

プログラミング演習 I  
課題1  
10進数と2進数

—チェックディジットと全加算器—

# プログラミング演習 I

## 課題1 10進数と2進数

- 目的
  - 関数の定義の仕方や使い方, 標準関数を利用する仕組みを学ぶ
- 題材
  - (1回目) 整数 $\leftrightarrow$ 10進数の変換
  - (1回目) 学籍番号のチェックディジット生成法
  - (2回目) 整数 $\leftrightarrow$ 2進数の変換
  - (3回目) 2進数での加算
- 課題は5つ

# 課題概要

本日作成

2回目～3回目

- 課題(1-1)
  - 学籍番号(整数)の10進6桁の各桁の数を取り出す
- 課題(1-2)
  - 学籍番号(整数)の10進6桁の各桁の数からチェックディジット生成

- 課題(1-3)
  - 0～255の整数の2進8桁の各桁の数を取り出す
- 課題(1-4)
  - 8桁の2進数から、それが表す整数を求める
- 課題(1-5)
  - 2つの8桁の2進数の和を計算する.

# 5つの課題のプログラムはそれぞれ 保存しておく

プロジェクト名, ソースプログラム名の例

- 課題(1-1)
  - プロジェクト名 p1\_1
  - ソースプログラム名 p1\_1.c
- 課題(1-2)
  - プロジェクト名 p1\_2
  - ソースプログラム名 p1\_2.c
- 各課題毎に, 別のプロジェクトにすること

# 学籍番号のチェックディジット

- テキストp.6の解説を読み, p.1の計算を行う
- p.1の計算を行うプログラムを作成することが今日の課題(1-1)と(1-2)
- 今日の目標: TAが出題する学籍番号について, 課題(1-2)のプログラムを用いてチェックディジットを確認すること.

# 課題(1-1)

学籍番号(整数)の入力に対し, 1の位~100,000の位の数(10進数1桁目~6桁目の数)を求め, 出力せよ.

1. キーボードから学籍番号を入力し, int型変数に格納
2. 1の位~100,000の位について, 各桁の数を求め, それぞれ, int型変数に格納
3. 各桁の数を出力

# 課題(1-1)

- プログラムの構造
  - main関数のみ

```
#include <stdio.h>
int main(void)
{
    変数宣言
    文
    . . . .
    return 0;
}
```

プログラム作成上の指針(1)

```
int n0, n1, n2, n3, n4, n5;
int n;
```

プログラム作成上の指針(2)

```
scanf ("%d", &n);
```

プログラム作成上の指針(3)

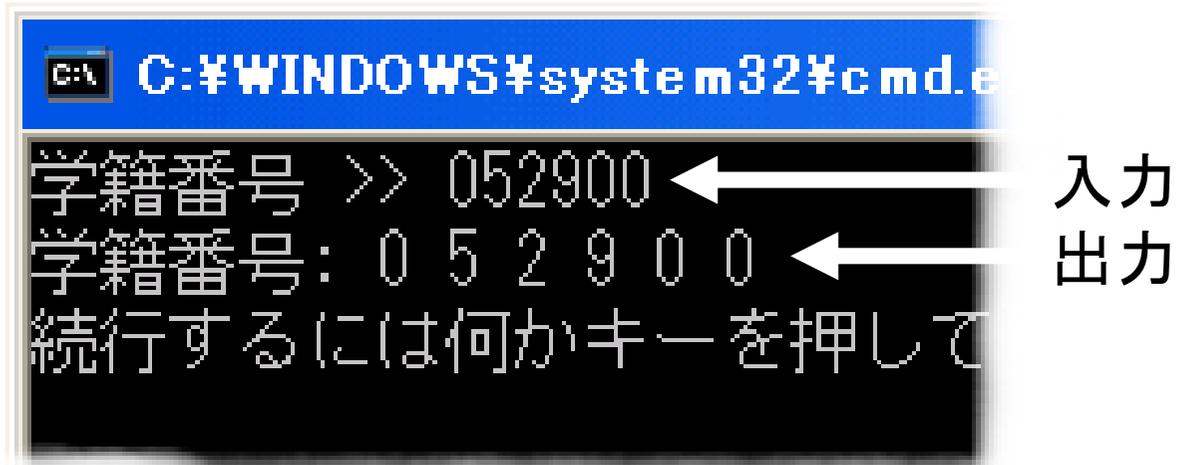
```
n2 = (n / 100) % 10;
n3 = (n / 1000) % 10;
```

プログラム作成上の指針(4)

```
printf ("%d %d . . . .");
```

scanfの使い方を調べてみよう

# 課題(1-1)のプログラムを作成しよう



```
C:\> C:\WINDOWS\system32\cmd.e  
学籍番号 >> 052900  
学籍番号: 0 5 2 9 0 0  
続行するには何かキーを押して
```

入力  
出力

## 課題(1-2)

- 課題(1-1)のプログラムに, 各桁の数から学籍番号のチェックディジットを求める関数を加え, 学籍番号と共にチェックディジットを出力するようにせよ.
- チェックディジットを求める関数を定義することが主な作業

# 課題(1-2)

- プログラムの構造
  - 関数check\_digitとmain関数

## 課題(1-2)のソースコード

### 課題(1-1)のソースコード

```
#include <stdio.h>
int main(void)
{
    変数宣言
    文
    . . .
    return 0;
}
```

```
#include <stdio.h>

/*関数check_digitの定義
をここに追加*/

int main(void)
{
/*課題(1-1) + 関数
check_digitの呼び出し*/
}
```

# ソースコードのコピーの仕方(1)

- Visual Studioを2つ起動するのが最も簡単
  1. Visual Studioを起動し, 課題(1-1)のpプロジェクト(ソリューション)を開く
  2. Visual Studioをもう一つ起動し, 課題(1-2)のプロジェクトを新規作成し, ソースファイルを新規作成する(通常のプログラム作成手順)
  3. ソースコードを(1-1)から(1-2)へコピー&貼り付け

# ソースコードのコピーの仕方(2)

- プロジェクトフォルダ構成を知っている人向け
  1. Visual Studioを起動し, 課題(1-2)のプロジェクトを新規作成する. ソースファイルは新規作成しない
  2. エクスプローラ(マイコンピュータのダブルクリック)で, 課題(1-1)のプロジェクトフォルダ内のソースファイル(~.c)を課題(1-2)のプロジェクトフォルダにコピーする. 必要であれば名前を変更する.
  3. Visual Studioの課題(1-2)のプロジェクトで, [プロジェクト]→[既存項目の追加]で, 2でコピーしたファイルを追加する

# 関数check\_digitの呼び出し

```
#include <stdio.h>
```

```
/*関数check_digitの定義をここに追加*/
```

```
int main(void)  
{
```

```
    char cd;
```

```
    . . .
```

```
    cd = check_digit(. . . .);
```

```
    printf(“%d %d %d %d %d %d %c\n”,  
           n5, n4, n3, n2, n1, n0, cd);
```

```
    return 0;
```

```
}
```

main関数は課題(1-1)をコピーし、  
以下を追加

char型→p. 6

関数の呼出し

%cは単一文字への  
変換を指定

# 関数check\_digitの定義

戻り値の型

パラメータ

関数の定義の仕方→p. 8

```
char check_digit(int n5, int n4, int n3,  
                 int n2, int n1, int n0)
```

リスト1

```
{
```

```
char cd; ← ローカル変数  
int m; ← ローカル変数
```

ローカル変数→p. 9

```
.....  
n5~n0からチェックディジットを求め、  
それを変数cdにセットする。  
.....
```

```
return cd; ← cdの値を戻り値として関数を終了
```

```
}
```

# チェックディジットの求め方

```
m = (7*n5 + 6*n4 + 5*n3 + 4*n2 + 3*n1 + 2*n0) % 11;
if ((m == 0) || (m == 1))
    cd = 'A';
else if (m == 2)
    cd = 'Z';
else if (m == 3)
    . . . . .
else if (m == 10)
    cd = 'B';
```

switch文を使ってもよい

## 解説: char型(p.6)

- 1文字を格納するための変数を宣言するときに指定する型
- 1バイト(8ビット)の符号付き整数型(通常, 2の補数表現)で-128~127の値をとる
- 文字を代入するときは, `c = 'A';` のように.
- `'A'` はAの文字コード
- `"A"` は1文字の文字列(2文字目が0の配列)

「'」はシングルクォート、  
「"」はダブルクォート、等と読む

# C言語における関数

- C言語の関数: 名前をつけた「ひとまとまりの手順」
  - その名前を呼ぶと, 「ひとまとまりの手順」を実行できる
  - 「ひとまとまりの手順」は入力をもつことができる
  - 「ひとまとまりの手順」は値を返すことができる
- (例) 数学関数sqrt: sqrtと名付けた平方根を計算する手順

$y = \text{sqrt}(3)$ ; 関数呼び出し「sqrtさん. 値3をよろしく」  
関数からの返答「1.7320508・・・だったよ」

# 関数定義（関数を作ること）

- 値を返すのであればその型を指定する（戻り値の型）
- 名前を付ける（関数名）
- 入力があれば、その個数だけ、型を指定し、個々の入力値に名前をつける（パラメータ宣言）
- ひとまとまりの手順を書く（{ }の中に）

```
戻り値の型 関数名 (パラメータ宣言)
{
    変数宣言等
    文
}
```

定義しただけ、  
他から呼び出して初めて実行される。

# 関数呼び出し(関数を使うこと)

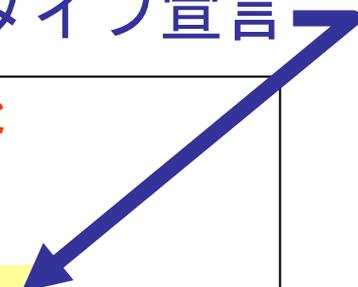
- 関数名(入力値, 入力値, 入力値, ...)
  - 「入力値, 入力値, 入力値, ...」は, 関数定義によってはない場合もある.
  - 「入力値, 入力値, 入力値, ...」の個数と型は関数定義と一致
  - 値をもつ(値をもたないこともある)

- 例

```
y = sqrt(3); // 3の平方根をsqrtに計算してもらい,  
            //その戻り値をyに代入  
sqrt(y);    // yの平方根を計算してもらうが,  
            //その結果は捨てている
```

# 関数定義を記述する位置(p.7)

関数プロトタイプ宣言



・ 呼び出す場所より前

```
#include <stdio.h>
```

```
int mul(int a, int b)
{
    return a*b;
}
```

```
int main(void)
{
    printf("%d¥n", mul(10, 20));
    return 0;
}
```

関数を呼び出す場所より前: OK

・ 呼び出す場所より後

```
#include <stdio.h>
```

```
int mul(int a, int b);
```

```
int main(void)
{
    printf("%d¥n", mul(10, 20));
    return 0;
}
```

```
int mul(int a, int b)
{
    return a*b;
}
```

関数プロトタイプ宣言があれば、  
関数定義は呼び出す場所より後でOK

# 関数の定義の仕方(p.8)

戻り値の型 関数名(パラメータ宣言)

{

変数宣言等

文

}

```
int add(int a,int b)
```

{

```
int c;
```

```
c = a + b;
```

```
return c;
```

}

パラメータ宣言 : 2つのint型の値を呼び出し側から受け取る。それぞれの名前をa, bとして、関数定義のブロック内で使用

処理結果を関数値としてセットし、関数を終える文

# 関数定義の仕方(p.8)

- 戻り値のない関数の場合

```
void add(int a,int b)    戻り値の型はvoid
```

```
{
```

```
    g_result = a + b;
```

```
}
```

return文はなくてもよい  
途中で終わりたいときは,  
return;

g\_resultはグローバル変数

# 関数の定義の仕方(p.8)

- パラメータ宣言がない場合

```
void add(void)          パラメータ宣言はvoid
{
    g_result = g_operand1 + g_operand2;
}
```

g\_result, g\_operand1,  
g\_operand2  
はグローバル変数

呼び出すときは add();  
「add;」ではない。

# 今日の目標

- 課題(1-2)まで行い, TA が出題した学籍番号について, 課題(1-2)のプログラムを用いてそのチェックディジットを解答し, チェックを受けること.