Zadanie A4: Kompilacja z Makefilem (30 punktów)

Napisz pojedyńczego Makefile-a którego zadaniem będzie skompilowanie i zlinkowanie projektu napisanego w języku C. Makefile będzie umieszczony w katalogu głównym projektu. Pliki źródłowe znajdują się w poddrzewie `src`. Katalog `src` może zawierać kolejne podkatalogi które też trzeba sprawdzić.

Domyślna reguła `all` powinna skompilować i zlinkować wszystkie pliki. To znaczy:

Każdy plik z końcówką `.c` trzeba skompilować do odpowiedniego pliku z końcówką `.o`. Postały plik `.o` powinien znaleźć się w poddrzewie `build` które powinno mieć taką samą strukturę podkatalogów jak `src` (nawet jeśli któreś podkatalogi `src` są puste, lub nie zawierają plików źródłowych)

Wszystkie pliki `.o` powinny zostać zlinkowane do programu `main` umieszczonego w podkatalogu `bin` głównego katalogu projektu.

Ponadto dla każdego pliku `.c` należy wygenerować listę zależności, umieszczoną w pliku `.d`, w odpowienim miejscu podkatalugo `build`. Zmiana w plikach źródłowych powinna powodować minimalną rekompilację projektu.

Żadne inne pliki nie powinny powstawać.

Wszystkie reguły budowania plików `.o` i `main` powinny wypisywać na początku

Building <nazwa celu>

i nic poza tym. Pozostałe reguły powinny być zupełnie ciche.

Reguła `clean` powinna usunąć wszystkie pliki tworzone przez Makefilea i wszystkie puste katalogi w drzewie `build` i `bin`. Jeśli w `build` lub `bin` istniały wcześniej jakieś inne pliki, powinny one zostać nienaruszone.

Na zajęciach nie omawialiśmy przypadku gdy jakiś plik źródłowy zostanie usunięty. Wtedy stara lista zależności `.d` może odwoływać się do plików już nie istniejących. Można ten przypadek rozwiązać dodatkową komendą przekazywaną do gcc (jaką?).

- Wysyłany plik powinien nazywać się `Makefile`
- Plik powinien mieć uniksowe kodowanie końców linii (`\n`, a nie `\r\n`)
- Makefile powinien byc 'odporny' na zmiany w kodzie źródłowym
- Makefile powinien byc 'odporny' na usuwanie plików w katalogu `build` i `bin`.

Uzupełnienie zajęć:

Reguła dla `X` nie musi tworzyć pliku `X`, nawet jeśli `X` nie jest zadeklarowany jako `.PHONY`. Przyjęte jest, że uruchomienie reguły dla `X` satysfakcjionuje niezbędnik `X` nawet jeśli taki plik nie powstał.

Na zajęciach nie omawialiśmy, a mogą się pojawić niezbędniki których czasu modyfikacji nie musimy sprawdzać (ważne tylko, ze istnieją). Wówczas możemy je umieścić po znaku `|`. Są to tak zwane *order-only dependencies*.

```
<target> : <dependencies> | <order-only dependencies>
```

Przykład:

Załóżmy, że w projekcie znajdują się następujące pliki:
Plik `src/main.c`:

```
#include <stdio.h>
#include <stdlib.h>
#include "nwd.h"

int main(int argc, char** argv) {
        if (argc!=3) {
                printf("Invalid argument count\n");
                return -1;
        }
        int a = atoi(argv[1]);
        int b = atoi(argv[2]);
        int result = nwd(a,b);
        printf("%d\n",result);
        return 0;
}
```

Plik `src/nwd.c`:

```
#include "nwd.h"

int nwd(int a, int b) {
        int t;
        while(b) {
                if (a<b)
                        t=a;
                else
                        t = a % b;
                a = b;
                b = t;
        }
        return a;
}
```

Plik `src/nwd.h`:

```
int nwd(int a, int b);
```

Uruchomienie Makefilea na tak przygotowanym projekcie powinno wypisać:

```
Building build/nwd.o
Building build/main.o
Building bin/main
```

Zaś uruchomienie skompilowanego programu `main` z argumentami `34 51` powinno wypisać jedną linię `17`.