Opracowywanie i weryfikowanie zadań
Podręcznik Użytkownika

1 Wstęp

Niniejszy dokument ma na celu standaryzację procedur dotyczących przygotowywania opracowań zadań z konkursów informatycznych.

1.1 Ważne adresy

System do zarządzania paczkami znajduje się w Internecie na stronie `http://www.oi.edu.pl/ sinol`.

`diks@mimuw.edu.pl`	Kierownik projektu
`aiwanicki@mimuw.edu.pl`	Administrator Sinola
`jrad@mimuw.edu.pl`	Kontroler jakości

1.2 Terminologia

identyfikator zadania — trzyliterowy skrót identyfikujący zadanie, najczęściej tworzony z tytułu zadania, np. dla zadania *Narciarze* — `nar`. Dalej w treści oznaczany jest przez *IDE*.

sygnatura zadania — 7-literowe oznaczenie zadania jednoznacznie identyfikujące zadanie, pierwsze cztery litery oznaczają pochodzenie zadania (nazwa konkursu), kolejne trzy liczby oznaczają numer zadania (dla danego pochodzenia). Przykład: `abcd023` — zadanie z konkursu `abcd` o numerze 23.

1.3 Cykl życia zadań

Zadania w trakcie swojego cyklu życia podlegają dwum fazom prac. Są to:

Redakcja i Opracowanie — sformułowanie treści zadania, przygotowanie zadania do wykorzystania w zawodach.

Weryfikacja — niezależne sprawdzenie poprawności opracowania.

Powyższe fazy są opisane szczegółowo w dalszej części tego dokumentu.

1.4 Postać paczek z zadaniami

W swoim cyklu życia zadania funkcjonują w formie tzw. „paczek", które zawierają komplet materiałów dotyczących danego zadania. Każda faza prac nad zadaniem polega na pobraniu całej paczki, zmodyfikowaniu jej i zwróceniu. Należy więc uważać, aby:

- Nie zniszczyć, ani nie usunąć materiałów powstałych w innych fazach prac.
- Nie umieszczać w paczce zbędnych elementów, np.: skompilowanych plików wykonywalnych, plików tymczasowych, pakietów LaTeX-a.
- Paczki z zadaniami zawierają pliki `makefile` wspomagające poszczególne fazy prac. Przed wysłaniem gotowej paczki można wykonać (w odpowiednim katalogu) polecenie: `make; make clean`.

Przy przesyłaniu, paczki z opracowaniami zadań powinny mieć format `tgz` (archiwum TAR skompresowane za pomocą programu gzip). Paczki powinny mieć nazwę będącą sygnaturą zadania (np. `abcd001.tgz`).

1.5 Postać dokumentów

Wszystkie dokumenty o których tu mowa, powinny być przygotowane w LaTeX-u, najlepiej jako dokumenty stylu `sinol.cls`. Szablony dostępne są w załączniku. Należy pamiętać, że nad tymi dokumentami będą pracować również inne osoby. Tak więc, należy zadbać o czytelność kodu źródłowego, a w szczególności:

- stosować wcięcia,
- wprowadzając polskie znaki narodowe używać kodowania iso-latin-2,
- zaczynać zdania od nowego wiersza.

Pewną wskazówką może tu być wygląd kodu źródłowego niniejszej instrukcji.

2 Redakcja

Redakcja zadania to przygotowanie treści zadania w wersjach językowych angielskiej oraz polskiej. Redakcja wchodzi w skład opracowania zadania. Czynności które należy wykonać podczas redakcji:

- Przygotowanie treści zadania: wersja polskojęzyczna powinna się znaleźć w pliku *IDE*`zad-pl.tex` a wersja anglojęzyczna w pliku *IDE*`zad.tex`.
- W oznaczeniach matematycznych należy używać liter alfabetu łacińskiego (a unikać liter greckich). Ten sam symbol nie może być używany w różnych znaczeniach. W liczbach wielocyfrowych należy wstawiać drobne odstępy co trzy cyfry, np. 1 000 000 000.
- Przygotowanie przykładowych plików wejściowych i odpowiedzi, a także rysunku (o ile ma zastosowanie). Rysunek powinien być przygotowany w postaci wektorowej, tak aby dało się go przekonwertować do formatu `eps`. Polecamy narzędzia `xfig` i `metapost`. Należy unikać wszelkich niestandardowych narzędzi i pakietów.
- Sprawdzenie poprawności pisowni.

3 Opracowanie

Opracowanie zadania polega na przygotowaniu materiałów koniecznych do jego automatycznego oceniania.

Czynności, które należy wykonać podczas opracowania:

- implementacja rozwiązań wzorcowych,
- przygotowanie zestawu testów,
- przygotowanie kilku rozwiązań mniej efektywnych (co najmniej jednego),
- przygotowanie kilku niepoprawnych rozwiązań (co najmniej jednego),
- przygotowanie programu do weryfikacji poprawności plików wejściowych,
- (dla zadań z niejednoznacznymi wyjściami) przygotowanie programu do weryfikacji odpowiedzi zawodników,
- przygotowanie dokumentu opisującego opracowanie w polskiej oraz angielskiej wersji językowej.

3.1 Opis opracowania

Jest to dokument zawierający następujące informacje:

- analiza problemu,
- opis rozwiązania wzorcowego (z dokładnym uzasadnieniem poprawności),
- opisy innych możliwych rozwiązań (w tym niepoprawnych),
- opis zestawu testów i proponowane czasy odcięcia,
- zestawienie które rozwiązania powinny przejść które testy.

Przykładowy dokument opracowania można znaleźć załącznikach. Wersja polskojęzyczna powinna się znaleźć w pliku *IDE*`opr-pl.tex`, anglojęzyczna zaś — w pliku *IDE*`opr.tex`.

3.2 Rozwiązania

Wszystkie rozwiązania powinny mieć następujące cechy:

- kod powinien być czytelny i dobrze skomentowany,
- należy unikać wymyślnych sztuczek programistycznych, zwłaszcza takich, które zmniejszają czytelność programów.

Wszystkie rozwiązania należy umieścić w podkatalogu `prog/`.

3.2.1 Rozwiązanie wzorcowe

Pliki z rozwiązaniami wzorcowymi powinny mieć nazwy postaci `prog/`*IDE*`[0-9]*.(java|c|cpp)`.

3.2.2 Rozwiązania mniej efektywne

Rozwiązania mniej efektywne to alternatywne implementacje rozwiązania wzorcowego, lub inne (w domyśle prostsze) rozwiązania zadania — np. o gorszej złożoności. Pliki z rozwiązaniami mniej efektywnymi powinny mieć nazwy postaci `prog/`*IDE*`s[0-9]*.(java|c|cpp)`.

3.2.3 Rozwiązania niepoprawne

Rozwiązania niepoprawne to rozwiązania, które na pierwszy rzut oka mogą się wydawać poprawne, ale takie nie są, np. niepoprawne heurystyki, rozwiązania nie pokrywające wszystkich przypadków, zachłanne itp. Opracowanie powinno zwierać co najmniej jedno takie rozwiązanie. Pliki z rozwiązaniami niepoprawnymi powinny mieć nazwy postaci `prog/`*IDE*`b[0-9]*.(java|c|cpp)`.

3.2.4 Język implementacji rozwiązań

Rozwiązanie wzorcowe i mniej efektywne należy zaimplementować we wszystkich językach programowania dopuszczonych w zawodach, czyli w C/C++ oraz Javie. Przy tym:

- jeżeli wykorzystanie STL-a nie jest istotne, to nie trzeba go implementować w C i C++, wystarczy implementacja tylko w jednym z tych języków (oczywiście nie korzystająca z STL-a),
- jeżeli wykorzystanie STL-a jest istotne w rozwiązaniu, to rozwiązanie w C++ powinno korzystać z STL-a, a rozwiązanie w C nie (w przypadku niemożności zapisania programu w C bez własnego implementowania drzew typu AVL czy czerwono-czarnych implementację rozwiązania w C można pominąć).

Rozwiązania niepoprawne można zaimplementować w dowolnym z dopuszczonych na zawodach języków programowania.

3.3 Testy

W trakcie opracowania należy przygotować zestaw testów.

3.3.1 Dobór testów

Testy powinny być tak dobrane by różnicowały poszczególne klasy (poprawnych) rozwiązań. Dobierając testy należy kierować się tym, żeby odróżnić rozwiązanie wzorcowe od wszystkich wymyślonych rozwiązań wolnych oraz błędnych. Odróżnienie to powinno być niezależne od języka programowania, w którym zaimplementowano dane rozwiązania.

3.3.2 Nazwy plików z testami

Pliki wejściowe powinny zostać umieszczone w podkatalogu `in/`, pliki wyjściowe (z poprawnymi odpowiedziami) powinny zostać umieszczone w podkatalogu `out/`. Pliki z danymi wejściowymi powinny nosić nazwy: *IDE* *nr*`.in`, gdzie *nr* oznacza numer testu (bez wiodących 0), np. `xyz0.in, xyz8.in, xyz12.in`. Test *IDE*`0.in` oznacza test przykładowy.

3.3.3 Format testów

Dane testowe powinny **dokładnie** odpowiadać składni podanej w zadaniu (dotyczy to również występowania białych znaków). Jako znak końca linii należy przyjąć znak `#10` (standard Unix). Dane powinny być zakończone pojedynczym znakiem końca linii.

3.3.4 Generator testów

Jeśli dane wejściowe mają rozmiar większy niż 1 MB konieczne jest dołączenie do opracowania programu/skryptu, który je generuje. Należy zadbać by program generujący był w pełni deterministyczny (w razie potrzeby należy inicjować ziarno generatora liczb losowych). Program powinien mieć nazwę `prog/`*IDE* `ingen.(c|cpp|java|sh)` i po uruchomieniu (bez żadnych argumentów) powinien wygenerować w *bieżącym* katalogu odpowiednie pliki z danymi wejściowymi.

3.3.5 Weryfikator danych wejściowych

Konieczne jest również przygotowanie programu do weryfikacji poprawności danych wejściowych. Program powinien nosić nazwę: `prog/`*IDE* `inwer.(c|cpp|java)` Program ten po uruchomieniu, powinien wczytać ze standardowego wejścia dane, oraz:

- jeśli dane są poprawne — wypisać `OK` i zakończyć działanie kodem 0,
- wpp. — wypisać `BŁĄD[w:` *nr wiersza*`, k:` *nr kolumny*`]:` *wyjaśnienie* i zakończyć działanie kodem różnym od 0.

W materiałach Jury dostępny jest przykładowy weryfikator danych wejściowych dla języka C++.

3.3.6 Weryfikator danych wyjściowych

W przypadku rozwiązań, w których istnieje wiele poprawnych odpowiedzi, opracowanie powinno zawierać weryfikator danych wyjściowych. Weryfikator powinien nosić nazwę: *IDE* `chk.(c|cpp|java)`.

System oceniający uruchamia weryfikator w następujący sposób:

IDE`chk` *plik_wejściowy odpowiedź_zawodnika podpowiedź* — standardowo jest to plik z prawidłową odpowiedzią

Weryfikator powinien wypisać odpowiedź w następującym formacie:

1. wiersz powinien zawierać jedno słowo:

   `OK` — jeśli odpowiedź jest poprawna, lub

   `WRONG` — w przeciwnym przypadku.

2. wiersz (opcjonalnie) powinien zawierać komentarz do odpowiedzi zawodnika (np. przyczyny uznania rozwiązania za niepoprawne).

Należy zwrócić uwagę, aby weryfikator wyjścia działał poprawnie nawet dla bardzo złośliwych danych (np. nie można nic zakładać o długości ciągów znaków znajdujących się w odpowiedzi zawodnika).

4 Weryfikacja

Czynności, które należy wykonać podczas weryfikacji:

- Sprawdzenie, czy treść zadania jest poprawnie sformatowana oraz sprawdzenie jej spell-checkerem.
- Przygotowanie alternatywnego, optymalnego rozwiązania zadania.
- Potwierdzenie poprawności składniowej plików wejściowych/wyjściowych.
- Potwierdzenie poprawności odpowiedzi z plików wyjściowych.
- Sprawdzenie, czy nie istnieje łatwe do znalezienia rozwiązanie, które przechodzi wszystkie testy i nie jest rozwiązaniem wzorcowym.
- Sprawdzenie, czy zestaw testów jest odpowiednio przygotowany. Jeśli okaże się to konieczne, należy dodać do zestawu własne testy poprawiające usterki.
- Opracowanie dokumentu zawierającego opis prac wykonanych podczas weryfikacji *IDE*`wer.tex`. Językiem obowiązującym w weryfikacji jest polski.
- W przypadku znalezienia nieprawidłowości w opracowaniu, należy je usunąć, oraz dokładnie opisać w *IDE*`wer.tex`.

Dokument *IDE*`wer.tex` powinien zawierać następujące informacje:

- listę wykonanych prac,
- krótki opis alternatywnego rozwiązania zadania, które zostało zaimplementowane podczas weryfikacji,
- listę znalezionych nieprawidłowości w opracowaniu,
- listę wykonanych modyfikacji w treści zadania i opracowaniu.

Szablon pliku *IDE*`wer.tex` można znaleźć w załącznikach.

5 Załączniki

5.1 Szablon treści zadania — redakcja

```
% W polskiej wersji językowej
\documentclass[zad]{sinol}
% W angielskiej wersji językowej
\documentclass[zad,en]{sinol}


  %% definicje Sinola
  \title{Bardzo Trudne Zadanie}
  \author{Foo Bar}    %% kto utworzył ten dokument
  \signature{abcd123} %% sygnatura
  \id{bar}            %% trzyliterowy skrót nazwy zadania
  \iomode{stdin}      %% Rodzaj opisów w przykładzie -- obecny standard

  %% Co się działo z tym dokumentem
  \history{KS, redakcja treści zadania}{2002.09.01}{1.00}
  \history{TW, zmiana definicji we/wyjścia}{2002.09.30}{1.01}

  %%
  % dodatkowe definicje LaTeX'owe, jeśli są konieczne
  %%
\begin{document}
  \begin{text}% Ten znak % jest istotny!

  % treść zadania...

    %% Ewentualny rysunek do przykładu
    \exampleimg{...}         % Rysunek do całego przykładu.
    \exampleinputimg{...}    % Rysunek tylko do wejścia.
    \exampleoutputimg{...}   % Rysunek tylko do wyjścia.
    %% Automatyczne złożenie przykładu
    \makestandardexample     % Przykład złożony w jednej kolumnie,
                             % rysunki po prawej stronie.
    \makecompactexample      % Przykład złożony w dwóch kolumnach,
                             % rysunki poniżej.

  \end{text}
\end{document}
```

5.2 Szablon dokumentu z opracowania

```
% W polskiej wersji językowej
\documentclass[opr]{sinol}
% W angielskiej wersji językowej
\documentclass[opr,en]{sinol}
  %% definicje Sinola
  \signature{abcd123}            %% Sygnatura
  \title{Bardzo Trudne Zadanie}  %% Tytuł
  \id{bar}                       %% Trzyliterowy skrót
  \author{Jan Kowalski}          %% autor opracowania
  \history{2002.10.01}{JK, przygotowanie opracowania}{1.00}
```

```
  %% tu można wstawić dodatkowe polecenia LaTeX'a
\begin{document}
  %% początek dokumentu
  \begin{text}% Ten znak % jest istotny!

    \section{Wprowadzenie} % oczywiście zmodyfikować w angielskiej
                           % wersji językowej
      ... %% Wprowadzenie i analiza problemu.

    \section{Rozwiązanie wzorcowe}
      ... %% Opis rozwiązania wzorcowego, uzasadnienie poprawności.

    \section{Inne rozwiązania}
      ... %% Opis innych rozwiązania które mogą zaproponować zawodnicy,
          %% o gorszych złożonościach, lub niepoprawnych.

    \section{Testy}
      ... %% Opis zestawu testów.
      \begin{tests}
        \test{0}{1s}{przykładowy test z treści zadania}
        \test{1}{1s}{prosty test poprawnościowy}
      \end{tests}

      ... %% Opis sprzętu na jakim dobrano limity czasowe.
          %% Uwagi dotyczące kalibrowania limitów czasowych.

    \section{Uwagi}
      ... %% Uwagi do opracowania.

    \section{Lista wykonanych prac}
      ... %% Lista prac wykonanych w ramach opracowania
          %% (istotna zwłaszcza, jeżeli wykonano jakieś nietypowe
          %% prace).

  \end{text}
\end{document}
```

5.3 Szablon dokumentu z weryfikacji

```
\documentclass[wer]{sinol}
  %% definicje Sinola
  \signature{abcd123}            %% Sygnatura
  \title{Bardzo Trudne Zadanie}  %% Tytuł
  \id{bar}                       %% Trzyliterowy skrót
  \author{Jan Kowalski}          %% Autor weryfikacji
  \history{2002.11.01}{JK, przygotowanie weryfikacji}{1.00}

  %% tu można wstawić dodatkowe polecenia LaTeX'a
\begin{document}
  %% początek dokumnentu
  \begin{text}% Ten znak % jest istotny!

    \section{Alternatywne rozwiązanie}
```

```
      ... % Krótki opis alternatywnego rozwiązania (prog/\ID wer.*)

    \section{Zmiany}
      %% Lista wykonanych zmian w opracowaniu

      \subsection{Zmiany w treści zadania}
        ...

      \subsection{Zmiany w danych testowych}
        ...

      \subsection{Inne zmiany}
        ...

    \section{Uwagi}
      ... % ewentualne uwagi do opracowania / zadania

    \section{Lista wykonanych prac}
      ... %% Lista prac wykonanych w ramach weryfikacji
          %% (istotna zwłaszcza, jeżeli wykonano jakieś nietypowe
          %% prace).

  \end{text}
\end{document}
```

5.4 Format opracowania — katalogi

Opracowanie powinno zawierać następujące katalogi:

`doc` — dokumenty opracowania, treści zadania, opisy rozwiązania, dokumenty weryfikacyjne,

`prog` — rozwiązania zadania,

`in` — dane testowe, pliki wejściowe,

`out` — dane testowe, pliki wyjściowe, ewentualne pliki z podpowiedziami.

5.5 Format opracowania — pliki

`Makefile` — główny plik Makefile,

`Makefile.in` — plik Makefile zawierający ustawienia specyficzne dla danego zadania,

`doc/`*IDE*`zad.tex` — treść zadania,

`doc/`*IDE*`rec.tex` — recenzja zadania,

`doc/`*IDE*`opr.tex` — opis opracowania,

`doc/`*IDE*`wer.tex` — opis weryfikacji,

`prog/`*IDE*`[0-9]*.(java|c|cpp)` — rozwiązanie wzorcowe zadania,

`prog/`*IDE*`s[0-9]*.(java|c|cpp)` — rozwiązania mniej efektywne, ale poprawne,

`prog/`*IDE*`b[0-9]*.(java|c|cpp)` — rozwiązania niepoprawne — na przykład heurystyki, czy rozwiązania opierające się na błędnych założeniach,

`prog/`*IDE*`[0-9]*wer.(java|c|cpp)` — rozwiązania zadania napisane przez weryfikatora,

`prog/`*IDE*`ingen.(java|c|cpp|sh)` — program generujący dane testowe,

`prog/`*IDE*`inwer.(java|c|cpp)` — weryfikator sprawdzający poprawność plików wejściowych,

`prog/`*IDE*`chk.(java|c|cpp)` — (tylko dla zadań, w których możliwe jest wiele poprawnych odpowiedzi) weryfikator poprawności odpowiedzi,

`in/`*IDE*`[0-9]+[a-z]*.in` — dane testowe — pliki wejściowe,

`out/`*IDE*`[0-9]+[a-z]*.out` — dane testowe — pliki wyjściowe,

`out/`*IDE*`[0-9]+[a-z]*.hnt` — dane testowe — pliki z podpowiedziami, służące do weryfikacji odpowiedzi zawodników.

5.6 Pomocnicze skrypt — Makefile

5.7 `Makefile`

Dostępne opcje:

`export` — przygotowanie archiwum opracowania gotowego do wysłania do Sinola,

`run` — uruchomienie rozwiązania wzorcowego na zestawie danych wejściowych z katalogu in/,

`run_%` — uruchomienie rozwiązania `prog/`*IDE*`%.(c|cpp|java)` na zestawie danych wejściowych z katalogu `in/`,

`verify` — przeprowadzenie kilku automatycznych testów badających poprawność formalną opracowania,

`clean` — usunięcie niepotrzebnych plików (kopie zapasowe, pliki wykonywalne, itp.),

`mrproper` — usunięcie wszystkich niepotrzebnych plików oraz plików które są generowane automatycznie (np. pliki wynikowe, pliki *.ps, *.pdf),

`ingen` — przygotowanie danych wejściowych generowanych automatycznie,

`inwer` — zweryfikowanie poprawności danych wejściowych,

`outgen` — wygenerowanie wzorcowych odpowiedzi dla danych wejściowych.

5.8 `doc/Makefile`

Dostępne opcje:

`all` — przygotowanie wersji PS i PDF dla dokumentów LaTex'owych zapisanych w katalogu `doc/`,

`ps`, `pdf`, `html` — przygotowanie odpowiednio wersji PS, PDF i HTML dokumentów,

`clean` — usunięcie wszystkich pomocniczych plików (np. kopii zapasowych), jednak pliki `*.ps`, `*.pdf` `*.html` pozostają bez zmian,

`mrproper` — dokładne wyczyszczenie katalogu `doc` — pozostają jedynie źródła dokumentów.

5.9 `prog/Makefile`

Dostępne opcje:

`all` — skompilowanie wszystkich programów umieszczonych w katalogu `prog/`,

`clean` — usunięcie wszystkich pomocniczych plików (kodów wykonywalnych, kopii zapasowych).