

# プログラミング課題の自動採点システム Hercules

### 1-1 自動採点システム Hercules の目的

- 大人数の学生を対象にして、効率的・効果的なプログラミング演習を実施するために教員を支援する
- 課題を採点する教師の負担を減らし採点の精度を上げる
- 演習時間中に学生のプログラムをほぼリアルタイムに採点し、進捗状況を可視化、効果的な指導を可能にする

所属	内務	近畿	東大	学生数	クラス	ラボ	課題数
コンピュータ	情報系 101	情報系 102	情報系 103	100	2	5	5
JAMU演習	工学部 工学系 104	JAMU演習	工学部 工学系	200	3	3	3
機械制御演習	M4, M4S, S20, S20F	M4, S20	工学部 工学系	50	1	2	2

### 1-2 プログラミング課題の例

### 1-3 プログラミング課題の特徴

- 課題の学習項目が具体的かつ明確である
  - work91では、構造体のメンバーの参照と構造体のコピー → 出力が正しいだけでは不十分
- 例題プログラムを修正してプログラムを作成する
  - 演習時間(180分)以内に全学習項目を網羅するため、4~5問のプログラムを作成する必要がある
  - プログラム構成のバリエーションは少ない
- 行数が10行~120行の小さなプログラム
  - work91.cの行数は34行
  - 関数への分解などのプログラムの設計法は問われない

### 1-4 プログラミング課題の問題点

- プログラミング課題で多い誤り
  - 仕様に従って誤り
  - 実行例が見えない、課題の使用の誤解、勝手に問題を作る
  - ソースコードに関する誤り
    - 関数や変数名の誤り、関数の引数の型や値、戻り値の型の誤り
    - else if を使わずに if を使ってしまう
  - 出力に関する誤り
    - 出力フォーマットの誤字脱字、余剰空白、不要な空白や奇怪な改行
    - 数値のフォーマットだけしか実行しない、エラーケースの検証をしない
    - ただし行末、別課題のコードと同じ、main関数の本体が空など
- 採点とモチベーション
  - 仕様と対応して作成されていないものを不通過とするのは学生のモチベーションを下げる
  - 誤りを発生して採点されないのも正しいプログラムを作成するモチベーションを高める

### 2-1 採点 = 検査+評定

- 検査: ソフトウェア工学の様々な技法を使って、プログラムの品質を計測する
- 評定: 複数の検査値を統合し、点数または等級を付ける

### 2-2 プログラムの検査技法

- ソフトウェアテスト技法を応用した多くの技法が提案されている。
- 静的な検査: プログラムを実行せずにソースコードを検査する
  - 行数や複雑度などのソフトウェアメトリクスを計測する\*
  - 正規表現などを使ってソースコードを検査する\*
  - 構文解析して生成した構文木を検査する、コンパイルエラーがあると実行できない
- 動的な検査: プログラムを実行して機能や効率を検査する
  - 正規表現などを使って出力メッセージを検査する\*
  - 関数やメソッドの呼び出しの回数や回数を検査する\*
  - UNIXシステムを使って、関数やメソッドの呼び出しを検査する
  - テストのコード覆盖率を測定する
  - 実行時間を計測する

### 2-3 検査項目の分類と種別記号

課題の仕様の分類

- プログラミング課題の仕様を5つに分類する
- 各分類に対応する検査項目を種別記号で示す
- 種別ごとに検査項目の重みも異なる

分類	説明	種別記号	重み
学習項目	新たに学ぶプログラムの構成要素および主要な能力の検査	■	8
必要条件	学習項目と入力能力を満たすために必要なプログラムの構成要素の検査	×	4
入力仕様	入力と出力のデータに関する検査	▲	4
エラー処理	誤った入力データに対する処理の検査	□	1
確保条件	コードと出力に関して望ましい条件の検査	△	1

### 2-4 課題の仕様と検査項目の分類

### 3-1 採点処理のフロー

### 3-2 課題の等級の定義

- 課題の完成度を示す等級を次のように5つに定める
- 点数は、パスした検査項目の種別ごとの重みと重みから計算する
- 等級5と4が合格である

等級	■	▲	□	△	○	1	点数
5	なし	なし	なし	なし	なし	なし	90点以上
4	なし	なし	なし	なし	なし	なし	80点以上
3	なし	なし	なし	なし	なし	なし	70点以上
2	なし	なし	なし	なし	なし	なし	60点以上
1	なし	なし	なし	なし	なし	なし	60点未満

### 3-3 システム構成

### 3-4 静的な検査の記述例

```

int main関数の定義で time 構造体を宣言していないか?
float memberFunction.sh # "AF" -# # "int" "main(SARGLIST)" NOT STRUCTURE IS TIME!!
# time 構造体を宣言しているか?
float structDef.sh # "AI" -# # "d TIME" "time"

# 仕様に与えられたメンバーを time 構造体に宣言していないか?
float memberFunction.sh # "AZ" -# # "d TIME" int width
float memberFunction.sh # "AX" -# # "d TIME" int height
float memberFunction.sh # "AM" -# # "d TIME" int x
float memberFunction.sh # "AY" -# # "d TIME" int y

# 仕様で与えられたメンバーを time 構造体に宣言しているか?
float member.sh # "AZ" -# # "d TIME" int width
float member.sh # "AX" -# # "d TIME" int height
float member.sh # "AM" -# # "d TIME" int x
float member.sh # "AY" -# # "d TIME" int y
    
```

### 4-1 進捗状況の可視化

- 不完全なプログラムの検査
  - 作成中のプログラムを採点するために、コンパイルエラーがあるプログラムも検査できればならない
- 複数レベルでの進捗の表示
  - 各クラス、各クラス、各学生という複数レベルの進捗を表示する
- 誤りの発見とリンク付け
  - 教員が指導をおこなうべき箇所を判断するために、検出した誤りや検査項目の種別順に表示する
- 高いパフォーマンス
  - 多くのプログラムを採点するために高いパフォーマンスが必要である。

### 4-2 進捗状況の可視化

各クラス間の進捗の差を知ることができる。

### 4-3 クラスおよび学生の進捗の提示

### 4-4 等級を用いた進捗推移グラフ