "b.c" 2

int
main(int argc, char **argv) {
        return (((3) > (14))? (3) : (14));
}
```

Makra - zaglądamy pod maskę

```c
#include <stdio.h>

#define MAX(a, b) (((a) > (b)) ? (a) : (b))

int
main(void) {
        int a, b;

        a = 4;
        b = 2;

        return MAX(--a, b);
}
```

Makra - zaglądamy pod maskę

```
int
main(void) {
        int a, b;

        a = 4;
        b = 2;

        return (((--a) > (b)) ? (--a) : (b));
}
```

Makro w pętli...

```
#include <stdio.h>
#define free_me(p)   free(p); p = NULL;
#define N 5

int main(int argc, char** argv) {
        char* names[N];

        for (i=0; i<N; i++)
                names[i] = malloc(i * sizeof(char));

        for (i=0; i<N; i++)
                free_me(names[i]);
}
```

Zaczynamy kod! *Excited?*

Makra

Zadanie

Przygotuj zestaw przydatnych makr do swoich przyszłych projektów w C. Np.:

- makro do tworzenia komunikatów aplikacji, które wspiera różne poziomy, automatycznie dodaje linię i nazwę pliku, z którego zostało wywołane.
LOG(level, …)

- makro, które wypisze podany komunikat a następnie (wyjdzie z funkcji)/(zakończy program) zwracając podaną wartość: CRY(status, …), DIE(status, …)