情報工学演習I
第１２回

C++の演習4（インライン展開）

授業の予定（後半）

#	月日	内容	担当者
7	11月13日	C言語の演習4 (ポインタの演算，列挙型)	内海
8	11月20日	C言語の演習課題	内海
9	11月27日	C++の演習1（クラス）	岩村
10	12月 4日	C++の演習2（クラスの継承）	岩村 (代理：谷川)
11	12月11日	C++の演習3 (関数のオーバロード)	岩村
12	12月18日	C++の演習4（インライン展開）	岩村
13	1月 8日	C++の演習5（仮想関数）	岩村
14	1月15日	C++の演習課題	谷川
15	1月22日	総合演習	谷川

今日の内容

- 第9回演習課題の解説
- インライン関数

第9回演習課題の解説

第９回演習課題（１）

- １. 以下の仕様を満たすプログラムを作れ
  - 以下の仕様を満たすクラスを持つ
    - 人の名前を保存することができる
    - 数学、理科、英語の点数を保存することができる
    - ３教科の点数を変更（上書き）できる関数と照会できる関数がある
    - ３教科の点数の平均を計算して返す関数がある
  - 上記のクラスを用いて、２人分のデータ（名前、３教科の点数）を順次コマンドラインから入力できる
  - 全員分の情報を入力した後、一人ずつ名前と平均点を表示する

第９回演習課題（１）の回答

読みにくいコードは
バグの温床

- ans9-1.cc
  - オブジェクトに名前や点数を渡す関数（set_nameやset_mathなど）を準備
  - オブジェクトから名前や点数をもらう関数（get_nameやget_mathなど）を準備
  - [欠点] main関数が込み入ってしまい、コードが読みにくい（何をしているのかを理解するのに考える必要がある）

名前の入力のコード

```
cout << "Enter a name: "; // 名前の入力
string name;
cin >> name;
person[i].set_name(name);
```

一時的に
変数を定義

第９回演習課題（１）の別解

- ans9-1_another.cc
  - クラスのget_nameやget_math関数で返す値を参照渡しにした（関数の宣言時、戻り値に「&」を付ける）

```
// 名前を調べる
string& get_name() {
  return name;
}
```

  - [利点] 参照渡しにすれば、関数を呼び出して普通の変数のように扱える
    → main関数がすっきりして、読みやすくなった

名前の入力のコード

```
cout << "Enter a name: "; // 名前の入力
cin >> person[i].get_name();
```

一時的な変数が不要

第９回演習課題（２）

- 2. 以下の仕様を満たすプログラムを作れ
  - 以下の仕様を満たすクラスを持つ
    - ２次元の座標を保持することができる
    - 座標の初期値はオブジェクト作成時に与える
    - 現在位置を更新（上書き）する関数を持つ
    - 現在位置を返す関数を持つ
    - 呼び出されると上下左右にそれぞれ移動する関数を持つ
  - コマンドラインから上下左右に移動する命令を受け取り、上記クラスの関数を呼び出して、移動させる
    - 命令の例：up, down, left, right
  - コマンドラインから現在位置を照会する命令を与えたときは、現在位置を返す

第９回演習課題（２）の回答

- ans9-2.cc
  - x座標とy座標をint型で宣言し、データメンバとした

```
class Coordinate { // 座標を扱うクラス
private:
  int x, y; // 座標
```

  - [欠点] 現在位置を返す関数を実装しようとすると、x座標かy座標しか返せない

```
 // 現在地のx座標を返す
 int ret_pos_x() {
   return x;
 }
```

第９回演習課題（２）の別解

- ans9-2_another.cc
  - 座標を表す構造体を導入した
  - [利点] コードが読みやすくなる

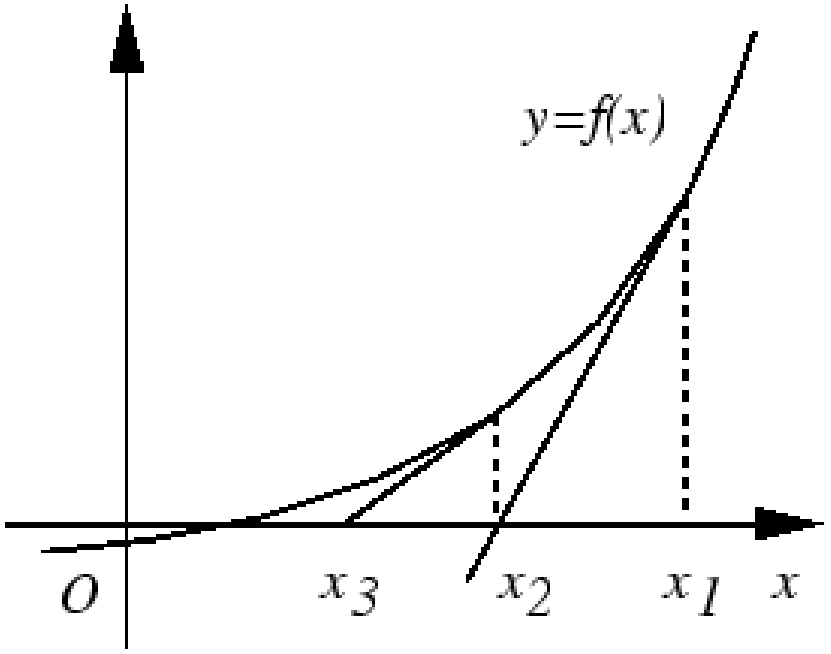
```
struct position {
  int x, y;
};

class Coordinate { // 座標を扱うクラス
private:
 position pos; // 座標
```

```
// 現在地を返す
 position ret_pos() {
   return pos;
 }
```

第９回演習課題（３）

- 3. ニュートン法で方程式を解くプログラムを作れ。ただし、以下の仕様を満たすものとする
  - 方程式は$ax^3+bx^2+cx+d=0$とし、a, b, c, dはコマンドラインから入力

ニュートン法：
曲線の接線とx軸の交点を求める処理を何度も繰り返して解を導く方法



何かを参考にした場合は、その旨を述べる。書いてない場合は０点

第９回演習課題（３）の回答

- ans9-3.cc
  - ニュートン法を計算するクラスを定義した
    （クラスの使用は必須ではないが、実はあると便利）

```
class Newton { // ニュートン法のクラス
private:
  double eps; // 終了条件(更新量の閾値)
  int mc; // 終了条件(更新回数)
  double a, b, c, d; // 関数の係数

public:
  // コンストラクタ
  Newton(double _eps, int _mc, double _a, double _b, double _c, double _d);
  double func(double x); // 関数の値
  double deriv(double x); // 関数を微分した値
  double solve(double x); // ニュートン法で解を解く
};
```

インライン関数

インライン関数

- 関数呼び出しで生じるオーバーヘッドを回避する方法
  - 関数呼び出しは処理のオーバーヘッドが生じる
    - 引数があれば余計にオーバーヘッドが大きくなる
  - インライン関数のコードは、呼び出し元のコードに埋め込まれる
  - C言語のマクロに似ている

```
#define PI 3.14159265358979
#define tasu(a,b) (a+b)
#define tasu2(a,b) a+b
```

ソースコードを置き換えてからコンパイルする

```
Int c=tasu(10,5);
```

→

```
Int c= (10+5);
```

```
Int c=3*tasu2(10,5);
```

→

```
Int c= 3*10+5;
```

インライン関数

実際にインライン化されるかどうかはコンパイラ依存

- 宣言方法
  - 関数の宣言の前にinlineを入れるだけ

```
int tasu(int x, int y) { // 足し算する関数
  return x+y;
}

inline int tasu_inline(int x, int y) { // 足し算する関数(インライン版)
  return x+y;
}
```

インライン関数

ex17_inline1.cc

```
#include <time.h> // 時間測定用
#include <iostream>
using namespace std;

int tasu(int x, int y) { // 足し算する関数
  return x+y;
}

inline int tasu_inline(int x, int y) { // 足し算する関数(インライン版)
  return x+y;
}

int main() {
  int sum;
  clock_t start, end; // 処理の開始時間と終了時間
  sum=0;
  start = clock(); // 開始時間測定
  for(int i=0; i<214748364; i++) {
    sum = tasu(sum,i);
  }
  end = clock(); // 終了時間測定
  cout << "Computation time without inline: " << end-start << endl;

  sum=0;
  start = clock(); // 開始時間測定
  for(int i=0; i<214748364; i++) {
    sum = tasu_inline(sum,i);
  }
  end = clock(); // 終了時間測定
  cout << "Computation time with inline: " << end-start << endl;

  return 0;
}
```

インライン関数（クラスの場合）

- 宣言方法：クラスの定義の中に関数の定義を書く

```
class tashizan {
public:
  int tasu(int x, int y); // 足し算する関数

  int tasu_inline(int x, int y) { // 足し算する関数(インライン版)
    return x+y;
  }
};

int tashizan::tasu(int x, int y) { // 足し算する関数
  return x+y;
}
```

インライン関数（クラスの場合）

ex18_inline2.cc

```
#include <time.h> // 時間測定用
#include <iostream>
using namespace std;

class tashizan {
public:
  int tasu(int x, int y); // 足し算する関数

  int tasu_inline(int x, int y) { // 足し算する関数(インライン版)
    return x+y;
  }
};

int tashizan::tasu(int x, int y) { // 足し算する関数
  return x+y;
}
```

```
int main() {
  tashizan t; // オブジェクトの生成

  int sum;
  clock_t start, end; // 処理の開始時間と終了時間

  sum=0;
  start = clock(); // 開始時間測定
  for(int i=0; i<214748364; i++) {
    sum = t.tasu(sum,i);
  }
  end = clock(); // 終了時間測定
  cout << "Computation time without inline: " << end-start << endl;
```

インライン関数（クラスの場合）

続き

ex18_inline2.cc

```
  sum=0;
  start = clock(); // 開始時間測定
  for(int i=0; i<214748364; i++) {
    sum = t.tasu_inline(sum,i);
  }
  end = clock(); // 終了時間測定
  cout << "Computation time with inline: " <<
end-start << endl;


  return 0;
}
```

演習課題

第１２回演習課題（１）

- 1. 以下の条件を満たすプログラムを作成せよ
  - 以下の仕様を満たす時を表すクラスを持つ
    - データメンバとして、時間、分、秒を表すものを持つ
    - メンバ関数を使用して、任意の時間、分、秒をオブジェクトに渡すことができる
    - メンバ関数を使用して、現在オブジェクトに保存されている時間、分、秒を受け取ることができる
    - メンバ関数を呼び出せば、自分のオブジェクトに保存されている時間、分、秒と別のオブジェクトが持つ時間、分、秒を足すことができる
  - 上記のクラスのオブジェクトを使用して、２つの時間、分、秒を設定して、足し算して結果を表示する

第１２回演習課題（２）

- 2. 以下の仕様を満たすプログラムを作成し、オブジェクトの深いコピーが実現できていることを示す
  - 以下の仕様を満たすクラスを持つ
    - データメンバとして、3次元配列を動的に確保する
    - コピーコンストラクタを持ち、オブジェクトの深いコピーが実現できる

提出に関して

- 提出するもの
  - ソースファイル(.ccまたは.cpp ファイル)
    - ファイル名はkadai1218_学籍番号_課題番号.cc（.cpp）
    - (Visual Studioの場合)
      ファイル名はkadai1218_学籍番号_課題番号_v.cc
      （.cpp）
  - 実行結果の出力と講義に関するコメント
    - .txt ファイルで、学籍番号、氏名を含む
    - ファイル名はreport1218_学籍番号.txt とする

提出に関して（続き）

- 提出期限
  - 1月15日（水）　００：００
- 提出方法
  - 授業支援システムから提出
- 注意点
  - ファイル名の命名規則が間違っているものは採点しない
  - コンパイルの通らないものは採点しない