Day18.C

Structure (1)
Applications

Young W. Lim

December 12, 2017

0.1 Passing Structures to Functions

```
::::::::::::::
h3.c
::::::::::::::
#include <stdio.h>


//-----------------------
struct aaa {
  int a;
  int b;
} ;

struct aaa X;
//-----------------------


//-----------------------
typedef struct bbb {
  double a;
  double b;
} BType;

//-----------------------

BType Y;

void pr_aaa( struct aaa *X, char * s) {
  printf("------------------\n");
  printf("%s->a= %d \n", s, X->a);
  printf("%s->b= %d \n", s, X->b);
}

/*****
void pr_aaa( struct aaa X, char * s) {
  printf("------------------\n");
  printf("%s.a= %d \n", s, X.a);
  printf("%s.b= %d \n", s, X.b);
}
****/

void pr_bbb( struct bbb *X, char * s) {
  printf("------------------\n");
  printf("%s->a= %g \n", s, X->a);
  printf("%s->b= %g \n", s, X->b);
}

/****
void pr_bbb( struct bbb X, char * s) {
```

```
  printf("------------------\n");
  printf("%s.a= %g \n", s, X.a);
  printf("%s.b= %g \n", s, X.b);
}
****/


int main(void) {

  X.a = 100;
  X.b = 200;

  Y.a = 11.1;
  Y.b = 22.2;

  pr_aaa( &X, "X" );
  pr_bbb( &Y, "Y" );

  // pr_aaa( X, "X" );
  // pr_bbb( Y, "Y" );
}


::::::::::::::
h3.out
::::::::::::::
------------------
X->a= 100
X->b= 200
------------------
Y->a= 11.1
Y->b= 22.2
```

0.2 Passing Structures by Value and by Reference

```
::::::::::::::
h2.c
::::::::::::::
#include <stdio.h>



struct aaa {
  double x, y;
  int A[2];
};

void func1(struct aaa  X) {
  int i;
```

```
  printf("* func1 ======================\n");
  printf("&X          = %p \n", &X);
  printf("&X.x        = %p \n", &X.x);
  printf("&X.y        = %p \n", &X.y);
  printf("&X.A        = %p \n", &X.A);
  printf("&X.A[2]     = %p \n", &X.A[2]);
  printf("&i          = %p \n", &i);

  for (i=0; i<2; ++i)
    printf("X.A[%d]= %d\n", i, X.A[i]);

}

void func2(struct aaa *X) {
  int i;

  printf("* func2 ======================\n");
  printf("&X          = %p \n", &X);
  printf("X           = %p \n", X);
  printf("&X->x       = %p \n", &X->x);
  printf("&X->y       = %p \n", &X->y);
  printf("&X->A       = %p \n", &X->A);
  printf("&X->A[2]    = %p \n", &X->A[2]);
  printf("&i          = %p \n", &i);

  for (i=0; i<2; ++i)
    printf("X->A[%d]= %d\n", i, X->A[i]);

}


int main(void) {
  struct aaa X;
  int i;

  printf("sizeof(X)= %ld %lx \n", sizeof(X), sizeof(X));

  for (i=0; i<2; ++i) X.A[i] = i;

  printf("* main ======================\n");
  printf("&X          = %p \n", &X);
  printf("&X.x        = %p \n", &X.x);
  printf("&X.y        = %p \n", &X.y);
  printf("&X.A        = %p \n", &X.A);
  printf("&X.A[2]     = %p \n", &X.A[2]);

  func1(X);

  func2(&X);
```

```
}
::::::::::::::
h2.out
::::::::::::::
sizeof(X)= 24 18
* main ======================
&X          = 0x7fffe2ed8240
&X.x        = 0x7fffe2ed8240
&X.y        = 0x7fffe2ed8248
&X.A        = 0x7fffe2ed8250
&X.A[2]     = 0x7fffe2ed8258
* func1 =====================
&X          = 0x7fffe2ed8210
&X.x        = 0x7fffe2ed8210
&X.y        = 0x7fffe2ed8218
&X.A        = 0x7fffe2ed8220
&X.A[2]     = 0x7fffe2ed8228
&i          = 0x7fffe2ed81f4
X.A[0]= 0
X.A[1]= 1
* func2 =====================
&X          = 0x7fffe2ed8208
X           = 0x7fffe2ed8240
&X->x       = 0x7fffe2ed8240
&X->y       = 0x7fffe2ed8248
&X->A       = 0x7fffe2ed8250
&X->A[2]    = 0x7fffe2ed8258
&i          = 0x7fffe2ed8214
X->A[0]= 0
X->A[1]= 1
```

func1 uses passing by value

- `void func1(struct aaa  X);`
- `func1(X);`
- the parameter `X` of `func1` copies the argument `X` of the `main` function
- every member of the structure variable is copied one-to-one
- the structure address of the parameter `X` and the argument `X` are different (duplication)
- the structure's member address of the parameter `X` and the argument `X` are different

func2 uses passing by reference

- `void func2(struct aaa *X);`
- `func1(&X);`
- the parameter `X` of `func1` copies only the argument address `&X` of the `main` function
- every member of the structure variable is not copied
- the structure's member address of the parameter `*X` and the argument `X` are the same
  - `&X->x` ≡ `&(*X).x` (parameter X) ⟺ `&X.x` (argument X)
  - `&X->y` ≡ `&(*X).y` (parameter X) ⟺ `&X.y` (argument X)
  - `&X->z` ≡ `&(*X).z` (parameter X) ⟺ `&X.z` (argument X)
  - `&X->A` ≡ `&(*X).A` (parameter X) ⟺ `&X.A` (argument X)

memory locations of the array member

- `main` and `func2` shares the same memory locations for `A[0]` and `A[1]`
  - `&X.A[0]  = 0x7fffe2ed8250`
  - `&X.A[2]  = 0x7fffe2ed8258`
- `func1` has its own memory locations (duplication) for `A[0]` and `A[1]`
  - `&X.A[0]  = 0x7fffe2ed8220`
  - `&X.A[2]  = 0x7fffe2ed8228`