Wstęp

Niniejsza instrukcja przeznaczona jest przede wszystkim dla studentów kierunku informatyka odbywających zajęcia laboratoryjne z programowania w Zakładzie Oprogramowania. Zawiera opis wzorcowego sprawozdania z wykonania ćwiczenia, jakim jest napisanie programu w języku wysokiego poziomu na temat zadany przez prowadzącego. Znajdą się tu przede wszystkim **wymagania** formalne, jakie powinno spełniać takie **sprawozdanie**, jak również uwagi mające pomóc w tworzeniu czytelnej i precyzyjnej dokumentacji technicznej nie tylko dla celów laboratorium, ale w pracy zawodowej. W naturalny sposób w tekście znalazły się również uwagi dotyczące stylu programowania.

Fragmenty rozwijające lub komentujące podane wcześniej informacje oznaczone są mniejsza czcionką.

Fragmenty nie będące bezpośrednio wymaganiami dotyczącymi dokumentacji, lecz podające zasady wynikające z inżynierii programowania, napisano mniejszą czcionką pochyłą.

Cały dokument na budowę wzorcowego sprawozdania, które omawia, a treść każdego rozdziału precyzuje wymagania dotyczące tego właśnie rozdziału.

Każdy rozdział zawiera:

- Ogólna definicję zawartości i przeznaczenia rozdziału
- Sugerowany układ treści – wymaganie dotyczące sprawozdania
- Komentarze na temat stylu programowania lub pisania dokumentacji

---

Strona tytułowa powinna zawierać następujące informacje:

- Nazwę laboratorium (*Laboratorium Programowania Komputerów*),
- Temat zadania,
- Imię i nazwisko autora programu,
- Nazwę kierunku, semestr, numer grupy i sekcji, i związaną z tym nazwę użytkownika (np. Informatyka, sem.3, gr.5, sekcja 5, PKC-55),
- Imię i nazwisko osoby prowadzącej zajęcia,
- Ścieżkę dostępu do końcowej wersji oprogramowania na dysku sieciowym.
- Datę sporządzenia sprawozdania

Przykładowa strona tytułowa znajduje się na końcu dokumentu.

Umieszczenie tych informacji w czytelnej postaci usprawnia proces oceniania i przechowywania prac. Wyjaśniamy, że powyższe wskazówki dotyczące strony tytułowej nie są wymaganiami o charakterze merytorycznym, jakie mogą wpłynąć na ocenę, a jedynie sugestiami porządkowymi. Podobnie estetyka dokumentacji (wyłączając być może skrajne przypadki ☺), czy jej układ graficzny nie jest przedmiotem oceny.

1. Temat

Tu podajemy temat zadania w formie, jaka została podana przez prowadzącego.

Przykłady:

- "Napisać program, który wyprowadzi na wyjście 100 najmniejszych liczb pierwszych posługując się algorytmem zwanym *sito Eratostenesa*.";
- "Napisać program obsługi małej biblioteki książek (kaset, CDs) umożliwiający dodawanie, usuwanie, przeglądanie, poszukiwanie rekordów. Dane przechowywane w pliku. Obsługa w trybie tekstowym/ za pomocą menu/ w trybie graficznym.";

- “Napisać grę strategiczno-zręcznościową typu StarCraft działającą w sieci, pod kontrolą Windows 95/ w Linuksie, z możliwością edycji zadań, terenu i scenariusza i listą zwycięzców. Wzorować się na rozwiązaniu firmy ...”

2. Analiza, projektowanie

Rozdział zawiera sprawozdanie z czynności wykonanych **przed** przystąpieniem do pisania kodu. Jest to ważny etap, którego nie należy opuszczać, ani zaniedbywać nawet w prostych zadaniach laboratoryjnych.

2.1. Algorytmy, struktury danych, ograniczenia specyfikacji

W rozdziale tym należy tu zastanowić się, wybrać i uzasadnić **odpowiedni** wybór:

- struktury danych:
  dane można przechowywać w strukturze statycznej, o ograniczonym z góry rozmiarze, jak łatwa w użyciu tablica czy rekord, albo w strukturze dynamicznej (lista, drzewo, graf), która pozwala przechowywać dowolnie dużo danych, lecz operacje wykonywane są wolniej, a jej użycie wymaga dużej uwagi i wprawy, ponieważ błędy w gospodarce pamięcią dynamiczną mogą mieć katastrofalne skutki;
- algorytmu:
  algorytmy różnią się złożonością czasową i pamięciową, czasem wymagają pewnej postaci danych czy wiedzy o nich, które to warunki w konkretnym zadaniu wykluczają pewne algorytmy, a faworyzują inne.
- ograniczeń specyfikacji:
  W ramach analizy można rozszerzyć lub ograniczyć specyfikację, co również wpłynie dobór algorytmów i struktur danych. Dobrze zaprojektowany program to nie ten, który koniecznie zawiera wszystkie funkcje, jakie projektant wymyślił, i wykonuje wszelkie możliwe operacje na danych, jakie kiedykolwiek mogą się przydać. Funkcjonalność programu (zestaw jego możliwości) powinna być określona w oparciu o oczekiwania użytkownika, jak również czas i środki, jakie będą na jego opracowanie przeznaczone. Natomiast wszelkie ograniczenia możliwości programu powinny być wyraźnie zaznaczone, szczególnie gdy istnieje możliwość, że użytkownik intuicyjnie uzna, że możliwości programu są szersze.

Uwaga: Wszystkie modyfikacje tematu czy koncepcje rozwiązania powinny być na **bieżąco** konsultowane z prowadzącym sekcję.

Rozdział niniejszy, jako sprawozdanie z etapu analizy, może zawierać dyskusję porównującą możliwe rozwiązania. Natomiast wszystkie **kolejne** rozdziały dokumentacji dotyczyć już będą **gotowego produktu** i powinny jednoznacznie opisywać to, co zostało zrobione. Błędem jest umieszczanie rozważań czy uzasadnień w rozdziałach dotyczących specyfikacji gotowego produktu.

W dużych projektach informatycznych na etapie analizy decyduje się o systemie operacyjnym, modelu maszyny czy komercyjnej bibliotece, którą użyje się do rozwiązania problemu. Np. w przypadku kompleksowej informatyzacji przedsiębiorstwa, oddziału firmy, banku istnieje możliwość wpływania na decyzję o zakupie dużego komputera mainframe*, czy konkretnego typu sieci i jej organizacji. Niestety w praktyce znaczenie częściej spotykamy się z sytuacją, gdy pewne części systemu (platforma sprzętowa lub użyty język i biblioteka) jako kluczowe i eksploatowane od lat, muszą pozostać niezmienione. Ogranicza to i utrudnia etap projektowania, choć nie zmniejsza jego wagi.*

2.2. Analiza problemu, podstawy teoretyczne

Jeśli do rozwiązania problemu niezbędne było wykorzystanie podstaw teoretycznych opartych na pewnej dziedzinie wiedzy, jak matematyka lub fizyka (np.: obliczanie całki oznaczonej metodą trapezów lub wykreślanie wykresu funkcji; wykreślanie trajektorii ruchu kuli armatniej, bądź symulacja odbijania bil na stole bilardowym), należy te podstawy (wzory, stałe, twierdzenia, modele) przytoczyć i krótko przybliżyć, na poziomie popularnonaukowym, wystarczającym do zrozumienia działania programu.

Jeżeli program nie korzysta z żadnych podstaw teoretycznych, rozdział ten jest zbędny.

2.3. Analiza obiektowa

Jeżeli program projektowany jest obiektowo, analiza powinna skupiać się głównie na projekcie klas, ich pól i metod, klas dostępu do składowych, powiązaniach między klasami (dziedziczenie) i wyznaczeniu funkcji wirtualnych.

3. Specyfikacja zewnętrzna

Rozdział ten jest inaczej **instrukcją użytkownika** [*user's manual*] i – mówiąc najogólniej – powinno się tu znaleźć wszystko to, co przeciętny użytkownik powinien się dowiedzieć o programie w celu jego prawidłowego użytkowania. Specyfikacja zewnętrzna dotyczy interfejsu[*interface*] użytkownika, sposobu obsługi programu, danych wejściowych (zadawanych) i wyjściowych (otrzymywanych).

W instrukcji użytkownika ważną częścią jest tzw. *troubleshooting*, czyli wyjaśnienia, jak radzić sobie w przypadku najczęściej pojawiających się problemów lub niejasności w działaniu programu. Do tego rozdziału sięga użytkownik, gdy ma jakiś problem, więc bardzo przydatny może okazać się skorowidz, który na podstawie określonego słowa kluczowego odsyła do odpowiedniego akapitu (-ów) instrukcji.

Nie umieszczamy tu informacji, **jak** coś zostało zaprogramowane, w jakiej tablicy, strukturze czy pliku przechowywane są dane, jaka jest ich wewnętrzna reprezentacja – te wszystkie informacje należą do następnego rozdziału. Jeżeli jednak użytkownik jest świadomy istnienia pliku z danymi, bo ma możliwość pobierania danych lub zapisywania wyników w pliku o podanej nazwie, **należy** tu opisać format danych w tym pliku i jego przeznaczenie.

3.1. Obsługa programu

Następujące kwestie powinny być w tym rozdziale wyraźnie podane:

Rozdział ten opisuje sposób posługiwania się programem. Opis ten z oczywistych względów będzie zupełnie inny dla programu wsadowego, korzystającego tylko z parametrów wywołania, a inny dla edytora tekstu czy grafiki, obsługiwanego interaktywnie przy pomocy interface'u typu GUI, posiadającego menu i paski narzędzi. Jednak w opisie obsługi prawie każdego programu powinny się znaleźć następujące ścisłe informacje:

- Klawisze – jakimi klawiszami wywołuje się poszczególne akcje [*shortcuts* – skróty klawiaturowe], wybiera elementy menu, odpowiada na zapytania programu; które klawisze są w określonych kontekstach czy momentach przyjmowane/ interpretowane/ ignorowane;
- Dane wejściowe – wymagany format, ograniczenia na ilość rekordów, zakresy wartości;
- Kontrola (lub jej brak) poprawności danych wejściowych, zachowanie programu w przypadku wykrycia danych nieprawidłowych. Uwaga: program **musi** prawidłowo

przetwarzać dane poprawne (zgodne ze specyfikacją). Natomiast dane z nią niezgodne **powinny** być wykrywane i odpowiednio traktowane przez program; w przypadku programów niezabezpieczonych mogą one powodować nieprawidłową pracę – jednak w obu przypadkach taka sytuacja **musi** być w dokumentacji opisana.

Z punktu widzenia definicji poprawności programów można stwierdzić teoretycznie, że program uznany za poprawny powinien poprawnie przetwarzać poprawne dane (czyli dane opisane w specyfikacji), natomiast jeżeli specyfikacja nie przewiduje pewnych przypadków danych wejściowych, zachowanie programu dla danych spoza dziedziny może być dowolne. Z drugiej strony kryterium dobrego stylu programowania wymaga, żeby program był zabezpieczony przed danymi spoza dziedziny, tzn. wykrywał je i reagował odpowiednim komunikatem, a w szczególności nie kończył się w takim przypadku błędem.

W opisie programu wsadowego trzeba położyć nacisk na opis parametrów, składnię prawidłowego wywołania i reakcję programu na błędne wywołanie.
W przypadku programu obsługiwanego interaktywnie powinno się zamieścić opis organizacji ekranu, rozmieszczenia elementów sterujących i menu.

3.2. Format danych wejściowych

Jeżeli program pobiera dane wejściowe z pliku (lub w trybie interaktywnym z klawiatury), sprawozdanie powinno zawierać pełen opis formatu tych danych zawierający:

- **Ilość** danych
  (konkretnie ograniczona/ ograniczona pamięcią operacyjną/ nieograniczona)
- **Typ** każdej czytanej wartości i jej akceptowaną postać (np. liczba rzeczywista postaci *[-]dddd.ddd* lub *[-]dddd.dddE+dd*)
- **Zakres** lub dopuszczalne wartości, znaki dla każdej czytanej wartości (np.: liczba całkowita dodatnia; całkowity numer istniejącego rekordu od 0 do określonego `max_nr`; litera ‘T’ lub ‘N’ (mała lub duża); litera oznaczająca opcję od ‘A’ do ‘K’; liczba rzeczywista oznaczająca prędkość w m/s z zakresu -1e+18..1e+18)
- Informację czy ilość, postać i zakres są przez program **sprawdzane** i jaka będzie reakcja na ewentualny błąd.

Program powinien być tak napisany, żeby użytkownik nie miał nigdy wątpliwości, na jakie dane oczekuje program, lub jaki błąd popełniono w danych wejściowych.

Jeżeli program wykorzystuje pliki konfiguracyjne (np. *ini*) lub inne, jakie użytkownik może edytować – opis ich formatu również musi się znaleźć w dokumentacji.

3.3. Komunikaty

W ramach opisu obsługi lub osobno należy wymienić wszystkie komunikaty produkowane przez program, wraz z ich wyjaśnieniami i, w przypadku komunikatów o błędach, zalecanymi reakcjami na te sytuacje.

W ramach laboratorium ze względów praktycznych do programu złożonego i pracochłonnego nie wymaga się tak szczegółowej i pracochłonnej dokumentacji, jak do programu prostego.

W rzeczywistych projektach informatycznych zastosowanie takiej ulgowej reguły byłoby błędem i niedbalstwem. Dokumentacja programu złożonego, o wielu modułach i funkcjach musi być proporcjonalnej objętości. Z drugiej strony dobrze, przejrzyście ze względu na użytkownika zaprojektowany program, nawet wyposażony w wiele funkcji, można uczynić łatwym i przyjemnym w obsłudze wykorzystując i dostosowując się niejako do ***intuicji*** *użytkownika.*

Standard graficznego interface GUI wprowadzono właśnie w celu ujednolicenia obsługi programów w środowiskach graficznych.

Takie powtarzające się elementy interface'u użytkownika, jak okna dialogowe, powinny być do siebie w miarę możliwości podobne, a zasady wyboru z menu jednorodne, tak aby nie było potrzeby omawiania każdego kontekstu użycia programu osobno, lecz aby wystarczyło jednokrotne podanie znaczenia pól, przycisków czy zasad wprowadzania danych

4. Specyfikacja wewnętrzna

Specyfikacja wewnętrzna jest **dokumentacją** techniczną **biblioteki** użytkowej. Jej zasadnicza część – omówienie zmiennych i funkcji, powinna być wzorowana na dokumentacji pisanej lub on-line'owej istniejących bibliotek, np. wbudowanym w środowisko programistyczne IDE Borlanda lub Microsoftu systemie pomocy. Rozdział ten przeznaczony jest dla **programisty**, który zna język, a jego ewentualnym zadaniem byłoby zmodyfikować nasz program lub użyć części funkcji we własnym programie, więc opisujemy tu precyzyjnie językiem technicznym tylko własne zmienne i funkcje. **Nie** opisujemy konstrukcji samego języka programowania ani użytych bibliotek standardowych (czyli nie przepisujemy dostępnego Helpa).
W specyfikacji wewnętrznej powinien znaleźć się opis algorytmu, struktury danych lub innej ciekawej, nietypowej lub niebanalnej koncepcji użytej podczas jego projektowania.

4.1. Zmienne

Należy opisać zmienne globalne i ewentualnie ważniejsze zmienne lokalne.
Opis każdej zmiennej musi zawierać:

- nazwę (identyfikator),
- typ,
- znaczenie (zastosowanie – licznik pętli, licznik obiektów, tablica kodów, zmienna sterująca wyborem w menu, wskaźnik do bufora z bitmapą),
- zakres lub wartości przyjmowane przez zmienną i ich znaczenie (0..n – numer maksymalnego elementu,

Przykłady:

`username`	zmienna globalna typu **`string[32]`**. Przechowuje nazwę użytkownika od chwili odczytania jej przez funkcję **`read_username`** aż do końca wykonywania programu.
`i`	zmienna typu **`int`**. Jest licznikiem głównej pętli **`for`** procedury **`xxx`**()
`glowa`	zmienna globalna typu **`el_listy*`**. Przechowuje wskaźnik na pierwszy element tworzonej przez program listy (książek, liczb, wierzchołków grafu). Jeśli lista jest pusta, ma wartość **`NULL`**.
`stan`	zmienna typu **`int`**. Steruje kontynuacją pętli w funkcji **`szukaj`**(). Możliwe wartości: **`-1`** gdy nie znaleziono, **`0`** **`-`** kontynuacja pętli, **`1`** **`-`** wyjście z pętli, **`2`** **`-`** wyjście z powodu błędu

4.2. Funkcje

Opis każdej funkcji musi zawierać następujące punkty:

- nagłówek (z parametrami);
- zadanie wykonywane przez funkcję (pobranie od użytkownika danych wejściowych – ilości osób i ich imion i umieszczenie w tablicy **osoby**);
- wszystkie jej argumenty i ich opis taki jak zmiennych (patrz wyżej);
- zwracany wynik, typ, zakres, wartości i ich znaczenie – nie należy tu zapominać o wynikach zwracanych w przypadku wykrycia błędnych parametrów aktualnych i efektach działania funkcji w nietypowych przypadkach.
  Jeżeli funkcja korzysta lub modyfikuje zmienne nielokalne nie przekazywane jawnie jako argumenty lub powoduje inne efekty uboczne (np. na ekranie), **musi** to być tu wyraźnie zaznaczone.
  Używanie przez funkcję danych nielokalnych, nie przekazywanych jawnie jako argumenty, mimo że dopuszczalne w większości języków wysokiego poziomu, jest w sensie stylu programowania nieeleganckie i potencjalnie niebezpieczne. Należy używać tej techniki tylko w przypadku, gdy przekazanie jawne nie jest możliwe, zaciemniłoby czytelność kodu lub w innych wyraźnie uzasadnionych przypadkach. Korzystanie z niej bez zastanowienia będzie traktowane jako usterka w projekcie programu.

Przykłady:

osoba* Czytaj_osobe(FILE *baza);
Funkcja alokuje dynamicznie strukturę typu **osoba** i czyta z pliku **baza** opis jednej osoby. Plik musi być otwarty do odczytu. **Wynik zwracany:** Jeśli operacja odczytu bądź alokowania nie powiedzie się, funkcja zwraca **NULL**, w przeciwnym wypadku adres utworzonej i wypełnionej struktury.

double cexp2(double x, double y);
Funkcja zwraca wartość x^y. Funkcja jest obliczana wyłącznie dla nieujemnych wartości **y**. Parametr **x** jest dowolną liczbą rzeczywistą. W przypadku wywołania z `y<=0` albo wystąpienia przepełnienia, funkcja wypisuje komunikat o błędzie i przerywa wykonywanie programu.

int strcmppl(const char *s1, const chat *s2, PlStandard std);
Funkcja porównuje napisy **s1** i **s2** w sensie ich porządku alfabetycznego, interpretując polskie znaki zgodnie ze standardem **std. s1** i **s2** muszą być wskaźnikami na ciąg znaków zakończony '\0', który może zawierać polskie znaki. **Zwraca**: **-1** gdy **s1<s2** **0** gdy **s1=s2** **1** gdy **s1>s2**

4.3. Program obiektowy

Jeżeli program wykonany jest techniką obiektową, oprócz opisu zmiennych i funkcji globalnych, w specyfikacji wewnętrznej powinien się znaleźć opis każdej klasy. Opis klasy obejmuje:

- rolę, jaką klasa odgrywa w projekcie (abstrakcyjna, bazowa itp.)
- opis wszystkich pól tak jak zmiennych
- opis wszystkich metod tak jak funkcji
- diagram klas
- diagram struktur danych

5. Wydruk/ postać elektroniczna

Gotowy program oddany do oceny lub eksploatacji powinien przede wszystkim być poprawny i przetestowany, jednak jego forma tekstowa, styl zapisu i postać wydruku również powinna spełniać pewne wymagania. Dokładniej takimi wymaganiami zajmuje się inżynieria programowania – tu tylko przypomnimy i zasygnalizujemy pewne sprawy.

5.1. Styl zapisu

- Wcięcia:
  powinny być wzorowane na przykładach zaczerpniętych z Helpa, oryginalnej dokumentacji kompilatorów lub wskazanych przez prowadzącego książek; powinny podnosić czytelność wydruku i podkreślać strukturę składniową programu
- Komentarze:
  powinny znajdować się przy deklaracji każdej zmiennej (za wyjątkiem przypadków banalnych i oczywistych); nad deklaracją każdej funkcji (podobna treść powinna się znaleźć w specyfikacji wewnętrznej); przy ważniejszych pętlach, skomplikowanych miejscach w algorytmach i we wszystkich innych fragmentach, które prawdopodobnie wymagałyby wyjaśnienia w czasie analizy tekstu programu.
- Nazewnictwo zmiennych:
  Nazwy zmiennych powinny być znaczące (związane z ich znaczeniem, zastosowaniem z programie), mogą być wielowyrazowe, np.: **`PredkoscKonc, IloscElementow, MaxIndex, wsk_do_poprzednika.`**
  Jednoliterowe nazwy (**`i, j, k, n, m, p, x, y`**) nadają się tylko do lokalnych indeksów, liczników pętli i wskaźników.
  Stałe preprocesora i inne można oznaczać `WIELKIMI` literami, np.: **`SIZE, MAX_INDEX, ESC, ARROW_UP`**
- Czcionki:
  Słowa kluczowe i identyfikatory występujące w opisie powinny być podawane w dokumentacji inną czcionką, niż zwykły tekst (rozważmy fragment: "dla każdego elementu listy od **`pierwszy`** do **`biezacy`** funkcja **`dodaj`**() dodaje do pola **`wartosc`** liczbę **`ile`**"). W przeciwnym wypadku identyfikatory mogą zostać zinterpretowane jako zwykłe wyrazy, a dokumentacja będzie mniej czytelna.
  Cały wydruk powinien być napisany czcionką **nieproporcjonalną** (najlepiej **`Courier`**).

*W ramach inżynierii programowania opracowano teorie określające wymagane proporcje między ilością tekstu a ilością komentarzy w dobrze napisanym programie. Jest to podejście być może zbyt rygorystyczne, jednakże z drugiej strony, bez wątpienia należy stwierdzić, że program napisany w ogóle bez komentarzy, jest napisany źle i **nie** będzie przyjmowany. Natomiast ilość niezbędnych komentarzy zależy od intuicji i doświadczenia programisty, które to grają podstawową rolę w dziedzinie, jaką jest inżynieria. Komentarze – często zaniedbywane – są ważnym środkiem pomagającym utrzymać czytelność kodu i ustrzec się błędów.*

*Opisy towarzyszące funkcjom w tekście programu i w specyfikacji wewnętrznej **mogą być** bardzo zbieżne, jako że traktują dokładnie o tym samym (komentarz może być bardziej zwięzły). Jednakże nie jest to powód, dla którego można by zaniedbywać lub pomijać którykolwiek z tych opisów.*

W dziedzinie nazewnictwa zmiennych istnieje kilka konwencji, np. w programowaniu pod Windows lansuje się tzw. notację węgierską, w której pierwsze znaki identyfikatora związane są z jego typem (np. lpDoc – long (far) pointer; hWnd – handle, szName – zero-terminated string).

5.2. Zawartość

Z powodów praktycznych wydruk nie powinien przekraczać kilku stron. Teksty mniejszych programów powinny być przytaczane w całości, natomiast w przypadku programów dużych na wydruku powinny znaleźć się przede wszystkim:

- Deklaracje typów, klas, zmiennych globalnych
- Deklaracje (prototypy) wszystkich funkcji
- Definicje (ważniejszych lub mniej banalnych funkcji)

Na wydruku dużego programu można opuścić fragmenty, które nie są istotne z punktu widzenia algorytmu lub są nie wnoszą niczego ciekawego, jak np.:

- Inicjalizacje zmiennych (chyba że mają znaczenie dla poprawności algorytmu)
- Fragmenty dotyczące interface'u użytkownika, interaktywnie pobierające dane lub zajmujące się przejrzystą prezentacja wyników na ekranie.

6. Testowanie

6.1. Dane testowe – uzasadnienie

Rozdział ten może być sprawozdaniem z procesu systematycznego testowania programu. Powinien zawierać odpowiednio przygotowane **dane testowe** wraz z formalnym **uzasadnieniem** ich doboru i ilości. Odpowiednie pliki z wymienionymi danymi testowymi powinny znajdować się wraz z tekstem programu na dysku sieciowym, tak aby można było powtórzyć proces testowania.

6.2. Wyniki

W dokumentacji umieszczamy wyniki działania programu dla konkretnych danych testowych z opisem W tym rozdziale należy umieścić wynik działania programu dla danych wejściowych. Można tu umieścić np. zrzut ekranu lub wydruk zawartości pliku wynikowego.

7. Wnioski

W niniejszym rozdziale można umieścić komentarze na temat pracy nad programem i inne spostrzeżenia. Można tu ustosunkować się do założeń umieszczonych w analizie zadania i stwierdzić, czy oczekiwania były trafne (np. że przewidywany algorytm okazał się odpowiedni do takiego charakteru/ takiej ilości danych lub że przyjęte rozwiązanie zbyt ograniczyło możliwości programu i że poszczególne moduły należałoby rozwijać).

Rozdział ten nie wpływa na merytoryczną ocenę (o ile całe, zdefiniowane w porozumieniu z prowadzącym zadanie zostało zrealizowane).

Uwagi końcowe:

Aktualna wersja przygotowywanego oprogramowania powinna być dostępna na dysku sieciowym (U:) w czasie zajęć i konsultacji.

Poniżej przykładowa strona tytułowa: Artur Migas

Miejscowość, data

Laboratorium Programowania Komputerów

Temat:
Program wykonujący określone zadanie

Autor: imię i nazwisko studenta
Kierunek, sem., grupa dziekańska, sekcja
użytkownik laboratoryjny
Prowadzący: Imię i Nazwisko prowadzącego zajęcia
Ścieżka: U:\katalog\program.c