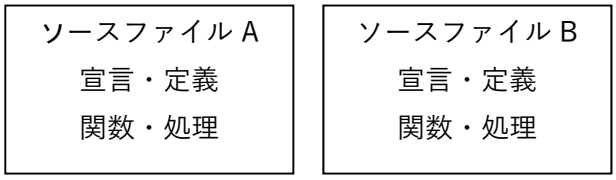
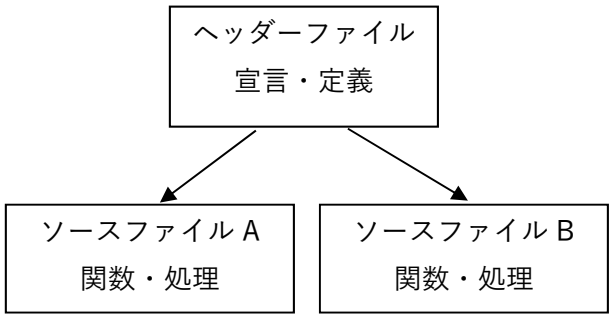
【ヘッダーファイルとソースファイル】

C 言語ではソースコードを「宣言・定義」と「関数・処理」に分け、「宣言・定義」はヘッダーファイル（拡張子 h）に、「関数・処理」はソースファイル（拡張子 c）に記述するのが一般的です。



このような形のソースファイルを



このような形に分類します。

＜自作ヘッダーファイルのインクルード＞

```
#include "ヘッダーファイル名.h"
```

ヘッダーファイルはインクルードして使用します。
stdio.h 等の標準関数のヘッダーファイルを読み込む場合は「#include <stdio.h>」とファイル名を<>で括ります。
<>を使用した場合、ヘッダーファイルのパスはコンパイラで指定された標準パスのみを参照しますが、""を使用した場合は標準パス＋カレントディレクトリ内も含まれます。
よって自作ヘッダーファイルを読み込む場合は""で括ります。

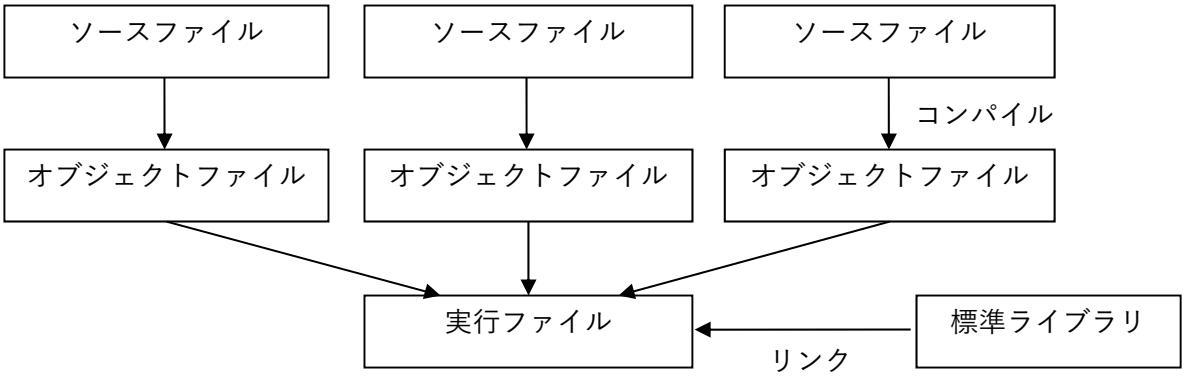
C 言語では関数ごとにソースファイルを分けて記述し、各関数共通で使用するインクルード文、関数プロトタイプ宣言、グローバル変数の宣言、構造体の定義、列挙体の定義等はヘッダーファイルに記述します。

Copyright 2018 DoubleQuotation Limited.
20181225

【ビルド】

ソースファイルから実行ファイル（拡張子 exe）を作成する流れは

1. ソースファイルを作る。
2. ソースファイルをコンパイルしてオブジェクトファイル（拡張子 o もしくは obj）を作る。
3. オブジェクトファイルとライブラリをリンク（結合）して実行ファイルを作る。



コンパイルとリンクを行い、実行ファイルを作成する作業を「ビルド」と称します。
「標準ライブラリ」は「標準関数のオブジェクトファイル」をまとめた物で、ソースファイルが無くても、オブジェクトファイルさえあれば実行ファイルを作成できます。

【GCC／WinGW のコマンド】

gcc -c ソースファイル名

-c は compile（コンパイル）のことで、コンパイルは行うがリンクは行わないオプションです。
ソースファイルをコンパイルして、オブジェクトファイル（拡張子 o）は作られますが、実行ファイル（拡張子 exe）は作られません。
複数のファイル名をスペースで区切れば、複数のファイルをコンパイルできます。

gcc -o 実行ファイル名 オブジェクトファイル名

-o は out（出力）のことで、実行ファイルを作るオプションです。
実行ファイルはオブジェクトファイルから作成されるため、オブジェクトファイルさえあれば、ソースファイルは必要ありません。

Copyright 2018 DoubleQuotation Limited.
20181225

【makefile】

makefile はビルドのコマンドを記述しておくファイルです。
ソースファイルの数が多くなってくると、ビルドするコマンド入力も大変になります。そこで、あらかじめビルドのコマンドを「makefile」に記述しておき、コマンドプロンプトから「makefile」を読むだけでビルドを行えるようにします。

1. テキストエディタで下記のように記述する。

```
ターゲット名:
    （Tab で空ける）コマンドを記述
```

＜例文＞

```
test:
    gcc -o exefile sauce1 sauce2 sauce3
```

2. ファイル名を「makefile」にして、ソースファイルと同じフォルダに保存する。
3. コマンドプロンプトから下記のように入力する。

```
make ターゲット名
```

＜例文＞

```
make test
```

これでビルドのコマンドが実行されます。
ターゲット名に「all」と記述した場合は「make」コマンドのみでビルドが行えます。

Copyright 2018 DoubleQuotation Limited.
20181225

【例文】

header.h

```
/* 標準関数のヘッダーファイルも共有 */
#include <stdio.h>
#include <string.h>

/* 構造体も共有 */
struct personal {
    char name[10];
    int  age;
};

/* 関数プロトタイプ宣言も共有 */
struct personal create(void);
void display(struct personal*);
```

main.c

```
/* header.h をインクルード */
#include "header.h"

int main(void) {
    struct personal val;
    val = create();
    display(&val);
    return 0;
}
```

create.c

```
/* header.h をインクルード */
#include "header.h"

struct personal create(void) {
    struct personal ret;
    strcpy(ret.name, "A 太郎");
    ret.age = 20;
    return ret;
}
```

display.c

```
/* header.h をインクルード */
#include "header.h"

void display(struct personal* arg) {
    printf("%s さん%d 歳です。¥n",
    arg->name, arg->age);
}
```

＜コンパイル＞

GCC／WinGW では「gcc -o *** main.c create.c display.c」

「makefile」を使用する場合は

```
all:
    gcc -o *** main.c create.c display.c
```

コマンドラインから「make all」か「make」で makefile を読み込ませます。

Copyright 2018 DoubleQuotation Limited.
20181225

【extern 変数】

extern 変数は別のソースファイルにあるグローバル変数を使いたい時に使用します。

main.c

```
#include <stdio.h>
int var;    /* グローバル変数 */
int main(void) {
    var = 10;
    func();
    printf("%d¥n", var);
    return 0;
}
```

func.c

```
/* main.c のグローバル変数 var を使用する */
extern int var;    /* extern 変数 */
void func(void) {
    var = 20;
}
```

＜実行結果＞

```
20
```

extern 変数はグローバル変数でもローカル変数でも宣言することができ、グローバル変数であればそのファイル内で、ローカル変数であればその関数内でのみ使用できます。
それぞれのファイル内で同名のグローバル変数を別々に使用したい場合、そのグローバル変数を static 変数にします。

Copyright 2018 DoubleQuotation Limited.
20181225