Day13.C

Pointers (2) Applications

Young W. Lim

December 9, 2017

0.1 const qaulifier

```
::::::::::::::
t1.c
::::::::::::::
#include <stdio.h>
#include <string.h>

int main(void) {
  int i = 100;
  const int j = 200;

  int *p = &i;
  const int *q = &i;

  printf("i= %d j= %d \n", i, j);

  i = 0;
  // j = 0;

  printf("i= %d j= %d \n", i, j);

  printf("*p= %d *q= %d \n", *p, *q);

  *p = 100;

  // *q = 200;

  printf("*p= %d *q= %d \n", *p, *q);

}

::::::::::::::
t1.out
::::::::::::::
i= 100 j= 200
i= 0 j= 200
*p= 0 *q= 0
*p= 100 *q= 100

::::::::::::::
error messages after uncommenting
::::::::::::::
young@USys01 ~ $ gcc -Wall t1.c
t1.c: In function main:
t1.c:14:5: error: assignment of read-only variable j
   j = 0;
     ^
t1.c:22:6: error: assignment of read-only location *q
   *q = 200;
```

ˆ

const int j = 200;

- denotes int j is a constant and it must not be changed.
- a read access is ok.
- no write access is allowed.
- if any write access is found in the code, an error message will be shown.
- j=0 attempts to change the initialized value 200 into the new value 0.
- this write attempt will be prohibited by issuing an error message..

const int *q = &i;

- denotes int *j is a constant and it must not be changed.
- a read access is ok.
- no write access is allowed.
- if any write access is found in the code, an error message will be shown.
- *q=0 attempts to change the initialized value 200 into the same value 200.
- this write attempt will be prohibited by issuing an error message.
- the same result when j is declared without const.

0.2 Pointer Type Cast

```
::::::::::::::
h1.c
::::::::::::::
#include <stdio.h>

int main(void) {
  int a1 = 0x10203040;
  int a2 = 0x11223344;

  long  *l;  // long  pointer l
  int   *i;  // int   pointer i
  short *s;  // short pointer s
  char  *c;  // char  pointer c

  int m;

  printf("sizeof(long   )= %ld  ", sizeof(long   ));
  printf("sizeof(long  *)= %ld\n", sizeof(long  *));
  printf("sizeof(int    )= %ld  ", sizeof(int    ));
```

```
  printf("sizeof(int   *)= %ld\n", sizeof(int   *));
  printf("sizeof(short  )= %ld  ", sizeof(short  ));
  printf("sizeof(short *)= %ld\n", sizeof(short *));
  printf("sizeof(char   )= %ld  ", sizeof(char   ));
  printf("sizeof(char  *)= %ld\n", sizeof(char  *));

  l = (long  *) &a1;
  i = (int   *) &a1;
  s = (short *) &a1;
  c = (char  *) &a1;

  printf("*l = %lx \n", *l);
  printf("*i = %x \n", *i);
  printf("*s = %x \n", *s);
  printf("*c = %x \n", *c);

  printf("&a1= %p \n", &a1);
  printf("&a2= %p \n", &a2);


  printf("---------------------------------\n");
  for (m=0; m<8; ++m)
    printf("c+%d = %p  c[%d]= %x\n", m, c+m, m, c[m]);

  printf("---------------------------------\n");
  for (m=0; m<4; ++m)
    printf("s+%d = %p  s[%d]= %x\n", m, s+m, m, s[m]);

  printf("---------------------------------\n");
  for (m=0; m<2; ++m)
    printf("i+%d = %p  i[%d]= %x\n", m, i+m, m, i[m]);

  printf("---------------------------------\n");
  for (m=0; m<1; ++m)
    printf("l+%d = %p  l[%d]= %lx\n", m, l+m, m, l[m]);



}
::::::::::::::
h1.out
::::::::::::::
sizeof(long   )= 8  sizeof(long  *)= 8
sizeof(int    )= 4  sizeof(int   *)= 8
sizeof(short  )= 2  sizeof(short *)= 8
sizeof(char   )= 1  sizeof(char  *)= 8
*l = 1122334410203040
*i = 10203040
*s = 3040
*c = 40
```

```
&a1= 0x7ffe1f5355ec
&a2= 0x7ffe1f5355f0
---------------------------------
c+0 = 0x7ffe1f5355ec  c[0]= 40
c+1 = 0x7ffe1f5355ed  c[1]= 30
c+2 = 0x7ffe1f5355ee  c[2]= 20
c+3 = 0x7ffe1f5355ef  c[3]= 10
c+4 = 0x7ffe1f5355f0  c[4]= 44
c+5 = 0x7ffe1f5355f1  c[5]= 33
c+6 = 0x7ffe1f5355f2  c[6]= 22
c+7 = 0x7ffe1f5355f3  c[7]= 11
---------------------------------
s+0 = 0x7ffe1f5355ec  s[0]= 3040
s+1 = 0x7ffe1f5355ee  s[1]= 1020
s+2 = 0x7ffe1f5355f0  s[2]= 3344
s+3 = 0x7ffe1f5355f2  s[3]= 1122
---------------------------------
i+0 = 0x7ffe1f5355ec  i[0]= 10203040
i+1 = 0x7ffe1f5355f0  i[1]= 11223344
---------------------------------
l+0 = 0x7ffe1f5355ec  l[0]= 1122334410203040
```

Point Type Cast

- `int a1 = 0x10203040;`
  - MSByte : `0x10`
  - LSByte : `0x40`
- `int a1 = 0x11223344;`
  - MSByte : `0x11`
  - LSByte : `0x44`
- pointer variables
  - `l` : pointer to long integer
  - `i` : pointer to integer
  - `s` : pointer to short integer
  - `c` : pointer to character integer
- sizes of integer types
  - `sizeof(long)` = 8 bytes
  - `sizeof(int)` = 4 bytes
  - `sizeof(short)` = 2 bytes
  - `sizeof(char)` = 1 bytes
- sizes of integer pointer types

- `sizeof(long *)` = 8 bytes
- `sizeof(int *)` = 8 bytes
- `sizeof(short *)` = 8 bytes
- `sizeof(char *)` = 8 bytes

- ```
  c+0 = 0x7ffe1f5355ec  c[0]= 40  .... LSByte of a1
  c+1 = 0x7ffe1f5355ed  c[1]= 30
  c+2 = 0x7ffe1f5355ee  c[2]= 20
  c+3 = 0x7ffe1f5355ef  c[3]= 10  .... MSByte of a1
  c+4 = 0x7ffe1f5355f0  c[4]= 44  .... LSByte of a2
  c+5 = 0x7ffe1f5355f1  c[5]= 33
  c+6 = 0x7ffe1f5355f2  c[6]= 22
  c+7 = 0x7ffe1f5355f3  c[7]= 11  .... MSByte of a2
  ```

- store the LSByte in the lower address
- store the MSByte in the higher address
- a1 (4bytes), a2 (4bytes) : total 8 bytes
  - 2 * 4-byte `int`
  - 4 * 2-byte `short`
  - 8 * 1-byte `char`
  - 1 * 8-byte `long`
- a1 is declared before a2 : a1 is stored first (lower address), then a2 is stored (higher address)
- pointer type cast
  - `(long *)` : conversion to the pointer to `long` type
  - `(short *)` : conversion to the pointer to `short` type
  - `(char *)` : conversion to the pointer to `char` type
- dereferenced variables
  - `*l` : can be considered as a `long` type variable
  - `*i` : can be considered as a `int` type variable
  - `*s` : can be considered as a `short` type variable
  - `*c` : can be considered as a `char` type variable
- array names
  - `l` : can be considered as the name of the array `long l[1];`
  - `i` : can be considered as the name of the array `long i[2];`
  - `s` : can be considered as the name of the array `long s[4];`
  - `c` : can be considered as the name of the array `long c[8];`
  ```

0.3 Single Precision Number Format

```
::::::::::::::
h1.c
::::::::::::::
#include <stdio.h>


// 23bit mantissa  : m
// 8bit exponent   : e
// 1bit sign       : s

int main(void) {
  float x = 0.15625F;
  int  *p = (int *) &x;
  int   m, e, s;
  float M, E, S;
  float X;


  m = (*p) & 0x7fffff;   // 3+4*5= 23
  e = (*p >> 23) & 0xff; // 4*2= 8
  s = (*p >> 31) & 0x1;

  printf("m= %#10x %10d\n", m, m);
  printf("e= %#10x %10d\n", e, e);
  printf("s= %#10x %10d\n", s, s);

  M = 1.0 + (float) m / (1 << 23) ;
  E = e - 127;
  S = s ? -1 : +1;

  printf("M= %10f \n", M);
  printf("E= %10f \n", E);
  printf("S= %10f \n", S);

  if (E >= 0) X = S*M*(1<<(int) E);
  else        X = S*M/(1<<(int)-E);

  printf("X= %10f \n", X);

}

::::::::::::::
h1.out
::::::::::::::
m=   0x200000    2097152
e=       0x7c        124
s=          0          0
M=   1.250000
```

```
E=  -3.000000
S=   1.000000
X=   0.156250
```

floating point number suffix

- `F` for `float` type numbers (4-byte)
- `L` for `long double` type numbers (16-byte)
- `double` is the default type for floating point numbers (8-byte)

pointer type cast

- `x` : a 4-byte `float` type floating point number
- `p` : a pointer to a 4-byte integer number
- `p = &x` causes a type mismatch error
- `(int *)` a pointer type cast is necessary
- `p = (int *) &x`

extracting three bit fields

- `s` : 1-bit sign bit (b_{31})
- `e` : 8-bit exponent bit ($b_{30} \cdots b_{23}$)
- `m` : 23-bit mantissa bit ($b_{22} \cdots b_0$)

- `i` : a 4-byte (32-bit) integer number
- `i & 0x7fffffff` : extracing lower 23-bits (3+4+4+4+4+4)
- `i & 0xff` : extracing lower 8-bits (4+4)
- `i & 0x1` : extracing lower 1-bit

- `i >> 23` : shift `i` to the right by 23-bit positions
- after shifting out 23-bit mantissa field, the sign bit and 8-bit exponent fields are together aligned to the right
- `i >> 31` : shift `i` to the right by 31-bit positions
- after shifting out 31-bit exponent and mantissa fields, the sign bit is located in the least significant bit

constructing a number from the three bit fields

- $(-1)^s$ becomes $+1$ when `s=0`, otherwise -1
- the real mantissa must include the hidden 1
- the `m` field is converted into a fraction number ($0 \leq fraction \leq 1$)
- to make a fraction number, `m` must be divided by 2^{23}
- the real mantissa `M = 1.0 + (float) m / (1 << 23);`
- the type cast `(float)` is necessary to avoide integer division
- the exponent field is in the excess-127 code
- the real exponent is `E = e - 127`
- for a positive `E`, multiply $S \cdot M$ with 2^E
- for a negative `E`, divide $S \cdot M$ by 2^{-E}

0.4 Single Precision Number Range

implicit one

- mantissa is converted into 1 + faction number
- the binary point is assumed between 1 and the fraction number
- this 1 need not be stored (implicit one)

the range of float type

- unsigned 23-bit integer : $[0, 2^{23} - 1] = [0, 8388607]$
- fraction number : $[0, 0.999999881]$
- adding implicit one : $[1, 1.999999881]$
- unsigned 8-bit integer : $[0, 255] = [0], [1, 254], [255]$
- subtrac 127 offset : $[-126], [-126, 127], NaN$
- exponentiation : $[2^{-126}, 2^{127}] = [1.18 \times 10^{-38}, 1.70 \cdot 10^{+38}]$
- for normalized numbers
  - max positive number : $1.999999881 \cdot 1.70 \cdot 10^{+38} = +3.40 \cdot 10^{+38}$
  - min positive number : $1.0 \cdot 1.18 \times 10^{-38} = +1.18 \times 10^{-38}$
  - max negative number : $-1.0 \cdot 1.18 \times 10^{-38} = -1.18 \times 10^{-38}$
  - min negative number : $-1.999999881 \cdot 1.70 \times 10^{+38} = -3.40 \times 10^{+38}$
- for denormalized numbers
  - min fraction : $2^{-23} = 0.000000119 = 1.19 \times 10^{-7}$
  - exponentiation : $2^{-126} = 1.18 \times 10^{-38}$
  - min positive number $= 2^{-23} \times 2^{-126} = 2^{-149}$
    $= 1/(7.14 \times 10^{44}) = 1.4 \times 10^{-45}$
  - max negative number $= -2^{-23} \times 2^{-126} = -2^{-149}$
    $= -1/(7.14 \times 10^{44}) = -1.4 \times 10^{-45}$

the range of double type

- unsigned 52-bit integer : $[0, 2^{52} - 1] = [0, 9.01 \times 10^{15}]$
- fraction number : $[0, 1]$
- adding implicit one : $[1, 2]$
- unsigned 11-bit integer : $[0, 2047] = [0], [1, 2046], [2047]$
- subtrac 1023 offset : $[-1022], [-1022, 1023], NaN$
- exponentiation : $[2^{-1022}, 2^{1023}] = [2.23 \times 10^{-308}, 8.99 \times 10^{+307}]$
- for normalized numbers
  - max positive number : $2 \cdot 8.99 \cdot 10^{+307} = +1.80 \cdot 10^{+308}$
  - min positive number : $1.0 \cdot 2.23 \times 10^{-308} = +2.23 \times 10^{-308}$
  - max negative number : $-1.0 \cdot 2.23 \times 10^{-308} = -2.23 \times 10^{-308}$
  - min negative number : $-2 \cdot 8.99 \times 10^{+307} = -1.80 \cdot 10^{+308}$
- for denormalized numbers
  - min fraction : $2^{-52} = 2.22 \times 10^{-16}$
  - exponentiation : $2^{-1022} = 2.23 \times 10^{-308}$
  - min positive number = $2^{-52} \times 2^{-1022} = 2^{-1074}$ $= 1/(2.02 \times 10^{323}) = 4.95 \times 10^{-324}$
  - max negative number = -$2^{-52} \times 2^{-1022} = -2^{-1074}$ $= -1/(2.02 \times 10^{323}) = -4.95 \times 10^{-324}$

0.5 Pre/Post-Inc/Dec Dereferencing

```
::::::::::::::
h1.c
::::::::::::::
#include <stdio.h>

void pr(int *p, int x, char *s) {
  printf("p= %p *p= %d x= %d  %s\n", p, *p, x, s);
}

int main(void) {
  int A[] = {111, 222, 333, 444};
  int *p;
  int x = 0;

  printf("&A[0]= %p  A[0]= %d \n", &A[0], A[0]);
  printf("&A[1]= %p  A[1]= %d \n", &A[1], A[1]);
  printf("&A[2]= %p  A[2]= %d \n", &A[2], A[2]);
  printf("&A[3]= %p  A[3]= %d \n", &A[3], A[3]);
```

```
  printf("-------------------------\n");
  p = A+1;  pr(p, x, "");

  printf("-------------------------\n");
  p= A+1;   x = *p++;  pr(p, x, "x= *p++");
  p= A+1;   x = *p--;  pr(p, x, "x= *p--");

  printf("-------------------------\n");
  p= A+1;   x = *++p;    pr(p, x, "x= *++p");
  p= A+1;   x = *--p;    pr(p, x, "x= *--p");

  printf("-------------------------\n");
  p= A+1;   x = (*p)++;  pr(p, x, "x= (*p)++");
  p= A+1;   x = (*p)--;  pr(p, x, "x= (*p)--");

  printf("-------------------------\n");
  p= A+1;   x = ++*p;    pr(p, x, "x= ++*p");
  p= A+1;   x = --*p;    pr(p, x, "x= --*p");

}
::::::::::::::
h1.out
::::::::::::::
&A[0]= 0x7ffc94d70800  A[0]= 111
&A[1]= 0x7ffc94d70804  A[1]= 222
&A[2]= 0x7ffc94d70808  A[2]= 333
&A[3]= 0x7ffc94d7080c  A[3]= 444
-------------------------
p= 0x7ffc94d70804 *p= 222 x= 0
-------------------------
p= 0x7ffc94d70808 *p= 333 x= 222  x= *p++
p= 0x7ffc94d70800 *p= 111 x= 222  x= *p--
-------------------------
p= 0x7ffc94d70808 *p= 333 x= 333  x= *++p
p= 0x7ffc94d70800 *p= 111 x= 111  x= *--p
-------------------------
p= 0x7ffc94d70804 *p= 223 x= 222  x= (*p)++
p= 0x7ffc94d70804 *p= 222 x= 223  x= (*p)--
-------------------------
p= 0x7ffc94d70804 *p= 223 x= 223  x= ++*p
p= 0x7ffc94d70804 *p= 222 x= 222  x= --*p
```

1.A p= A+1; x = *p++;

- `*p` ≡ `A[1]` ≡ 222
- `*p++` ≡ `*(p++)`
- post-increment : access data first, then increment address

- `x` ≡ `A[1]` ≡ `222`
- `p` ≡ `A+2`
- `*p` ≡ `A[2]` ≡ `333`

1.B p= A+1; x = *p− −;

- `*p` ≡ `A[1]` ≡ `222`
- `*p--` ≡ `*(p--)`
- post-decrement : access data first, then decrement address
- `x` ≡ `A[1]` ≡ `222`
- `p` ≡ `A+0`
- `*p` ≡ `A[0]` ≡ `111`

2.A p= A+1; x = *++p;

- `*p` ≡ `A[1]` ≡ `222`
- `*++p` ≡ `*(++p)`
- pre-increment : increment address first, then access
- `p` ≡ `A+2`
- `x` ≡ `A[2]` ≡ `333`
- `*p` ≡ `A[2]` ≡ `333`

2.B p= A+1; x = *− −p;

- `*p` ≡ `A[1]` ≡ `222`
- `*--p` ≡ `*(--p)`
- pre-decrement : decrement address first, then access
- `p` ≡ `A+0`
- `x` ≡ `A[0]` ≡ `111`
- `*p` ≡ `A[2]` ≡ `111`

3.A p= A+1; x = (*p)++;

- `*p` ≡ `A[1]` ≡ `222`
- `(*p)++`
- post-increment : access data first, then increment data
- `x` ≡ `A[1]` ≡ `222`
- `(*p)++` ≡ `A[1]++`
- `*p` ≡ `A[1]` ≡ `223`

3.B p= A+1; x = (*p)− −;

- `*p` ≡ `A[1]` ≡ 222
- `(*p)--`
- post-decrement : access data first, then decrement data
- `x` ≡ `A[1]` ≡ 223
- `(*p)--` ≡ `A[1]--`
- `*p` ≡ `A[1]` ≡ 222

4.A p= A+1; x = ++*p;

- `*p` ≡ `A[1]` ≡ 222
- `++*p` ≡ `++(*p)`
- pre-increment : increment data first, then access data
- `(*p)++` ≡ `A[1]++`
- `x` ≡ `A[1]` ≡ 223
- `*p` ≡ `A[1]` ≡ 223

4.B p= A+1; x = − −*p;

- `*p` ≡ `A[1]` ≡ 222
- `--*p` ≡ `--(*p)`
- pre-decrement : decrement data first, then access data
- `(*p)--` ≡ `A[1]--`
- `x` ≡ `A[1]` ≡ 222
- `*p` ≡ `A[1]` ≡ 222

other combinations

- only 8 (=2·4) distinct combinations

(1a)	`*(++p)`	*	(2a)	`(++p)*`	(X)
(1b)	`*(--p)`	*	(2b)	`(--p)*`	(X)
(3a)	`*(p++)`	*	(4a)	`(p++)*`	(X)
(3b)	`*(p--)`	*	(4b)	`(p--)*`	(X)
(5a)	`++(*p)`	*	(6a)	`(*p)++`	*
(5b)	`--(*p)`	*	(6b)	`(*p)--`	*
(7a)	`(*p)++`	(6a)	(8a)	`++(*p)`	(5a)
(7b)	`(*p)--`	(6b)	(8b)	`--(*p)`	(5b)
(9a)	`*++p`	(1a)	(10a)	`++p*`	(X)
(9b)	`*--p`	(1b)	(10b)	`--p*`	(X)
(11a)	`*p++`	(3a)	(12a)	`p++*`	(X)
(11b)	`*p--`	(3b)	(12b)	`p--*`	(X)
(13a)	`++*p`	(5a)	(14a)	`*p++`	(3a)
(13b)	`--*p`	(5b)	(14b)	`*p--`	(3a)
(15a)	`*p++`	(3a)	(16a)	`++*p`	(5a)
(15b)	`*p--`	(3a)	(16b)	`--*p`	(5b)