シラバス

9,10. 関数と記憶域クラス
関数と記憶域クラスにより多様なデータを処理する方法の理解
事前学修：授業スライド「第5回　関数と記憶域クラス」と教科書p.95～p.98, p.101～p.107の熟読。（60分）
事後学修：授業スライドの用語とその意味の理解。記憶域クラス，自動変数・外部変数・静的変数それぞれの意味と有効範囲と生成消滅タイミング，及びC言語による変数宣言方法を説明できること。（60分）
演習課題のプログラム及びレポートの完成と提出。（180分）

プログラミングの基礎及び演習

情報工学科

プログラミングの基礎　第5回

- 復習：関数（１）
  - 自作の関数
  - 関数と変数
- 関数　（２）
  - 変数の種類と記憶域クラス
  - 自動変数
  - 外部変数
  - 静的変数
- 変数のネーミング
  - 教科書 p.95～p.98, p.101～p.107

復習:関数の例 (1)

```
#include <stdio.h>
int keisan(int m, int n) {
    int a;
    a = m + n;
    return a;
}

int main(void) {
    int x;
    x = keisan(5, 3);
    printf("x = %d¥n", x);
    return 0;
}
```

(2) 5がmに，3がnにコピーされ，m+nの計算結果がaに格納

(3) 計算結果a=8を呼び出し元に戻す

(4) xに値8が格納される

(1) “5+3の計算をしてくれ”と関数keisanを呼び出す

復習:関数の例 (2)

```
#include <stdio.h>
int keisan(int m, int n) {
    int a;
    a = m + n;
    return a;
}

int main(void) {
    int x;
    x = keisan(5, 3);
    printf("x = %d¥n", x);
    return 0;
}
```

keisan関数の定義

main関数の他に，
自分で作成した関数
keisanを記述する．
keisanだけ，あるいは
mainだけでは動かない．

keisan関数を呼び出す

復習:関数の例 (3)

型宣言
関数を実行すると,
呼び出し元にint型の戻り値を返す

関数の名前
keisanという名前

引数ならび
int型の引数を２つ受け取る
keisanの内部では,
この２つの引数をm, nという名前の
変数として扱う

```
int keisan( int m, int n ) {
    int a;
    a = m + n;
    return a;
}
```

ボディ
２つの引数の値を加算する

p.88, 例題7

return文
呼び出し元に変数aの戻り値を返す

復習:関数と変数 (1)

- 局所変数
  - 関数の中だけで有効な変数

```
#include <stdio.h>
int keisan( int x, int y );
int main( void ) {
    int a, b, c;
    a = 2;
    b = 3;
    c = keisan( a, b );
    printf( "sum = %d¥n", c );
}
int keisan( int x, int y ) {
    int z;
    z = x + y;
    return z;
}
```

a,b,cはmain関数の中でのみ使用可能
a,b,cをkeisan関数の中で使うとエラー

x,y,zはkeisan関数の中でのみ使用可能
x,y,zをmain関数の中で使うとエラー

関数の中で計算に必要なデータは,
すべて引数として渡す必要がある
main関数のa,bを両方とも
keisanに渡さないと計算できない

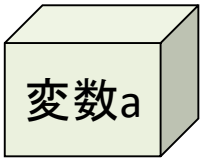
復習:関数と変数 (2)

- 局所変数は，各関数ごとに独立
  - 別の関数にある同じ名前の局所変数は互いに影響しない
    - main関数，関数func1と 関数func2で宣言している変数aは異なる変数．同様に変数bも異なる．

```
int main(void) {
    int a, b, …
    ・・・・
}
```

func1の変数a

func2の変数a



同じ名前だが，別の”箱”

```
int func1( int a) {
    int b;
    b = a + 5;
    return b;
}

int func2( int a ) {
    int b;
    b = 5*a;
    return b;
}
```

プログラミングの基礎　第5回

- 復習：関数（１）
  - 自作の関数
  - 関数と変数
- 関数　（２）
  - 変数の種類と記憶域クラス
  - 自動変数
  - 外部変数
  - 静的変数
- 変数のネーミング
  - 教科書 p.95～p.98, p.101～p.107

変数の寿命の違い

- 変数の
  初期化のタイミング・有効範囲が決まる
  - 自動変数
  - 外部変数
  - 静的変数
  - レジスタ変数 ← 現在では不要
- 有効範囲の違いによる変数の分類
  - 局所（ローカル）変数は,
    その関数内でのみ使用可能
  - 大域（グローバル）変数は,
    そのプログラム全体で使用可能

自動変数 (1)

- 自動変数： 通常の局所変数と同じ
  - 関数の内部で宣言される
  - その関数内部でのみ使用可能
    - 他の関数で同一名の自動変数があった場合，別々の変数として扱われる
  - 関数を実行（呼び出し）するたびに生成される
  - 関数を終了（return）すると消滅する
- 通常は自動変数を使用する

自動変数の例

```
int main( void ) {
  int n = 10;
  printf("n (main) = %d¥n", n);
  sub1();
  printf("n (main) = %d¥n", n);
  return 0;
}
void sub1(void) {
  int n = 20;
  printf("n (sub1) = %d¥n", n);
}
```

関数main内部での自動変数
mainの中でのみ使用可能

main内部のnと
sub1内部のnは
異なる変数

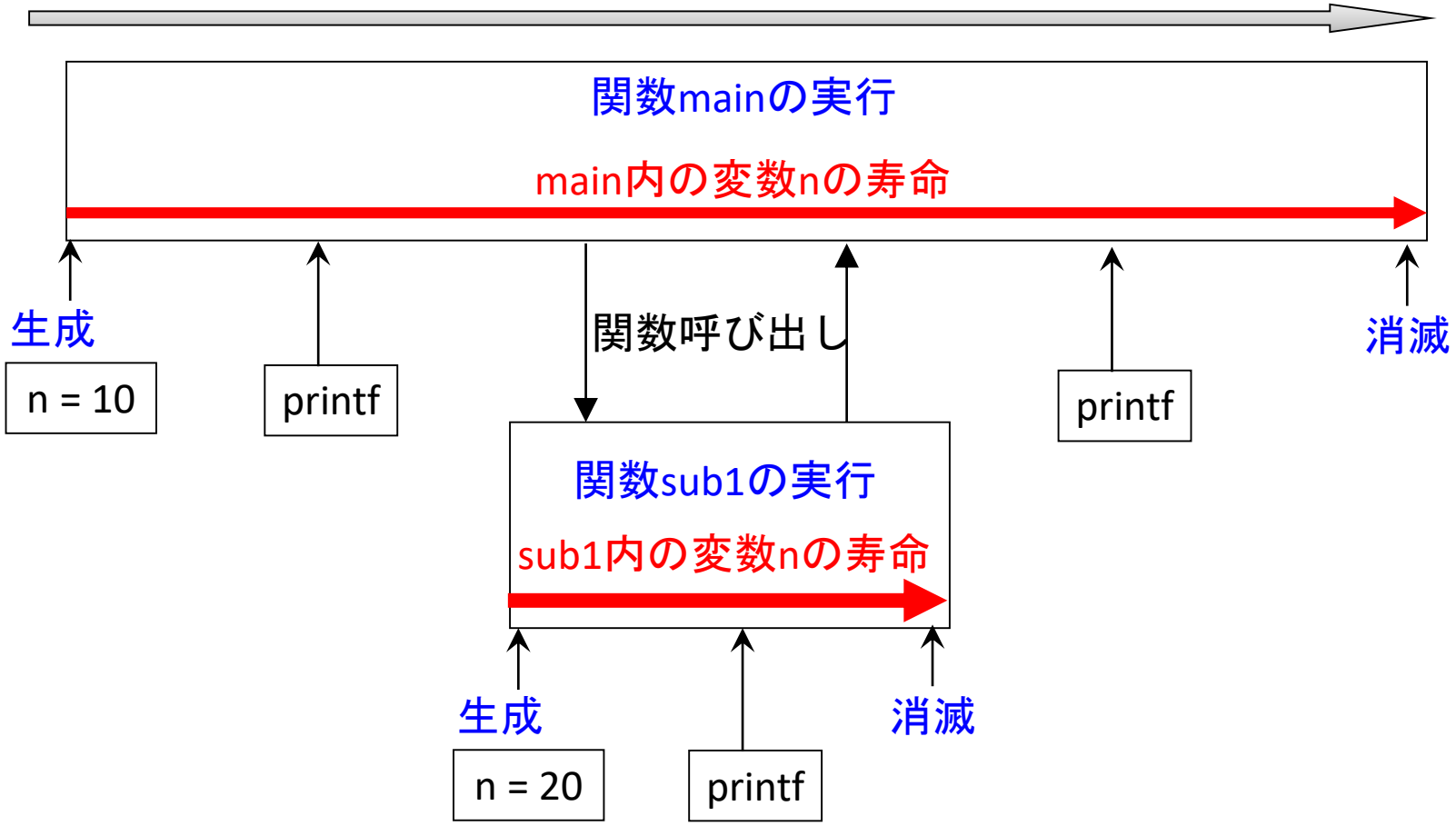
関数sub1内部での自動変数
sub1の中でのみ使用可能

実行結果：

```
n (main) = 10
n (sub1) = 20
n (main) = 10
```

自動変数の「寿命」

前ページのプログラム



自動変数 (2)

- 補足： ブロック内の自動変数
  - ブロック内で宣言する自動変数は，そのブロック内のみの寿命を持つ

```
int main(void) {
  int x = 100;
  int i;

  printf("[A] x = %d¥n", x);
  for(i = 0; i < 5; i++) {
    int x = i * 10;
    printf("[B] x = %d¥n", x);
  }
  printf("[C] x = %d¥n", x);
  return 0;
}
```

関数内の自動変数

異なる変数

ブロック内の自動変数

forのブロック内でのみ有効
（外側のxは一時的に見えなくなる）

自動変数 (3)

```
int main(void) {
  int x = 100; ←―― 外側のx
  int i;

  printf("[A] x = %d¥n", x);
  for(i = 0; i < 5; i++) {
    int x = i * 10;
    printf("[B] x = %d¥n", x);
  }
  printf("[C] x = %d¥n", x);
  return 0;
}
```

実行結果

```
[A] x = 100
[B] x = 0
[B] x = 10
[B] x = 20
[B] x = 30
[B] x = 40
[C] x = 100
```

[C]で表示されるのは「外側のx」

プログラミングの基礎　第5回

- 復習：関数（１）
  - 自作の関数
  - 関数と変数
- 関数　（２）
  - 変数の種類と記憶域クラス
  - 自動変数
  - 外部変数
  - 静的変数
- 変数のネーミング
  - 教科書 p.95～p.98, p.101～p.107

外部変数

- 外部変数：
  グローバル変数（大域変数）と同じ
  - 関数の外で宣言される変数
    - プログラム実行開始時に生成される
    - プログラム終了時に消滅する
  - すべての関数から使用可能
    - 外部変数と局所変数の名前が同じ場合，
      局所変数が優先される
- どうしても必要な場合以外は使用禁止
  - 外部変数を多用すると，分かりにくくなるため

外部変数の例

```
int n;
int main(void) {
    n = 10;
    printf("n (main) = %d¥n", n);
    sub1();
    printf("n (main) = %d¥n", n);
    return 0;
}
void sub1(void) {
    n = 20;
    printf("n (sub1) = %d¥n", n);
}
```

外部変数は，関数の外で宣言する

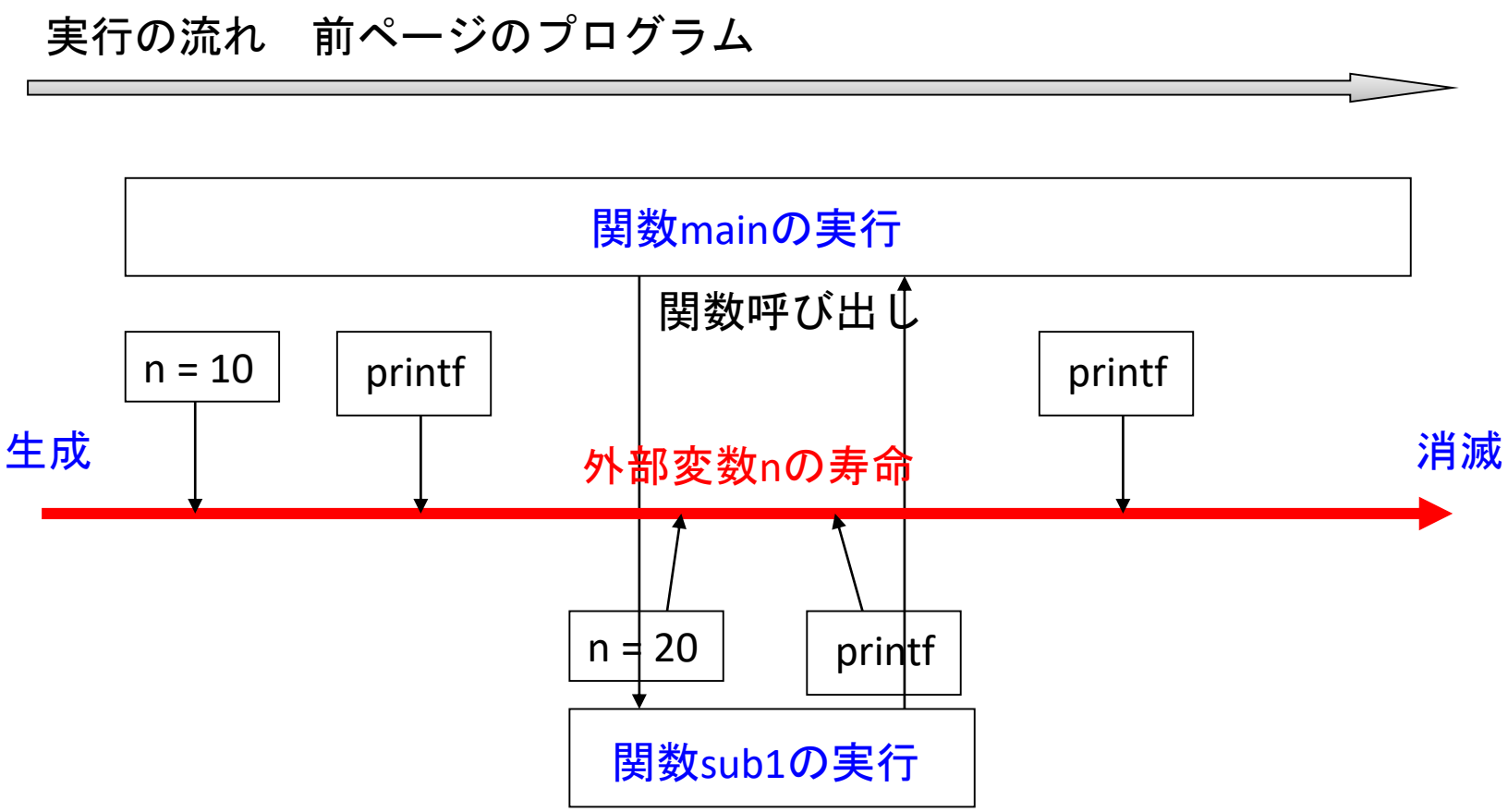
外部変数は，どの関数からも使える

実行結果

```
n (main) = 10
n (sub1) = 20
n (main) = 20
```

main内部で使用しているnと，
sub1内部で使用しているnは
同一の変数

外部変数の「寿命」


実行の流れ　前ページのプログラム
関数mainの実行
関数呼び出し
n = 10
printf
printf
生成
外部変数nの寿命
消滅
n = 20
printf
関数sub1の実行

プログラミングの基礎　第5回

- 復習：関数（１）
  - 自作の関数
  - 関数と変数
- 関数　（２）
  - 変数の種類と記憶域クラス
  - 自動変数
  - 外部変数
  - 静的変数
- 変数のネーミング
  - 教科書 p.95～p.98, p.101～p.107

静的変数

- 静的変数： 特殊な局所変数
  - 宣言時に static を付ける
  - 関数内部のみで使用可能
    - プログラム実行開始時に生成（初期化）される
    - プログラム終了時に消滅する
- 自動変数と外部変数の特徴の組み合わせ
  - 有効範囲は自動変数と同じ
  - 生成・消滅は外部変数と同じ
- 局所変数の値を
  保存しておきたいときに使用

静的変数の例

```
int main(void) {
   int i;
   for(i = 1; i <= 5; i++)
     printf("sum = %d¥n", sum( i ));
   return 0;
}
```

```
int sum( int x ) {
   int s = 0;
   s += x;
   return s;
}
```

例題8.3

```
int sum( int x ) {
   static int s = 0;
   s += x;
   return s;
}
```

例題8.4

実行結果　（例題8.3）

```
sum = 1
sum = 2
sum = 3
sum = 4
sum = 5
```

実行結果　（例題8.4）

```
sum = 1
sum = 3
sum = 6
sum = 10
sum = 15
```

違いが生じるのはなぜか？

自動変数： 例題8.3の場合

例題8.3

```
int main(void) {
   int i;
   for(i = 1; i <= 5; i++)
     printf("sum = %d¥n", sum(i));
   return 0;
}

int sum(int x) {
   int s = 0;
   s += x;
   return s;
}
```

sum関数を呼び出す毎に
s = 0を実行する

```
sum = 1
sum = 2
sum = 3
sum = 4
sum = 5
```

main関数：
i = 1
sum(1)を呼び出す

sum関数：
s = 0
s += 1
return 1

main関数：
i = 2
sum(2)を呼び出す

sum関数：
s = 0
s += 2
return 2

以下繰り返し

静的変数： 例題8.4の場合

例題8.4

```
int main(void) {
   int i;
   for(i = 1; i <= 5; i++)
     printf("sum = %d¥n", sum(i));
   return 0;
}

int sum(int x) {
   static int s = 0;
   s += x;
   return s;
}
```

s = 0は
プログラム開始時に
一回だけ実行される

```
sum = 1
sum = 3
sum = 6
sum = 10
sum = 15
```

main関数：
i = 1
sum(1)を呼び出す

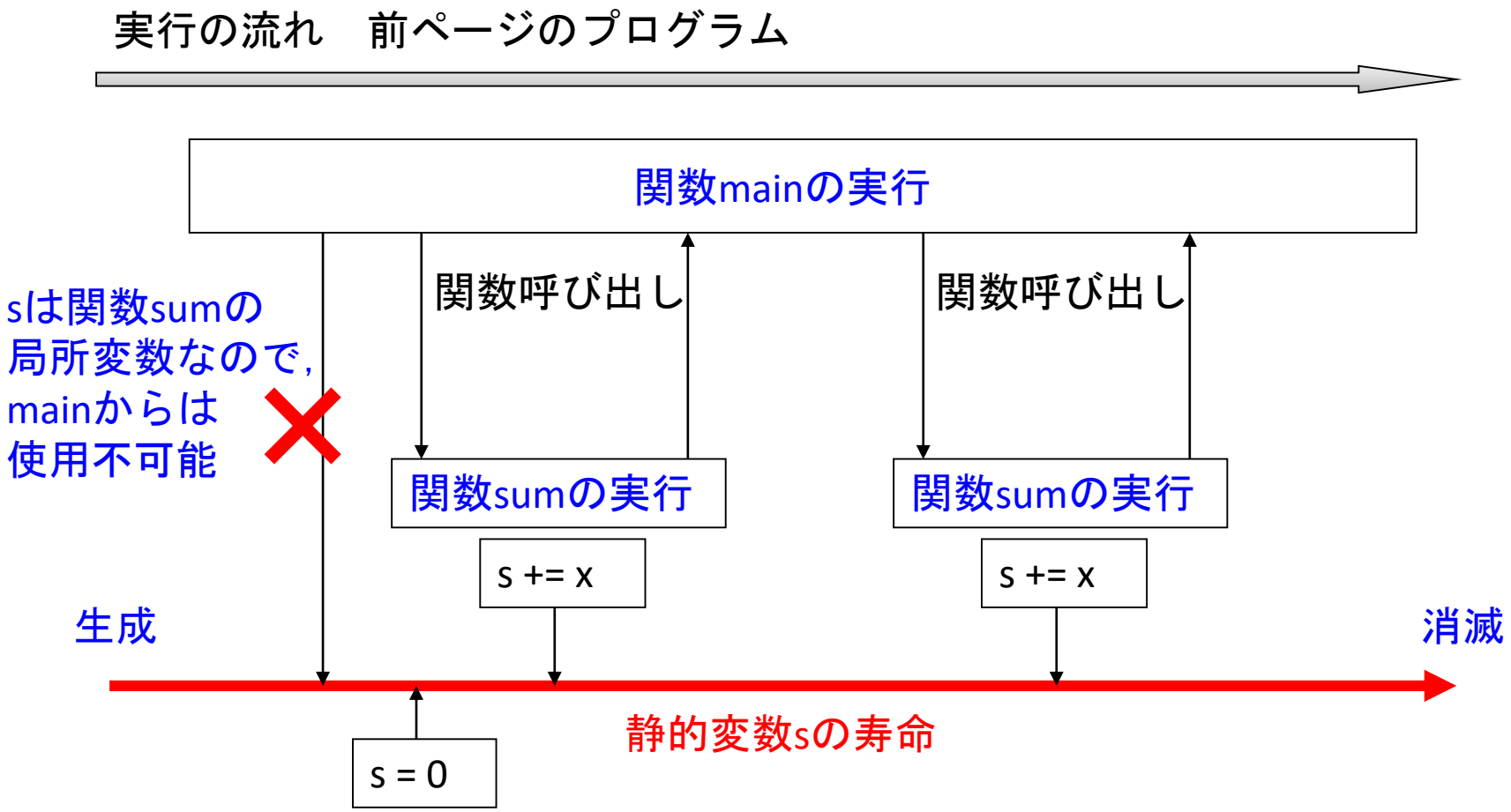
sum関数：
s = 0
s += 1
return 1

main関数：
i = 2
sum(2)を呼び出す

sum関数：
s += 2
return 3

以下繰り返し

静的変数の「寿命」


実行の流れ　前ページのプログラム
関数mainの実行
関数呼び出し
関数呼び出し
sは関数sumの
局所変数なので，
mainからは
使用不可能
関数sumの実行
関数sumの実行
s += x
s += x
生成
消滅
静的変数sの寿命
s = 0

記憶域クラスのまとめ

	自動	外部	静的
有効範囲	関数内部	ファイル 全体	関数内部
宣言場所	関数内	関数外	関数内
初期化 （生成）	関数 呼び出し時	プログラム 開始時	プログラム 開始時
消滅	関数 終了時	プログラム 終了時	プログラム 終了時

プログラミングの基礎　第5回

- 復習：関数（１）
  - 自作の関数
  - 関数と変数
- 関数　（２）
  - 変数の種類と記憶域クラス
  - 自動変数
  - 外部変数
  - 静的変数
- 変数のネーミング
  - 教科書 p.95～p.98, p.101～p.107

なぜ，変数の名前が重要なのか？

- 「名前を見ただけ」で
  意味が分からないと困るから
  - 例1：プログラムを作って１年後に見直すとき
    - 「この変数，何に使うんだったかな？」となる
  - 例2：チームでプログラムを作るとき
    - 他の人が意味不明な名前を付けると困る

変数の名前 (1)

- ループの制御（forやwhileの繰り返し回数）
  - 良い名前：
    - i
    - 繰り返し回数には i を使うのが世界標準
    - ループが2回出てきたら，j を使う
  - 悪い名前：
    - 上記以外
    - a, b, c, d, e, f, ...
    - I, J, Kなどの大文字もダメ

変数の名前 (2)

- データの個数
  - 良い名前
    - number, num（numberの省略形.　「数」という意味）
    - num_data　（number of dataの略）
    - size, datasize（データ量という意味）
  - 悪い名前
    - n　（省略しすぎ）
    - a, b, c, ...　（「データ数」という意味につながらない）
    - x　（xは「変化するもの」に付ける名前）
    - data　（dataは「データそのもの」に付ける名前）
    - NUM　（大文字は使わない）

変数の名前 (3)

- 身長
  - 良い名前
    - height （身長という意味が明確）
  - 悪い名前
    - x, y, z （何を表しているか意味不明）
    - a, b, c （同上）
    - s, t （省略しすぎ）
    - cm （長さの単位であって，身長を表す訳ではない）
    - l （小文字のエル１文字だけの変数名は絶対に禁止）
    - sin, tai（上の名前よりはマシだが，良い名前ではない）

変数の名前 (4)

- 身長の合計
  - 良い名前
    - sum_height
    - sumHeight　（区切りに大文字を使うのは可）
  - 良いけれど，多少問題のある名前
    - sumheight　（区切りがないと読みにくい）
    - sum_shincho　（英語と日本語が混ざっている）
  - 悪い名前
    - sumshin, shingo（省略しすぎ）
    - sum1, sumX　（身長が１で体重が２なんて誰が決めた？）
    - a,b,c,...　（何を表しているか意味不明）

【演習課題】

- 演習課題提出システムの
  「第5回」演習課題を実施
- 演習課題5-4～5- 6
  - 演習時間に，設計後に演習環境に
    リモートログインしプログラムを作成
  - 演習環境でコンパイル，テストし，完成
  - ソースファイル（5-x.c）をWinSCP等でPCへ転送
  - 課題提出システムへSubmitし各課題100点を目指す
  - 時間内に終了しない場合　⇒　宿題
- 授業翌週水曜日の午後5時までに必ず提出

【レポート】

- 対象：演習課題5-1,5-2, 5-3
  - Google Classroomの【資料】にアップした課題ファイル「exercise_2021_05(function2)」に回答を記述せよ.
    ※5-2,5-3では理由を記述する必要がある.
    誤字脱字がないことを確認し，正しい用語を用いて，具体的に分かり易く説明すること.
- 対象：演習課題5-4, 5-5
  - フローチャート（astah*にて作成し「ツール」，「画像出力」にてPNG形式ファイルを作成しポータルの授業資料にアップした”レポートファイル”に貼り付け）
  - ソースコード，コンパイル結果，実行結果をscriptコマンドでファイルにし印刷し”レポートファイル”に貼り付け.
- 授業翌週水曜日の午後5時までに
  55号館304室前のレポート提出小箱へ提出