

情報処理演習 (10) ポインタ

知能システム学 准教授
万 偉偉(ワン ウェイウェイ)

ポインタとは？

- 計算機のメモリは小さい単元で構成される。
- 各単元はアドレス値があり、プログラムはアドレス値をもって当該の単元をアクセスする。
- アドレス値もデータ型として処理する。有効のアドレス：データ型の範囲。

ポインタとは？

- 変数を宣言すると、当該変数のデータ型のサイズ分のメモリの単元が割り当てられる。変数の位置は割られた単元のアドレスとなる。この位置は以下の特徴を持つべきである。
 - データ型を持ち、範囲は有効のアドレスである。
 - 割当てた単元のサイズを表す必要がある。
- 故に、C言語は新たなデータ型、つまりポインタ型を定義する。

ポインタの宣言

- ポインタ型の変数を宣言するのは、変数名の先頭にアスタリスク*を付ける。

```
type *pointername;
```

- ここで、pointernameはポインタ型のデータを格納する。typeをint、doubleなどC言語のデータ型で書き換え、pointernameに割当てた単元のサイズを表す。

```
例 int *pa;
```

- paはポインタ型の変数であり、4バイト分のメモリの一番目のバイトのアドレスを指す。

ポインタの宣言

- 変数のアドレスを取得する場合、変数の先頭にアンパサンドを付ける。上記のpaに値を与えられるのは以下の文で行う。

```
int a = 1;  
pa = &a;
```

- &aでaのアドレスを取得し、ポインタ型の変数paに与えられる。

ポインタの宣言

- 注意：
 - aは整数型なので、取得したアドレスは整数型のデータを指すことを表す必要がある。paもint *で定義しないとイケない。

```
char *pca;  
int a = 1;  
pca = &a;
```

の場合、pcaはpaと同じくaの一番目のバイトのアドレスを指す。ただし、単元のサイズはchar型の変数が占めるメモリのサイズとなる。

ポインタの宣言

- ポインタ型のサイズ
 - 宣言に使われたtypeによらず4バイトであり、メモリのアドレスの数値を格納する。
- ポインタ型の変数が指す領域のサイズ
 - 宣言に使われたtypeに決められる。

ポインタの演算

- ポインタの演算については以下の五つがある。
 - 代入・参照・前進・後進・等価
- 代入
 - アドレス値をポインタ型の変数に与えること。前節の例のpa = &a;とpca=&a;は代入演算を示した。
- 参照
 - ポインタが指しているメモリの領域の中身を獲得する操作である。

```
int *pa;  
int a = 1;  
pa = &a;  
printf("%d", *pa);
```

*pcaはpcが指しているメモリの領域、つまりaに割り当てられた領域の値を取り出す。printf関数は該当値を出力する。

ポインタの演算

- 質問 右側はどうでしょう？

```
int *pa;
int a = 1;
pa = &a;
printf( "%d" , *pc);
```

```
char *pa;
int a = 1;
pa = &a;
printf( "%c" , *pc);
```

ポインタの演算

- 前進・後進
 - ポインタ型の変数に整数を足すと、指すデータの型から決定されるデータサイズ分の整数倍増加・減少する。例えばpa=pa+1はpaが指すアドレスを4バイト*1をずれて、4バイト後のアドレスに指す。pca=pca+2はpcaが指すアドレスを1バイト*2をずれて、2バイト後のアドレスに指す。

```
int *pa;
pa=pa+2と*pa=*pa+2の区別は？
```

ポインタの演算

- 等価
 - 同一のものを参照しているかどうかpa==pca?
 - ポインタ型と計算機のアドレスとは大きい異なりを持ち、強く型に縛られている。
 - NULLポインタ
 - 何も指していないポインタはNULLで定義される。
- ```
例
int *pint = NULL;
```
- ポインタはNULLかどうかについてはif (pint)で判断する。if (pint)が真の場合pintはNULLではない。if (!pint)が真の場合pintはNULLである。

# C言語の配列の謎

- C言語の配列名は、その一連の領域の先頭アドレスを指したポインタ、0番を基点として相対アドレスとしても考えられる。
- ```
char carray[10];
char *pcarray;
pcarray = carray;
```

C言語の配列の謎

- 配列名と整数型との加減算を行うと、ポインタの前進/後進する演算となる。
- ```
pcarray=carray+2;
```
- ```
pcarray=carray+2
```
- pcarrayを参照するとcarray[2]と同じ中身を表す。

C言語の配列の謎

- 「配列を関数に引き渡して関数内で値を書き換えると、なぜ関数を出てもその変更が維持されるのか？」
- データが格納される領域の中身は渡していない、共通の領域のまま、ポインタだけ渡した。関数の中で領域の中身を修正すると、関数外にも反映する。

C言語の配列の謎

- 左側の関数と右側の関数は等価
- ```
int change(int data[], int num);
int main(void) {
 int d [7] = {70, 85, ... };
 change (d, 7);
 printf("d[0] is %d\n" , d[0]);
}
int change(int data[], int num) {
 data[0] = 0;
}
```
- ```
int change(int *pdata, int num);
int change(int *pdata, int num)
{
    pdata[0] = 0;
}
```
- 一方、中身を直接に渡すのは引用渡すとなる。この場合、関数の引数に中身を渡したので、関数の中で当該引数を修正しても外部の元変数に反映しません。

二重ポインタ

- ポインタを指すポインタ型の変数は2重ポインタである。二次元配列の配列名は二重ポインタである。
- ```
int **ppa;
int *pa;
int a = 1;
pa = &a;
ppa = &pa;
```

```
int main()
{
 Int iarrarr[3][3]={{{11,12,13},{21,22,23},{31,32,33}}};
 printf("%d\n", iarrarr[2][2]);
 printf("%d\n", *((iarrarr+2)+2));
 return 0;
}
```

\*(\*(carrarr+2)+2)とcarrarr[2][2]は同じ中身を参照する。

## 二重ポインタ

- 文字列の配列を定義するのは

```
char *sarr[] = { "yamada", "yodobashi", "kizu" }
```

[]は配列の定義に使われ、後ろの中括弧と対応する。  
\*は文字に指すポインタを表し、各文字列を指す。