

DOKUMENTACJA FIRMY PROGRAMISTYCZNEJ

DRZEWO GENEALOGICZNE

wersja: 08.06.2017

Członkowie Grupy „4z”:

Łukasz Janus - kierownik grupy

Bartosz Bukowski

Mateusz Marchelewicz

Łukasz Witek vel Witkowski

SPIS TREŚCI

1.	STANDARDY PROGRAMOWANIA.....	3
1.1.	SUBSET STANDARDS	3
1.2.	OTHER RECOMMENDATIONS.....	4
1.3.	STANDARDY DOTYCZĄCE DEFINICJI KLAS	5
1.4.	WZORCE HEADERÓW	6
2.	SPIS DOKUMENTÓW PROJEKTU PROGRAMISTYCZNEGO.....	7
3.	QUALITY ASSURANCE.....	8
3.1.	OBOWIĄZKI OSOBY ODPOWIEDZIALNEJ ZA QA.....	8
3.2.	LISTA DOKUMENTACJI JAKĄ JEST ZOBOWIĄZANA WYPEŁNIAĆ (I PODPISYWAĆ)	8
4.	TESTY	9
4.1.	OBOWIĄZKI OSOBY ODPOWIEDZIALNEJ ZA TESTY	9
4.2.	LISTA DOKUMENTACJI JAKĄ JEST ZOBOWIĄZANA WYPEŁNIAĆ (I PODPISYWAĆ)	9
5.	ORGANIZACJA ŚRODOWISKA PROGRAMISTYCZNEGO.....	10
6.	REGULAMIN	11
7.	INNE	12
7.1.	WZORCE TABELI POSTĘPU.....	12
7.2.	WZORCE RAPORTU BŁĘDÓW	13

1. Standardy programowania.

Standardy programowania

1: Standard Programistyczny projektu „Drzewo Genealogiczne”.

STD-2017

C++ Language Subset interpretation

Prepared by: Mateusz Marchewicz.

02 May 2017

A Glossa C++ Language Subset as applied by Quality Assurance

1. Subset Standards

- S1 Deklaracja każdej klasy powinna znajdować się w osobnym pliku nagłówkowym.**
Interpretacja: ścisła
- S2 Implementacja każdego modułu powinna zawierać tylko jedną klasę.**
Interpretacja: ścisła
- S3 Każde wyrażenie powinno rozpoczynać się w osobnej linii.**
Interpretacja: ścisła
- S4 Nie używać przestrzeni nazw standardowych.**
Interpretacja: zamiast tego zalecane użycie `std::cout` itp.
- S5 Rekursja nie powinna być użyta.**
Interpretacja: ścisła
- S6 Nie mogą być użyte wbudowane biblioteki `string`, `vector` etc.**
Interpretacja: zamiast tego trzeba użyć własnych bibliotek
- S7 W klasach powinny być użyte wirtualne destruktory.**
Interpretacja: zalecane, gdyż występują wirtualne metody
- S8 Każda klasa powinna zawierać konstruktor i destruktor.**
Interpretacja: ścisła
- S9 Każda klasa powinna zawierać konstruktor kopiujący.**
Interpretacja: ścisła
- S10 Każda klasa powinna zawierać przynajmniej przeladowany operator.**
Interpretacja: ścisła
- S11 Zmienne globalne nie powinny być używane.**
Interpretacja: ścisła
- S12 Każde wyrażenie „switch” powinno mieć jeden punkt wejścia.**
Interpretacja: ścisła
- S13 Klasy mogą być „zaprzyjaźnione”.**
Interpretacja: ścisła
- S14 Liczby zmiennoprzecinkowe nie powinny być używane.**
Interpretacja: ścisła
- S15 Deklaracja każdej zmiennej powinna odbywać się w osobnej linii.**
Interpretacja: zmienne tego samego typu też mogą być deklarowane w osobnych liniach
- S16 Unie nie powinny być używane.**
Interpretacja: ścisła

- S17 Składnia z języka C nie powinna być używana.**
Interpretacja: chodzi o printf, scanf, struktury itp.
- S18 Używanie zmiennych const dozwolone.**
Interpretacja: ścisła
- S19 Nie używać skróconej wersji instrukcji if (?:)**
Interpretacja: ścisła
- S20 Wskaźniki powinny być używane w ostateczności.**
Interpretacja: ścisła
- S21 Nie łączyć wyrażeń arytmetycznych ze wskaźnikami.**
Interpretacja: ścisła
- S22 Wyrażenia pragma nie powinny być używane.**
Interpretacja: ścisła
- S23 Obowiązkowe jest użycie instrukcji #ifndef, #define w headerach.**
Interpretacja: każdy plik nagłówkowy musi mieć te instrukcje

2. Other Recommendations

- R1 Nazwy klas zaczynają się dużą literą.**
Interpretacja: ścisła
- R2 Nazwy metod zaczynają się małą jak i dużą literą.**
Interpretacja: ścisła
- R3 Komentarze jednolinijkowe zaczynają się od //. Wielolinijkowe zaś zaczynają się od /* oraz kończą na */.**
Interpretacja: ścisła
- R4 Metody bez parametrów powinny mieć postać np. foo().**
Interpretacja: nie używać foo(void)

3. Additional standards

- A1 Wszystkie argumenty w konstruktorach powinny być wyraźnie wyspecyfikowane.**
Interpretacja: ścisła
- A2 Wszystkie headery powinny mieć formę zgodną z wzorcem nagłówków plików.**
Interpretacja: ścisła

Konwencje nazw: w projekcie została użyta notacja węgierska.

Zasady edycji plików z kodem źródłowym:

Stosujemy wcięcia wielkości dwóch (trzech) spacji.

```
bool operator==(alfabet &a) {  
    if (litera == a.litera&&lp == a.lp&&ascii == a.ascii) return true; return false;  
}
```

Definicje metod i klas są otoczone klamrami za nazwą metody, jak również linię niżej:

```
bool operator==(alfabet &a) {  
    if (litera == a.litera&&lp == a.lp&&ascii == a.ascii) return true; return false;  
}
```

```
void ladowanie_sz(N_vektor<alfabet> &v)  
{  
    char lit = ' ';  
    for (int i = 0; i < 222; i++)  
    {  
        alfabet t;  
        t.litera = lit;  
        t.lp = i + 1;  
        t.ascii = (int)lit;  
        v.m_push_back(t);  
        lit++;  
    }  
}
```

Wzorce: nagłówków plików kodu i headerów

Headery obowiązkowo zawierają nazwę pliku, krótki opis, numer wersji kodu, datę implementacji, autora kodu oraz opis danej implementacji. Dodatkowo każdy header musi zawierać małe zabezpieczenie w postaci instrukcji preprocesora *#ifndef*, *#define*, w celu zabezpieczenia przed kolejnym dołączeniem.

Przykładowy header z projektu:

```
/**
 *
 * "gender.h"
 *
 *
 *
 *
 *CONTENTS:
 * "Klasa dziecko po C_data"
 *
 *HISTORY:
 *version  Date          Changes                      Author/Programmer
 *1.0      26.04.2017     Orginal design                Mateusz Marchelewicz
 *1.1      02.05.2015     Adding a virtual destructor    Lukasz Witek vel Witkowski
 *1.2      02.05.2015     Adding a virtual methods       Lukasz Witek vel Witkowski
 *1.3      02.05.2015     Adding parameter constructors  Lukasz Witek vel Witkowski
 */
#define GENDER_H
#include "data.h"

...

#endif // !GENDER_H
```

2. Spis dokumentów projektu programistycznego.

- 1) Raport z fazy strategicznej.**
- 2) Raport z fazy określania wymagań.**
- 3) Dokument z fazy analizy: opisanie stworzonego modelu i poprawiony dokument z fazy wymagań.**
- 4) Dokumentacja fazy projektowania:**
 - poprawienie dokumentu fazy określania wymagań,
 - poprawiony model, dokument opisujący stworzony projekt,
 - lista klas i powiązania między klasami.
- 5) Dokumentacja fazy implementacji: kod składający się z przetestowanych modułów.**
- 6) Raport z testów modułów.**
- 7) Harmonogram fazy testowania.**
- 8) Dokumentacja administratora.**
- 9) Dokumentacja użytkownika.**
- 10) Dokumentacja z fazy konserwacji: poprawiony i zmodyfikowany kod źródłowy z opisem.**
- 11) Raport postępów produkcji oprogramowania (Progress Report).**
- 12) Dokumentacja QA (kontroli jakości):**
 - unit list,
 - zgodność kodu ze standardami.
- 13) Dokumentacja testowania: Plan testów, Raport testów (Problem Report).**

3. Quality Assurance (kontrola jakości).

Zadania kontrolera jakości w Naszej „firmie”:

- a) sprawdzenie zgodności kodu z ustalonym standardem,
- b) sprawdzenie czytelności kodu,
- c) sprawdzanie poziomu skomplikowania kodu,
- d) na podstawie projektu programu, udostępnianego przez programistów, sporządza i na bieżąco aktualizuje dokument "Lista klas",
- e) wypełnia dla każdej klasy dokument pt. "QA class document" zawierający jedną klasę i potwierdzający jej zgodność ze standardem,
- f) wypełnia odpowiednie rubryki w dokumencie "Progress Report" („Tablica postępów”),
- g) sprawuje pieczę nad dokumentacją testowania.

3.2 Lista dokumentacji jaką QA jest zobowiązany wypełniać

- a) Lista klas,
- b) Tablicę Postępów,
- c) QA class document:

Class name:

Nr standardu lub jego opis	zgodny(Y/N)	uwagi
	podpis	

4. Testy.

Zadania testera w Naszej „firmie”:

- 1) wykonywanie testów poszczególnych klas,
- 2) sprawdzanie poprawności działania programu,
- 3) wypełnianie dokumentu „Raport błędów”
(informacja o typie błędu i sposobie jego usunięcia),
- 4) wypełnianie dokumentu „Tablica postępów”.

Lista dokumentacji jaką jest zobowiązana wypełniać (i podpisywać):

- a) dokument „Raport błędów”,
- b) dokument „Tablica postępów”.

5. Organizacja środowiska programistycznego.

1. Na potrzeby projektu, w celu komunikacji, wymiany pomysłów i plików zostaje utworzona wspólna skrzynka mail'owa - **aplikacjacpp@gmail.com** zabezpieczona hasłem.
2. Równolegle zostaje utworzone repozytorium na portalu *github.com* w celu umieszczania przez zespół kolejnych wersji plików (programu i dokumentacji).
3. Struktura plików projektu dzieli się na oddzielny folder z dokumentacją, oraz dwa foldery z plikami programu: Aplikacja zawierająca kolejne wersje programu i Drzewo_genealogiczne zawierający pliki, narzędzia będące elementami tymczasowymi, służącymi do budowy poszczególnych części programu.
4. Każdy z członków zespołu zobowiązany jest używać do pisania kodu aplikacji Visual Studio 2015 wraz z dodatkiem *github.visualstudio.vsix* w celu prawidłowej obsługi Github'a.
5. Każdy członek zespołu na swoim komputerze posiada stworzony folder o nazwie *genealogy tree* ze strukturą katalogów analogiczną do Github'a.

6. Regulamin.

Wszyscy członkowie zespołu zobowiązani są do przestrzegania zasad bezpieczeństwa związanych z tajemnicą służbową, a w szczególności do:

1. nieudostępniania tworzonego kodu i dokumentacji osobom niezwiązanym z projektem (w całości jak i we fragmentach).
2. nieudzielania osobom trzecim informacji technicznych dot. działania programu.
3. należytego zabezpieczenia swoich komputerów przenośnych (login, hasło, włączona usługa szyfrowania dysku).
4. posługiwania się tylko pocztą firmową w przypadku przekazywania jakichkolwiek plików
5. nieudostępniania nikomu loginów i haseł do kont pocztowych, Github'a.

W przypadku nieprzestrzegania powyższych zasad, osoba natychmiast zostanie odsunięta od pracy nad projektem.

7. Inne (wzorce tabeli postępu, raportu błędów).

Wzorzec tablicy postępów:

Tablica postępów obowiązkowo musi zawierać tytuł projektu do którego się odnosi, krótki opis projektu, twórców tablicy oraz imię i nazwisko odbiorcy. Dodatkowo musi się znaleźć historia wersji w tabelce. Tabelka ta musi zawierać datę modyfikacji, autora oraz krótki opis tego, co zostało dodane.

Tabela główna postępów to najważniejszy punkt w tym podrozdziale. Musi obowiązkowo zawierać numer wersji kodu oraz kompilacji. Do tego pola z opisem, krótkim wyjaśnieniem, tworzeniem, akceptacją, implementacją, testów QA oraz uwag końcowych. W/w pola wypełniane są literką Y lub N (co oznacza „Tak” lub „Nie”).

Document: **FamilyTree_Documentation**

FFFFFF Program tworzący drzewo genealogiczne

Originator: M. Marchelewicz Recipient: A. Kryś

Version history:

Date	Created	Remarks
2017/05/04	M. Marchelewicz	Original
..		

Ref.as per FFFFFFF	Build	Description (Opis)	Clarified (Wyjaśnione)	Design I	Accept	Design II	Implement	QA	Test	Remarks (Uwagi)
Platform										
1.1.1	[B1]	Hardware	Y	Y	Y	Y	NA	NA		
1.1.2	[B1]	OS	Y	Y	Y	Y	NA	NA		
1.1.3	[B1]	Compiler	Y	Y	Y	Y	NA	NA		
Displayed objects										

Wzór Raportu błędów:

Raport błędów musi obowiązkowo występować w postaci tabelki.

Tabelka ta musi zawierać datę odkrycia błędu, numer wersji kodu, testera, który odkrył błąd. Dodatkowo każdy błąd musi zawierać krótki opis czego dotyczył, rodzaj błędu mierzony w skali 1-4

(1 to krytyczny, 4 - kosmetyczny).

Tabela musi kończyć się inicjałami programisty/testera, który naprawił błąd oraz krótką uwagą.

Przykładowy raport błędów w firmie:

Rodzaj błędu: krytyczny(1), ważny(2), mało ważny(3), kosmetyczny(4).

data	wersja kodu	tester	opis błędu	rodzaj błędu	kto naprawił błąd	uwagi