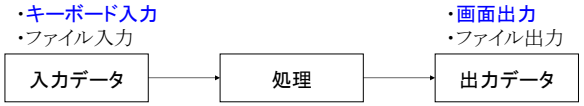
C言語を用いたプログラミング6 補足資料

Key Words:
- 標準関数，ユーザ関数
- 引数(argument)，戻り値(return value)
- プロトタイプ宣言(prototype)
- 値渡し(call by value)，アドレス渡し(call by reference)
- 変数の独立性，データ保護
- ポインタ(アドレスを記憶する変数)

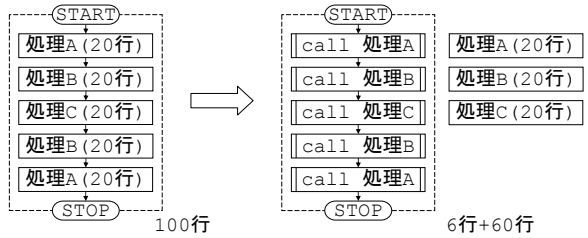
プログラム



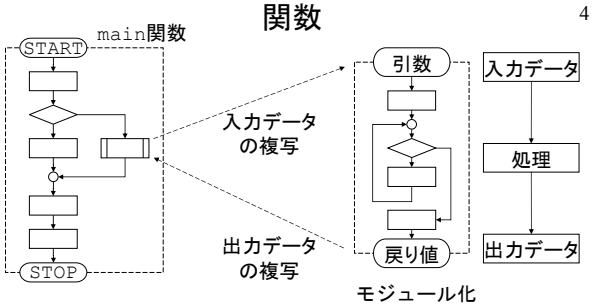
- データを記憶するための変数(データ構造)
  - 整数型
  - 実数型
  - 文字型
  - → 複数のデータを記憶 → 配列(数値リスト，ベクトル，行列，…)
- 処理をするための制御構造(アルゴリズム)
  - 順次
  - 選択
  - 反復
  - → モジュール化 → *関数*

関数の目的

- 大規模処理を小規模処理へ分割し，全体処理を明確化できる．
- 重複処理を効率化できる．
- 処理をブラックボックス化できる．(処理の隠蔽)
- 処理を分担して開発できる．
- 開発済み処理が容易に再利用できる．
  → 開発効率，保守性，再利用性の向上



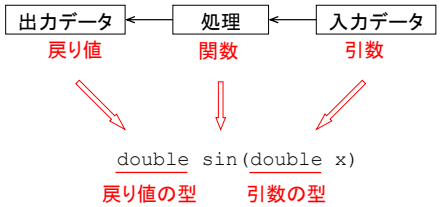
関数



- 入出力データの仕様が決まれば，独立に開発できる．
- 仕様に一致しているか検査が必要．→ プロトタイプ宣言
- 関数の仕様（関数名，引数の仕様，戻り値の仕様）

標準関数(例:数学関数)

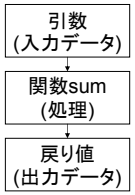
表2.1	書式，関数の仕様	定義
sin(x)	double sin(double x)	math.h
sqrt(x)	double sqrt(double x)	
pow(x,y)	double pow(double x, double y)	

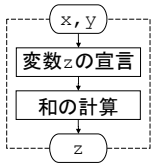


プログラム6.1

次の設計仕様を満たす関数を作成し，実行確認するプログラムを作成しなさい．

関数名	sum
機能	2つの整数の和を求める
引数	数値x,y (int型)
戻り値	合計z (int型)

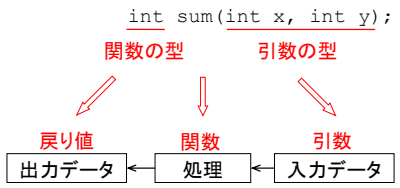





```
int sum(int x, int y)
{
  int z;
  z = x+y;
  return z;
}
```

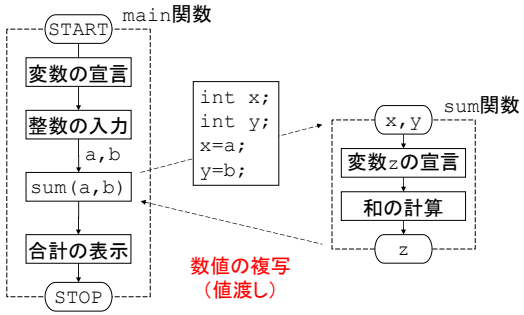

プロトタイプ宣言

- 関数の入出力データの仕様をユーザーに公開する．
- 仕様に一致しているか否かを検査する． → バグの低減



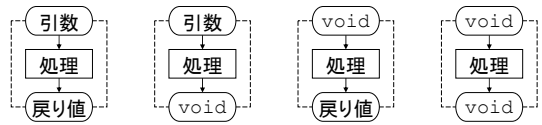
(用語) bug:虫，欠陥
debug: 害虫を取り除く．欠陥を治す．

プログラム6.1



- sum関数内のx,yは，main関数内のa,bに対して独立．
- a,bの値は保護される．(プログラム6.2参照)

「引数」や「戻り値」が無い場合

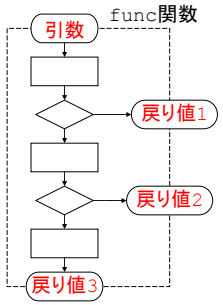



```
型 func(型 引数) {
  処理
  return 戻り値;
}
```

```
void func(void) {
  処理
  return;
}
```

(用語) void:空所，空き，無効にする

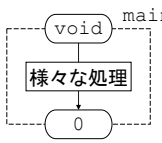
複数のreturnによる戻り値の切り替え




```
型 func(型 引数) {
  処理
  if (...) return 戻り値1;
  処理
  if (...) return 戻り値2;
  処理
  return 戻り値3;
}
```

return: 関数の中断，値の出力
break: 反復の中断

main()も一つの関数



戻り値	意味
0	プログラムが正常終了
0以外	異常終了

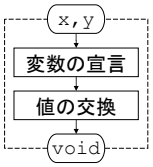
→ 関連：8章

プログラム6.2

与えられた2つの実数を交換する関数を作成しなさい．
この関数の場合，呼出側のデータが交換できないこと，
すなわち，呼出側のデータが保護されることを確認しなさい．

関数の設計仕様

関数名	swap
機能	2変数の値を交換する
引数	数値x,y（倍精度実数型）
戻り値	なし

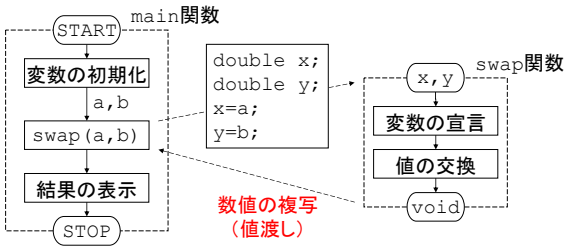


プロトタイプ宣言 ⇩

```
void swap(double x, double y);
```

（用語） swap：交換

プログラム6.2



・swap関数内のx,yは，main関数内のa,bに対して独立．
（メモリ上の記憶場所が異なる）
・a,bの値は保護される．

値渡しの問題点と解決策

関数の特徴：
・変数は関数毎に独立 （変数の独立性，ローカル変数）
・引数や戻り値は値のコピー （値渡し）

問題点：
・呼出側の値を関数側で変更できない． （データ保護）
・returnでは複数の値を戻せない．

解決策：
呼出側：データの場所（アドレス）を伝える．
関数側：その場所（アドレス）を記憶し，処理する

ポインタ変数
（アドレスを記憶する変数）

引数による参照（アドレス）渡し

引数によるアドレス渡し：
・呼出側のデータの場所（アドレス）を関数へ伝えること．
・関数側では，その場所をポインタ変数に記憶し，
その場所のデータを処理する．

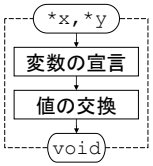
利点：
・複数の処理結果を呼出側に戻す（得る）ことができる．
・配列を渡す場合，先頭アドレスを渡せばよい．（連続した領域）

プログラム6.3

指定場所（アドレス）のデータを交換する関数を作成しなさい．

関数の設計仕様

関数名	swap
機能	指定場所（アドレス）に 記憶された値を交換する
引数	記憶場所x,y（データのdouble型）
戻り値	なし

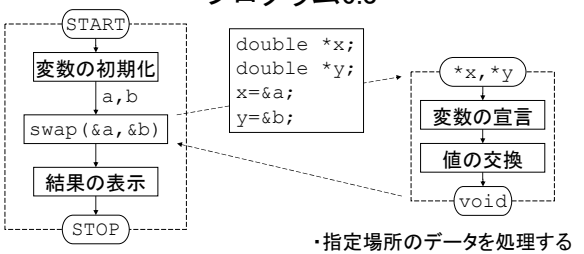


プロトタイプ宣言 ⇩

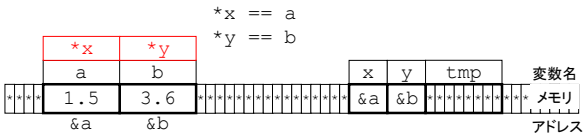
```
void swap(double *x, double *y);
```

「ポインタ変数」の記号「*」は乗算ではないことに注意する．
通常の変数と区別するために記号「*」を付けている

プログラム6.3



・指定場所のデータを処理する



ポインタ変数

変数の種別	意味	宣言方法の例	値
自動変数	値を記憶 する変数	int a;	a
ポインタ変数 (pointer)	アドレスを 記憶する変数	int *p;	*p

pにアドレスを記憶し，その場所のデータ(*p)を処理する

呼出側

```
int main(void) {
  int a;
  double b;
  ...
  func(&a,&b)
  ...
}
```

関数側

```
func(int *x, double *y) {

}
```