Day07.C

Functions (1)
Definitions

Young W. Lim

December 9, 2017

1 Function Calls

1.1 pass by value and by reference

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

int func1( int x, int y) {
  int result;

  result = x + y;

  return result;
}



int func2( int *m, int *n) {
  int result;

  result = (*m) + (*n);

  return result;
}


int main (void) {
  int a = 3;
  int b = 2;
  int result;


  result = func1(a, b);

  printf("%d + %d = %d \n", a, b, result);


  result = func2(&a, &b);

  printf("%d + %d = %d \n", a, b, result);


}


::::::::::::::
t1.out
::::::::::::::
```

```
3 + 2 = 5
3 + 2 = 5
```

1.2 modifying caller's variables

```
::::::::::::::
t2.c
::::::::::::::
#include <stdio.h>

int func1( int x, int y) {
  int result;

  result = x + y;

  x = 0; y = 0;

  return result;
}



int func2( int *m, int *n) {
  int result;

  result = (*m) + (*n);

  (*m) = 0; (*n) = 0;

  return result;
}


int main (void) {
  int a = 3;
  int b = 2;
  int result;


  result = func1(a, b);

  printf("%d + %d = %d \n", a, b, result);
  printf("a= %d b= %d \n", a, b);


  result = func2(&a, &b);

  printf("%d + %d = %d \n", a, b, result);
  printf("a= %d b= %d \n", a, b);
```

```
}


::::::::::::::
t2.out
::::::::::::::
3 + 2 = 5
a= 3 b= 2
0 + 0 = 5
a= 0 b= 0
```

1.3 getchar examples

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


int main(void) {
  int c, i=0;

  //--- 1 -----------------------------------------------------------
  c = getchar();  i++;

  if (c == '\n')      printf("%d c: a new line character \n", i);
  else if (c == '\r') printf("%d c: a enter character \n", i);
  else if (c == '\t') printf("%d c: a tab character \n", i);
  else if (c == ' ')  printf("%d c: a space character \n", i);
  else if (c == EOF)  printf("%d c: the EOF character \n", i);
  else                printf("%d c: %c %d %x \n", i, c, c, c);

  //--- 2 -----------------------------------------------------------
  c = getchar();  i++;

  if (c == '\n')      printf("%d c: a new line character \n", i);
  else if (c == '\r') printf("%d c: a enter character \n", i);
  else if (c == '\t') printf("%d c: a tab character \n", i);
  else if (c == ' ')  printf("%d c: a space character \n", i);
  else if (c == EOF)  printf("%d c: the EOF character \n", i);
  else                printf("%d c: %c %d %x \n", i, c, c, c);

  //--- 3 -----------------------------------------------------------
  c = getchar();  i++;

  if (c == '\n')      printf("%d c: a new line character \n", i);
  else if (c == '\r') printf("%d c: a enter character \n", i);
```

```
  else if (c == '\t') printf("%d c: a tab character \n", i);
  else if (c == ' ')  printf("%d c: a space character \n", i);
  else if (c == EOF)  printf("%d c: the EOF character \n", i);
  else                printf("%d c: %c %d %x \n", i, c, c, c);

  //--- 4 ------------------------------------------------------
  c = getchar();  i++;

  if (c == '\n')      printf("%d c: a new line character \n", i);
  else if (c == '\r') printf("%d c: a enter character \n", i);
  else if (c == '\t') printf("%d c: a tab character \n", i);
  else if (c == ' ')  printf("%d c: a space character \n", i);
  else if (c == EOF)  printf("%d c: the EOF character \n", i);
  else                printf("%d c: %c %d %x \n", i, c, c, c);

  //--- 5 ------------------------------------------------------
  c = getchar();  i++;

  if (c == '\n')      printf("%d c: a new line character \n", i);
  else if (c == '\r') printf("%d c: a enter character \n", i);
  else if (c == '\t') printf("%d c: a tab character \n", i);
  else if (c == ' ')  printf("%d c: a space character \n", i);
  else if (c == EOF)  printf("%d c: the EOF character \n", i);
  else                printf("%d c: %c %d %x \n", i, c, c, c);


}


:::::::::::::::
t1.out
:::::::::::::::
abcde[Enter]---------------------------------
abcde
1 c: a 97 61
2 c: b 98 62
3 c: c 99 63
4 c: d 100 64
5 c: e 101 65

a[space]b[tab]bcdefg[Enter]-----------------
a b bcdefg
1 c: a 97 61
2 c: a space character
3 c: b 98 62
4 c: a tab character
5 c: b 98 62

ab[Ctrl-d][Ctrl-d][Ctrl-d]z[Enter]-----------
ab1 c: a 97 61
```

```
2 c: b 98 62
3 c: the EOF character
4 c: the EOF character
z
5 c: z 122 7a
```

- repeat 5 times the followings
  - get one character
  - check if it is a new line character
  - check if it is a enter character
  - check if it is a tab character
  - check if it is a space character
  - check if it is EOF character
  - otherwise, print it as a character, a decimal, a hexadecimal form
- getinput() takes effect only after [Enter] or [Ctrl-d] is pressed
- think there is an internal buffer
- the program waits until [Enter] or [Ctrl-d] is pressed
- 1st input key sequence abcde[Enter]
- 5 characters a-b-c-d-e are displayed
- 2nd input key sequence a[space]b[tab]bcdefg[Enter]
- 5 characters a-[space]-b-[tab]-b-c-d are displayed
- in Linux, EOF is [Ctrl-d]
- if there are characters in the buffer which are not processed
- then the key [Ctrl-d] commands them to be processed
- (input characters in this case)
- if there is no character to be processed
- then the key [Ctrl-d] is displayed as EOT
- the key [Ctrl-z] is a key for suspending the execution of a program
- 3rd input key sequence a[Ctrl-z][Ctrl-d]bcdefg does not work
- another input key sequence ab[Ctrl-d][Ctrl-d][Ctrl-d]z[Enter]
- displays only two EOF

1.4 using a for loop

```
::::::::::::::
t2.c
::::::::::::::
#include <stdio.h>


int main(void) {
  int c, i=0;

  for (i=0; i<5; ++i) {
    c = getchar();

    if (c == '\n')      printf("%d c: a new line character \n", i);
    else if (c == '\r') printf("%d c: a enter character \n", i);
    else if (c == '\t') printf("%d c: a tab character \n", i);
    else if (c == ' ')  printf("%d c: a space character \n", i);
    else if (c == EOF)  printf("%d c: the EOF character \n", i);
    else                printf("%d c: %c %d %x \n", i, c, c, c);
  }


}

::::::::::::::
t2.out
::::::::::::::

abcde
0 c: a 97 61
1 c: b 98 62
2 c: c 99 63
3 c: d 100 64
4 c: e 101 65

a b bcdefg
0 c: a 97 61
1 c: a space character
2 c: b 98 62
3 c: a tab character
4 c: b 98 62

ab0 c: a 97 61
1 c: b 98 62
2 c: the EOF character
3 c: the EOF character
z
4 c: z 122 7a
```

1.5 using a function

```
::::::::::::::
t3.c
::::::::::::::
#include <stdio.h>

void myfunc(int c, int i) {
    if (c == '\n')      printf("%d c: a new line character \n", i);
    else if (c == '\r') printf("%d c: a enter character \n", i);
    else if (c == '\t') printf("%d c: a tab character \n", i);
    else if (c == ' ')  printf("%d c: a space character \n", i);
    else if (c == EOF)  printf("%d c: the EOF character \n", i);
    else                printf("%d c: %c %d %x \n", i, c, c, c);
}



int main(void) {
  int c, i=0;

  for (i=0; i<5; ++i) {
    c = getchar();

    myfunc(c, i);
  }


}

::::::::::::::
t3.out
::::::::::::::

abcde
0 c: a 97 61
1 c: b 98 62
2 c: c 99 63
3 c: d 100 64
4 c: e 101 65

a b bcdefg
0 c: a 97 61
1 c: a space character
2 c: b 98 62
3 c: a tab character
4 c: b 98 62

ab0 c: a 97 61
1 c: b 98 62
```

```
2 c: the EOF character
3 c: the EOF character
z
4 c: z 122 7a
```

1.6 using a function prototype

```
::::::::::::::
t4.c
::::::::::::::
#include <stdio.h>


void myfunc(int c, int i);


int main(void) {
  int c, i=0;

  for (i=0; i<5; ++i) {
    c = getchar();

    myfunc(c, i);
  }


}


void myfunc(int c, int i) {
    if (c == '\n')      printf("%d c: a new line character \n", i);
    else if (c == '\r') printf("%d c: a enter character \n", i);
    else if (c == '\t') printf("%d c: a tab character \n", i);
    else if (c == ' ')  printf("%d c: a space character \n", i);
    else if (c == EOF)  printf("%d c: the EOF character \n", i);
    else                printf("%d c: %c %d %x \n", i, c, c, c);
}


::::::::::::::
t4.out
::::::::::::::

abcde
0 c: a 97 61
1 c: b 98 62
2 c: c 99 63
3 c: d 100 64
4 c: e 101 65
```

```
a b bcdefg
0 c: a 97 61
1 c: a space character
2 c: b 98 62
3 c: a tab character
4 c: b 98 62

ab0 c: a 97 61
1 c: b 98 62
2 c: the EOF character
3 c: the EOF character
z
4 c: z 122 7a
```

1.7 using a switch statement

```
::::::::::::::
t5.c
::::::::::::::
#include <stdio.h>


void myfunc(int c, int i);


int main(void) {
  int c, i=0;

  for (i=0; i<5; ++i) {
    c = getchar();

    myfunc(c, i);
  }


}


void myfunc(int c, int i) {
  switch (c) {
    case '\n' : printf("%d c: a new line character \n", i); break;
    case '\r' : printf("%d c: a enter character \n", i);    break;
    case '\t' : printf("%d c: a tab character \n", i);      break;
    case ' '  : printf("%d c: a space character \n", i);    break;
    case EOF  : printf("%d c: the EOF character \n", i);    break;
    default   : printf("%d c: %c %d %x \n", i, c, c, c);    break;
  }

}
```

```
::::::::::::::
t5.out
::::::::::::::

abcde
0 c: a 97 61
1 c: b 98 62
2 c: c 99 63
3 c: d 100 64
4 c: e 101 65

a b bcdefg
0 c: a 97 61
1 c: a space character
2 c: b 98 62
3 c: a tab character
4 c: b 98 62

ab0 c: a 97 61
1 c: b 98 62
2 c: the EOF character
3 c: the EOF character
z
4 c: z 122 7a
```

1.8 using a function prototype

```
::::::::::::::
t4.c
::::::::::::::
#include <stdio.h>

void func1( void ) ;
void func2( void ) ;
void func3( void ) ;


void func1( void ) {
  printf("func1: called \n");

  func2();
}

void func2( void ) {
  printf("func2: called \n");

  func3();
}
```

```
void func3( void ) {
  printf("func3: called \n");
}

int main( void ) {
   printf("----------------\n");
   func1();

   printf("----------------\n");
   func2();

   printf("----------------\n");
   func3();
}
```

```
:::::::::::::::
t4.out
:::::::::::::::
----------------
func1: called
func2: called
func3: called
----------------
func2: called
func3: called
----------------
func3: called
```

func1 calls func2, and func2 calls func3

- func1 calls func2
- func1 needs func2's prototype or definition before its own definition
- func2 calls func3
- func2 needs func3's prototype or definition before its own definition
- functions are defined in the file : func1 - func2 - func3
- before func2's definition, there are no func1's definition
- before func3's definition, there are no func2's definition
- therefore, at the beginning put prototypes of func1, func2, and func3
- if func1 is called in the main, func1 calls func2, and func2 calls func3
- if func1 is called in the main, func1 call func2.

1.9 using a function prototype

```
::::::::::::::
t4.c
::::::::::::::
#include <stdio.h>

int iSum(int k) {
  int i, S=0;

  for (i=1; i<k+1; ++i) S += i;
  return S;
}

int rSum(int k) {
  if (k==1) return 1;
  else return (k + rSum(k-1));
}

int main(void) {
  int S;

  S = iSum(10);
  printf("S= %d \n", S);

  S = rSum(10);
  printf("S= %d \n", S);
}
```

```
::::::::::::::
t4.out
::::::::::::::
S= 55
S= 55
```

recusive Sum

$$\sum_{k=1}^{n} k = n + \sum_{k=1}^{n-1} k$$

1.10 Function Pointers

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

```
int add(int a, int b) ;
int sub(int a, int b) ;
int mul(int a, int b) ;
int div(int a, int b) ;


int main(void) {
  int (*fun) (int a, int b);
  int x= 30, y= 20, z;


  printf("&add= %p \n", &add);
  printf("&sub= %p \n", &sub);
  printf("&mul= %p \n", &mul);
  printf("&div= %p \n", &div);

  puts("----------------------------------------");
  fun = &add;
  z = (*fun)(x, y);

  printf("fun= &add; %5d= (*fun)(%d, %d);\n", z, x, y);

  fun = &sub;
  z = (*fun)(x, y);

  printf("fun= &sub; %5d= (*fun)(%d, %d);\n", z, x, y);

  fun = &mul;
  z = (*fun)(x, y);

  printf("fun= &mul; %5d= (*fun)(%d, %d);\n", z, x, y);

  fun = &sub;
  z = (*fun)(x, y);

  printf("fun= &sub; %5d= (*fun)(%d, %d);\n", z, x, y);

  puts("----------------------------------------");
  fun = add;
  z = fun(x, y);

  printf("fun= add;  %5d= fun(%d, %d);\n", z, x, y);

  fun = sub;
  z = fun(x, y);

  printf("fun= sub;  %5d= fun(%d, %d);\n", z, x, y);

  fun = mul;
```

```
  z = fun(x, y);

  printf("fun= mul;  %5d= fun(%d, %d);\n", z, x, y);

  fun = sub;
  z = fun(x, y);

  printf("fun= sub;  %5d= fun(%d, %d);\n", z, x, y);
}



int add(int a, int b) {
  return (a + b);
}

int sub(int a, int b) {
  return (a - b);
}

int mul(int a, int b) {
  return (a * b);
}

int div(int a, int b) {
  return (a / b);
}
```

```
::::::::::::::
h1.out
::::::::::::::
&add= 0x4007bf
&sub= 0x4007d3
&mul= 0x4007e5
&div= 0x4007f8
----------------------------------------
fun= &add;    50= (*fun)(30, 20);
fun= &sub;    10= (*fun)(30, 20);
fun= &mul;   600= (*fun)(30, 20);
fun= &sub;    10= (*fun)(30, 20);
----------------------------------------
fun= add;     50= fun(30, 20);
fun= sub;     10= fun(30, 20);
fun= mul;    600= fun(30, 20);
fun= sub;     10= fun(30, 20);
```

function prototypes

- all these 4 functions take two integer arguments return integer value
  - `int add(int a, int b) ;`
  - `int sub(int a, int b) ;`
  - `int mul(int a, int b) ;`
  - `int div(int a, int b) ;`
- the corresponding function definitions are after the main function

function pointers

- the function pointer definition in the main function
  `int ( *fun ) (int a, int b);`
- the function pointer variable fun can point to any function whose function prototype has the form of
  `int any_func (int, int);`
- `any_func` ⇔ `*fun` `&any_func` ⇔ `fun`
  - `add` ⇔ `*fun` `&add` ⇔ `fun`
  - `sub` ⇔ `*fun` `&sub` ⇔ `fun`
  - `mul` ⇔ `*fun` `&mul` ⇔ `fun`
  - `div` ⇔ `*fun` `&div` ⇔ `fun`
- function addresses
  - `&add= 0x4007bf`
  - `&sub= 0x4007d3`
  - `&mul= 0x4007e5`
  - `&div= 0x4007f8`
- using a function pointer, different functions can be called by the same function call `(*fun) (x, y)`
- the address-of and dereference operators can be omitted when function pointers are involved.
- the same results can be obtained even if we omit `&` and `*`
  - `fun = add; z = fun(x, y);`
  - `fun = sub; z = fun(x, y);`
  - `fun = mul; z = fun(x, y);`
  - `fun = sub; z = fun(x, y);`