プログラミング基礎

第6回

ポインタ

プログラム作成の場所

- 自分のホームディレクトリの中に「pp6」というフォルダを作成する
  - `mkdir ~/pp6`
- 現在のディレクトリを確認する場合、以下のコマンドをターミナルで入力
  - `pwd`
- pp6 のフォルダに移動
  - `cd ~/pp6`

変数とメモリ上アドレス

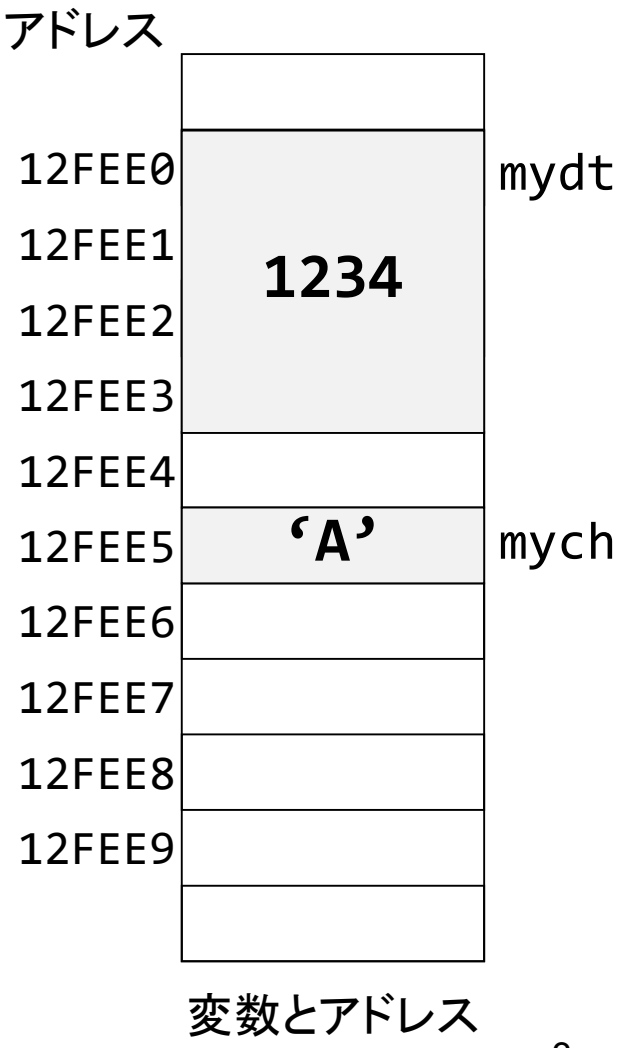
- データはメモリ上に置いて処理する
  - int mydt = 1234;
  - char mych = ‘A’;

は

  - 12FEE0 番地を mydt と呼んで整数 1234 を入れる
  - 12FEE5 番地を mych と呼んで文字 ‘A’ を入れる

と解釈する

int 型変数は4バイト，char 型変数は1バイト



変数のアドレス表示

- 「&mydt」の形で変数の格納されるアドレスを表示できる
  - アドレス表示の時に用いる変換指定は **“%p”**

```
#include<stdio.h>
int main(void){
    int mydt = 1234;
    printf(“mydt=%d¥n”, mydt);
    printf(“mydt’s address=%p¥n”, &mydt);
    return 0;
}
```

アドレスの格納

- 「&mydt」の値そのもの（アドレス）を格納したい
  - 文字型？整数型？実数型？
  - どれにも格納できない

⇒**ポインタ：アドレスを格納できる変数**

- ポインタ変数名の前に「*」を付けてポインタ宣言

```
int *pt;
pt = &mydt;
```

ポインタ pt を宣言

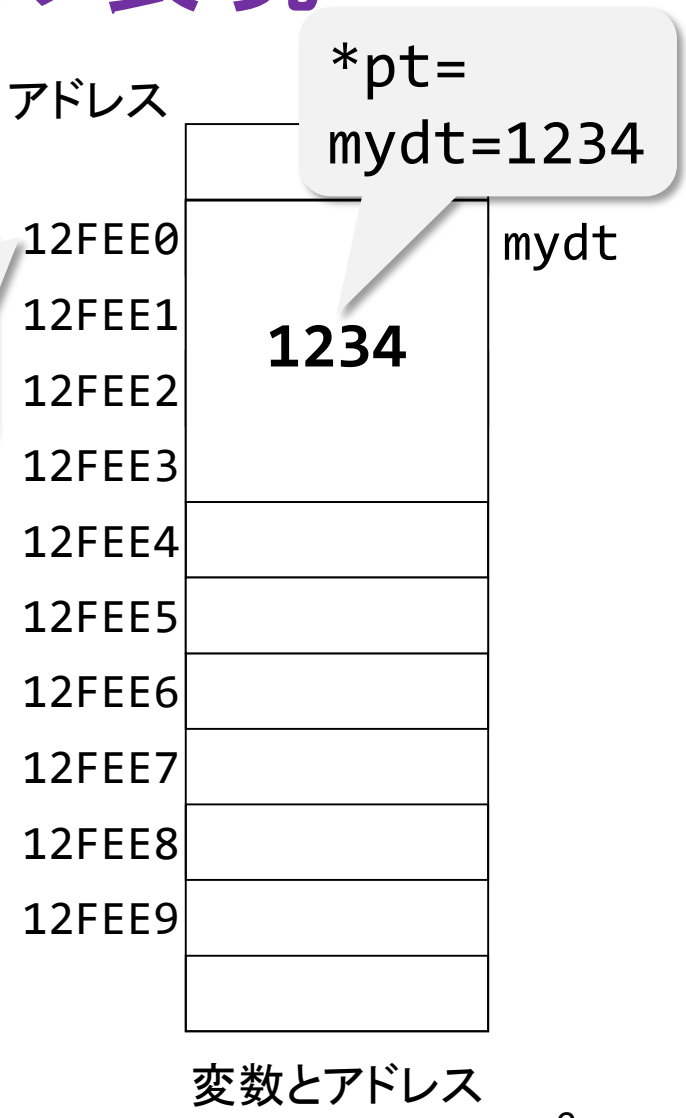
ポインタ pt に
mydt のアドレス
を格納

ポインタによる値の表現

- ポインタ pt が指しているアドレスに格納されているデータを表示したい

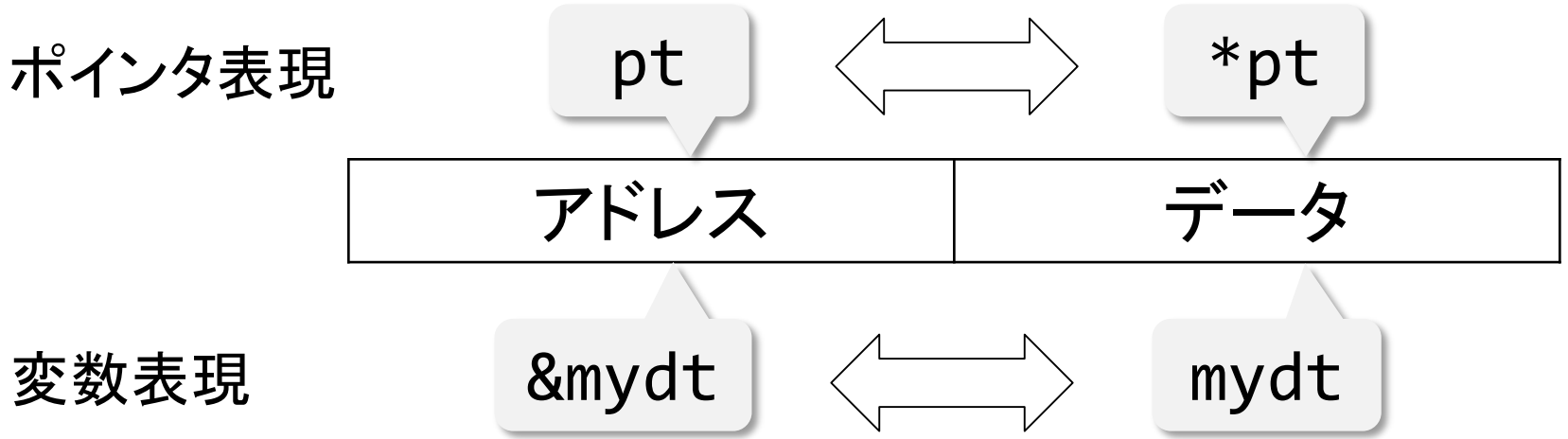
⇒*pt で表わす

「pt = &mydt」のとき,
「*pt」には
「mydt」が指示するアドレスに格納されている値
が格納されている



ポインタと変数の対応

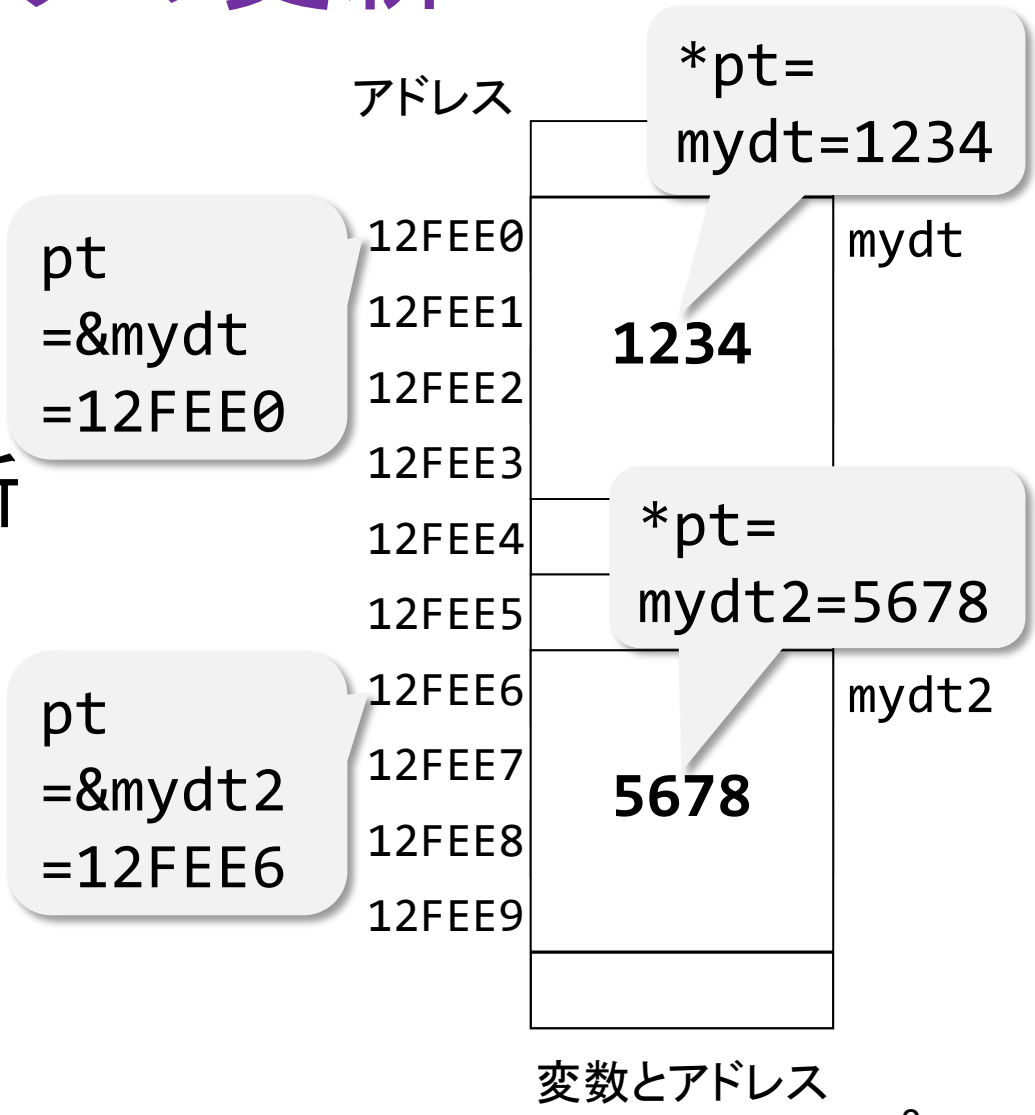
- pt = &mydt であるとき



ポインタの更新

- ポインタ変数に新しいアドレスを代入すると，ポインタ変数が更新される

```
int mydt = 1234;
int mydt2 = 5678;
int *pt;
pt = &mydt;
pt = &mydt2;
```



ポインタの初期化

- ポインタ宣言時にアドレスを初期設定することも可能
  - 通常の変数の初期化と同様

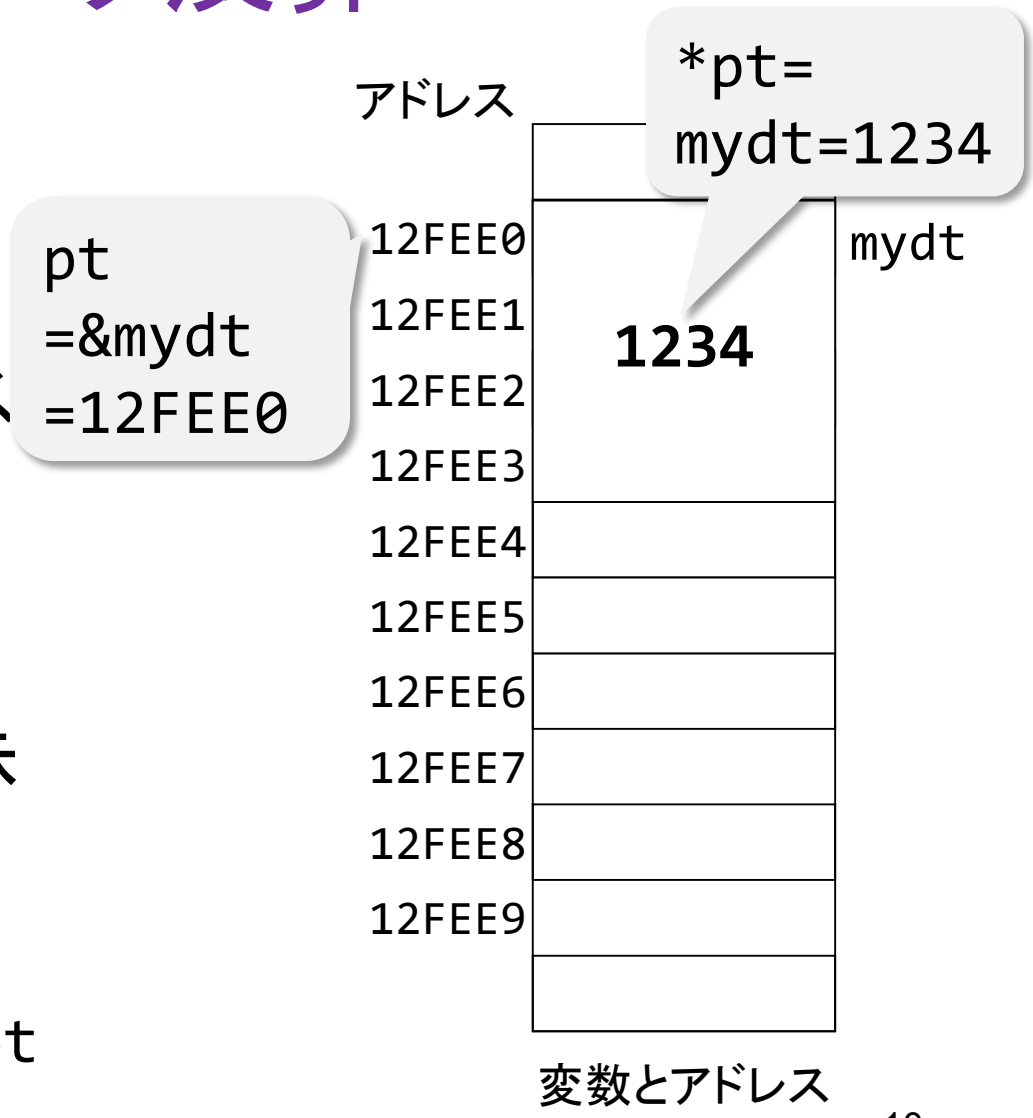
```
int nn;
int *pt = &nn;
```

変数 nn を宣言

ポインタ変数 pt に
変数 nn のアドレスを格納して
初期設定

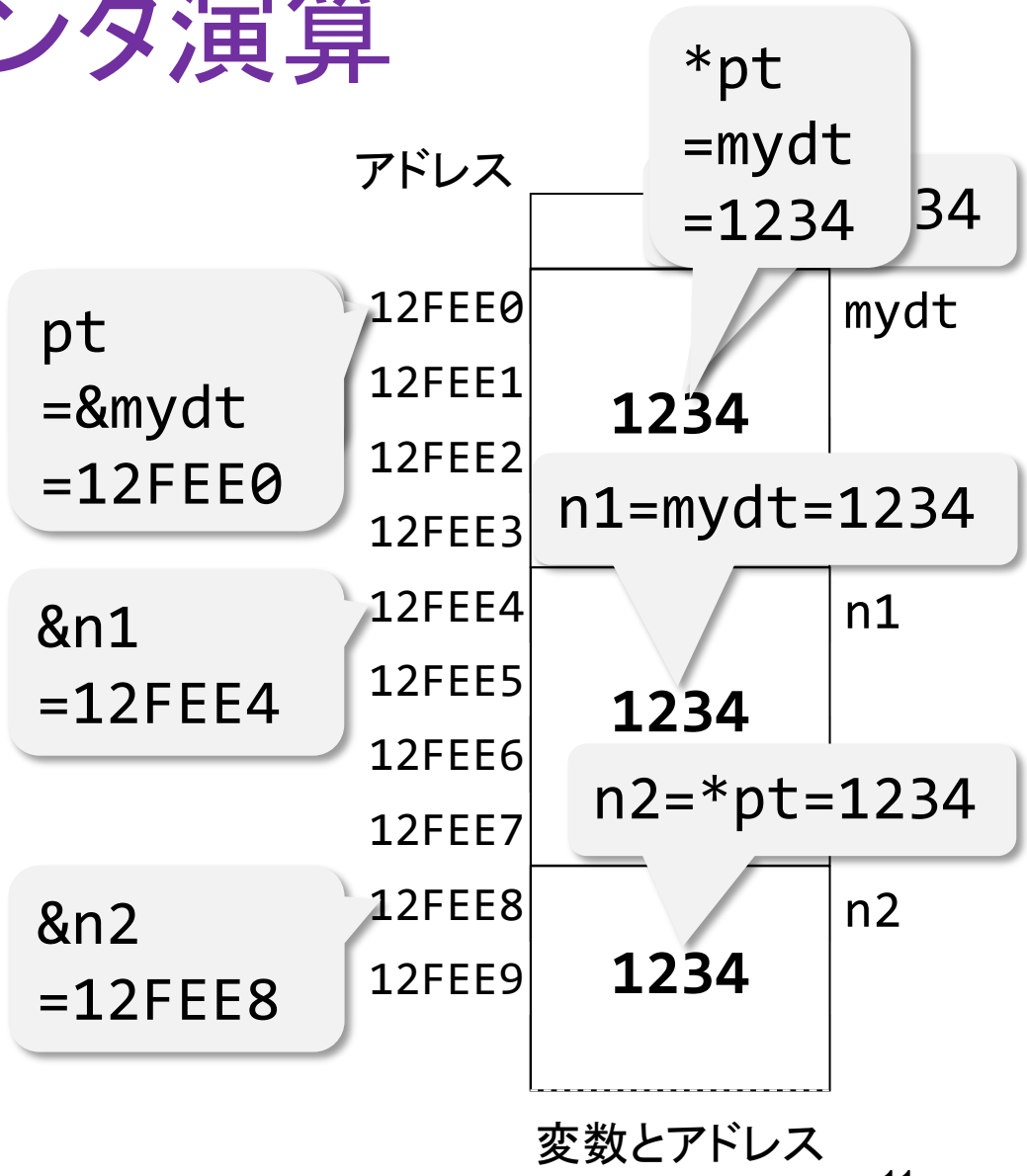
ポインタ演算

- ポインタを操作するための演算子
  - 「&」: 変数のアドレスを取り出す
    - mydt のアドレスは &mydt
  - 「*」: ポインタの指示するアドレスにある値を参照
    - pt の指す値は *pt



ポインタ演算

```
mydt = 1234;
n1 = mydt;
pt = &mydt;
n2 = *pt;
```



ポインタの型指定

- ポインタの型は操作したいデータの型と同一
  - `int` 型のデータを扱うポインタなら `int` 型
  - `double` 型のデータを扱うポインタなら `double` 型
  - `char` 型のデータを扱うポインタなら `char` 型

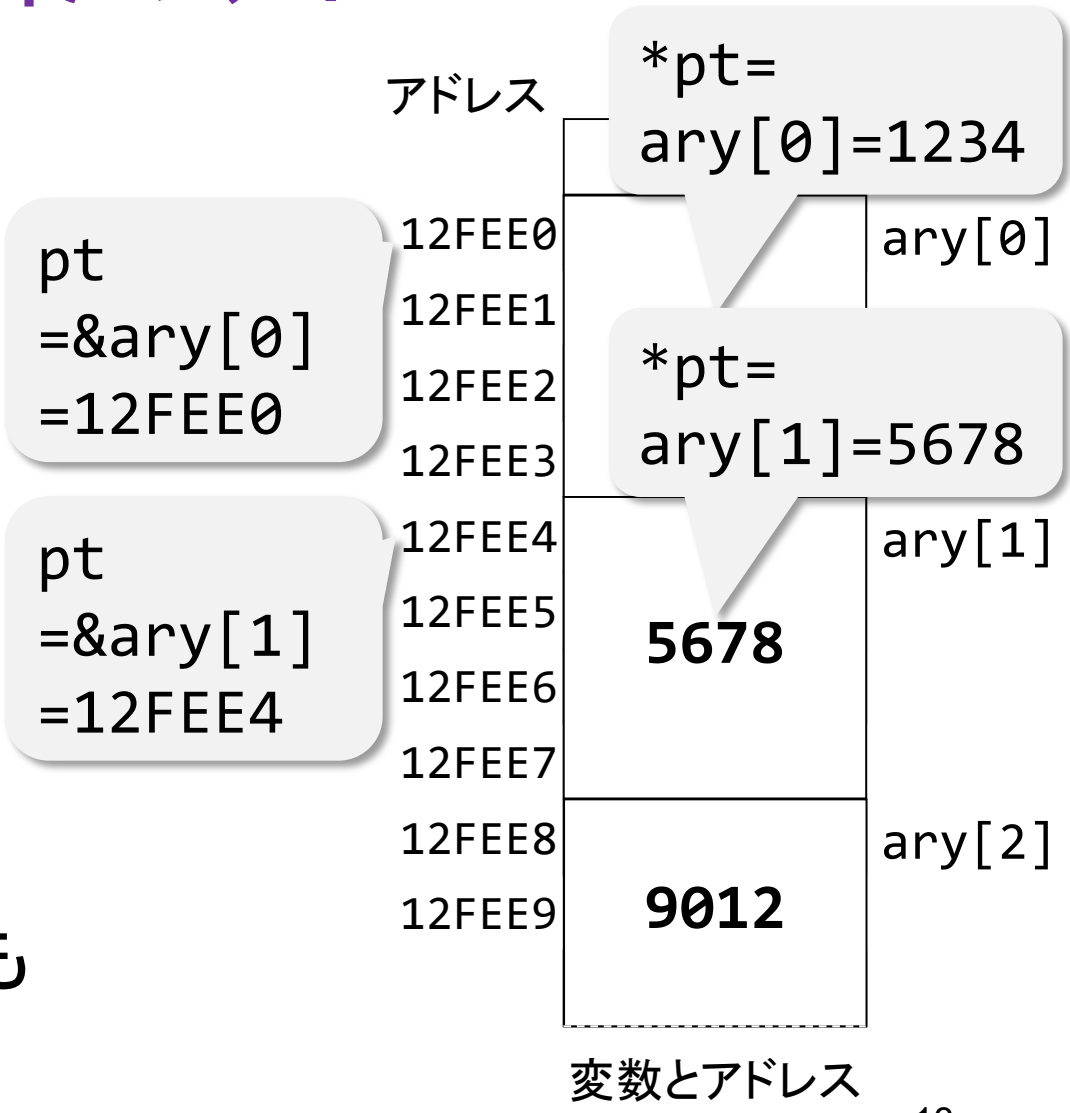
- ポインタの型が異なるとデータも操作できない
- ポインタの初期設定をせずにポインタ(*pの形)を使おうとするとコンパイル時に警告が出る場合がある

配列要素のアドレス

- 配列の各要素をポインタ変数に格納することもできる

```
int ary[3];
pt = &ary[0];
pt = &ary[1];
```

- 文字列のところでも触れた



配列名

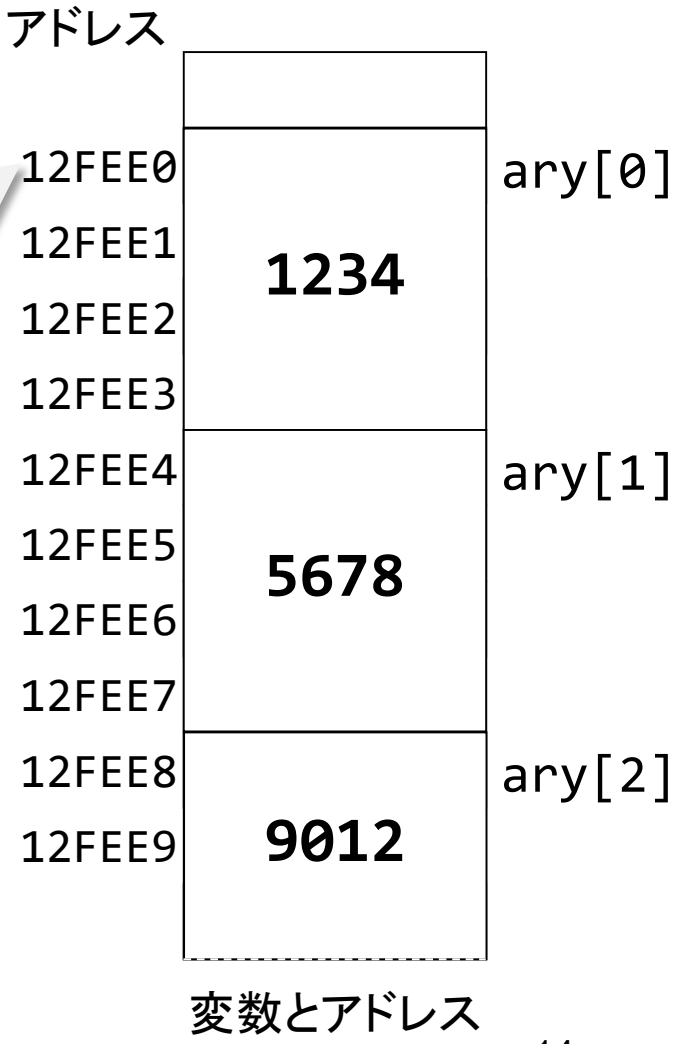
- 配列の名前は実は**配列全体の先頭アドレス**を指している

```
pt = ary;
```

という書き方も可能

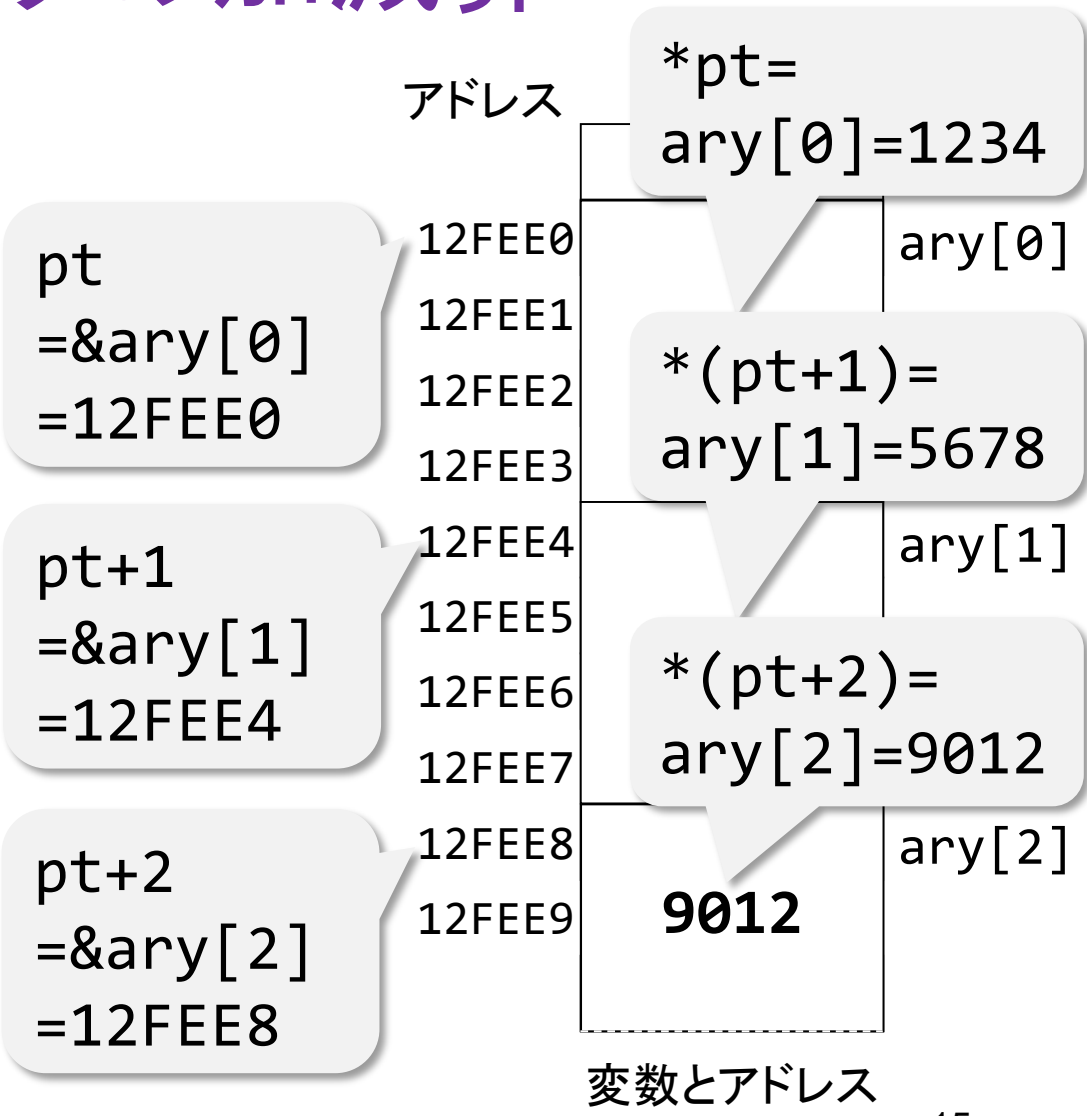
×`pt = &ary;`

×`ary = mych;`



ポインタの加減算

- ポインタの加減算も可能
  - 「+」,「-」,「++」,「--」が利用可能



ポインタと文字列

- 文字列：char 型の配列として扱える
  - 終端が「‘¥0’」

```
char s[10] = “ABCD”;
char *p;
p = s;
```

とすれば，

```
puts(s);
puts(p);
```

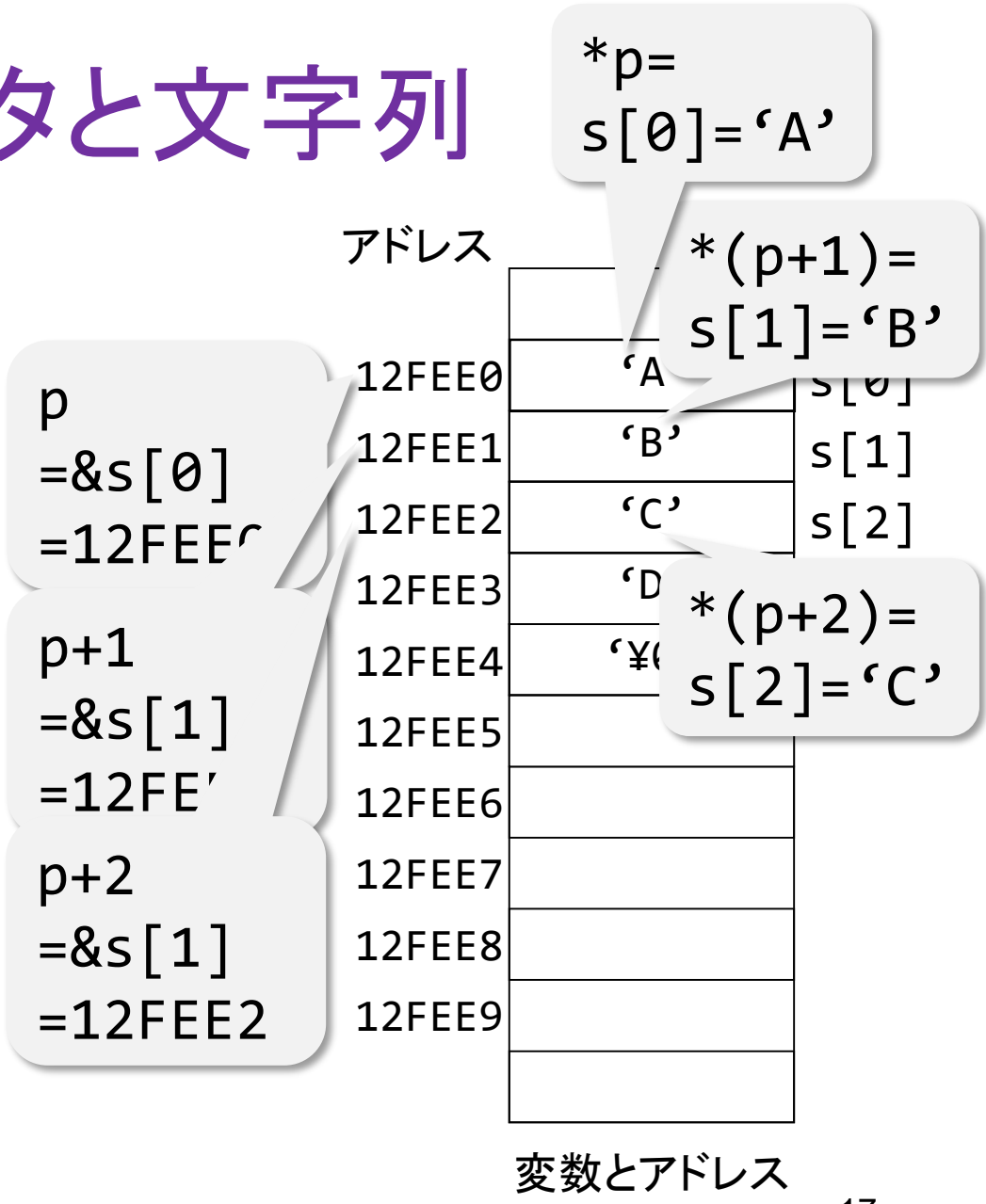
は同じ “ABCD” を出力

配列名 s：s[] の先頭アドレスを持つ
s 自身の値は変更不可
ポインタ p：s[] の先頭アドレスを持つ
p 自身の値は変更可能

ポインタと文字列

- 文字列も配列と同様に操作できる

```
putchar(*p)
→‘A’ を出力
putchar(*(p+1))
→‘B’ を出力
puts(p)
→“ABCD” を出力
puts(p+1)
→“BCD” を出力
```



まとめ

```
char *p;
char s[] = “ABCD”;
p = s;
```

制御変数を使わずに
p だけで文字列を表現できる

データ	配列名	p による表現
‘A’	s[0]	*p
‘B’	s[1]	*(p+1)
‘C’	s[2]	*(p+2)
‘D’	s[3]	*(p+3)
‘¥0’	s[4]	*(p+4)

関数にポインタを渡す

```
#include<stdio.h>
void func(int *dt){
   printf(“%d¥n”, *dt);
}
int main(void){
   int nn = 1234;
   func(&nn);
   return 0;
}
```

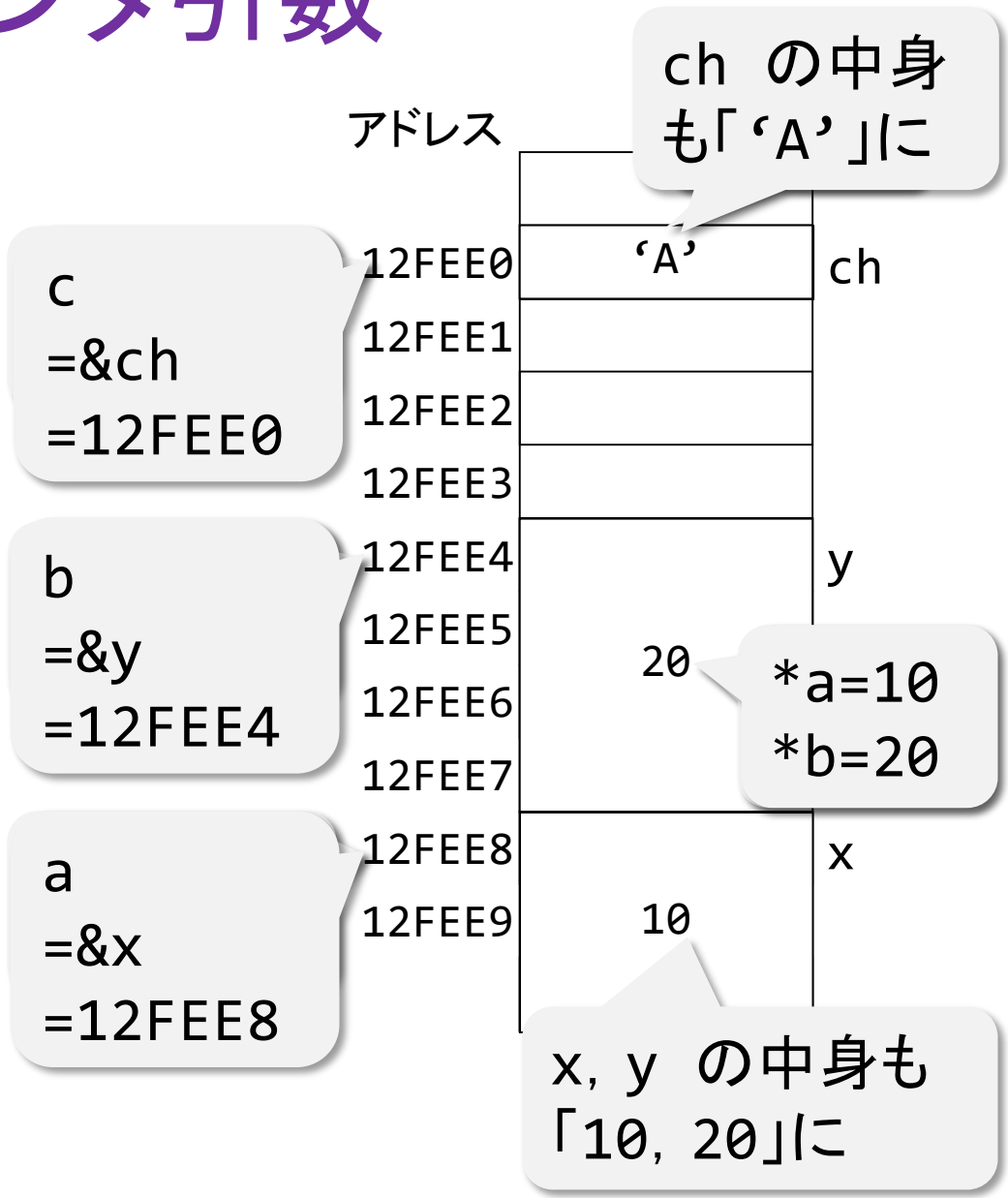
ポインタ dt の値を表示

実引数は「&nn」の形で
変数 nn のアドレスを渡す

ポインタ引数

- ポインタ引数はメモリ空間上の場所を渡しているだけ

⇒関数内で値を変更すれば呼び出し元での変数も変更される



scanf

- scanf は入力されたデータを指定のアドレスに格納する
  - 文字，整数，実数変数は変数名の前に「&」を付けてアドレス表現にする

    scanf(“%d”, &n);
  - 配列，文字列変数は変数名が変数の先頭アドレスを指しているので，「&」がいらない

    scanf(“%s”, s);

ポインタによる配列渡し

```
#include<stdio.h>
void func1(int d[], int len){
  int i;
  for(i=0;i<len;i++)
    printf(“%d ”,d[i]);
  printf(“¥n”);
}
void func2(int *d, int len){
  int i;
  for(i=0;i<len;i++)
    printf(“%d ”,*(d+i));
  printf(“¥n”);
}
int main(void){
  int nn[4]={ 10, 20, 30, 40 };
  func1(nn,4);
  func2(nn,4);
  return 0;
}
```

```
% ./prg
10 20 30 40
10 20 30 40
```

配列の形の仮引数

ポインタの形の仮引数

表現は違うが同じ処理になる

ポインタで文字列を渡す

配列の形の仮引数

```
#include<stdio.h>
void strout(char s[]){
  int i;
  printf(“%s¥n”,s);
  for(i=0;s[i]!=‘¥0’;i++)
    printf(“%X ”,s[i]);
  printf(“¥n”);
}
void strout_p(char *p){
  printf(“%s¥n”,p);
  while(*p){
    printf(“%X ”,*p);
    p++;
  }
  printf(“¥n”);
}
int main(void){
  char st[]=“ABCD”;
  strout(st);
  strout_p(st);
  return 0;
}
```

```
% ./prg
ABCD
41 42 43 44
ABCD
41 42 43 44
```

ポインタの形の仮引数

p の指すアドレスに文字が格納されている限りループ

表現は違うが同じ処理になる

ポインタを戻り値に

- 関数の戻り値としてポインタそのものを指定することも可能

```
char *bigstr(char *s1, *s2){
    if(strcmp(s1, s2) >0)
        return s1;
    else
        return s2;
}
```

ポインタとして戻す場合は関数名の前に「*」
呼び出し元では戻り値をポインタとして扱える

課題の進め方

- 課題のページの[課題6-1]を使って説明します。

  - プログラミング基礎2018のページの「課題に関する注意と提出方法」を読んでおくこと
    - コピーはしない
    - 作成したプログラムは時間内に所定のディレクトリにコピーする
    - コメント（プログラム名、学籍番号、氏名）の記入

提出する前に

- 課題を提出する前に、自分のプログラムをチェックしよう！！

- 以下のツールを用いる

  ppchkall

  使用例:

  ```
  ppchkall ファイル名（実行形式）
  ```

もしできていれば，正しい旨の表示がされます

課題の提出先

- 今回の提出先
  - /FUN/Work/pp2018/06/(クラス)/(学籍番号)
- 提出する際の注意点
  - 空白文字の位置や数を含めて同様の入出力結果が得られること
- 提出するもの
  - ソースファイル(テキスト形式)
  - 実行形式(コンパイルの結果)

ファイル名は課題のページに記載されているものを用いること！