Lecture08: Program Development & Preprocessor
11/12/2012

Slides modified from Yin Lou, Cornell CS2022: Introduction to C

1

Outline

- Midterm results
- Program development & midterm solutions
- Preprocessor

2

Midterm results

- 72 students
- Avg: 18.7
- Std: 15.3
- What happened?
  - Not enough time?
  - Problems too difficult?

C2012Mid

Problem Solving Status

Score	Mid - 1	Mid - 2	Mid - 3	Mid - 4	Mid - 5
10	53 (85.43%)	19 (23.46%)	14 (17.28%)	11 (13.58%)	18 (22.22%)
9	1 (1.23%)	1 (1.23%)	2 (2.47%)	1 (1.23%)	1 (1.23%)
8	1 (1.23%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	1 (1.23%)
7	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
6	0 (0.00%)	1 (1.23%)	0 (0.00%)	1 (1.23%)	1 (1.23%)
5	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	2 (2.47%)
4	1 (1.23%)	0 (0.00%)	0 (0.00%)	2 (2.47%)	0 (0.00%)
3	3 (3.70%)	2 (2.47%)	5 (6.17%)	0 (0.00%)	0 (0.00%)
2	5 (6.17%)	4 (4.94%)	5 (6.17%)	0 (0.00%)	4 (4.94%)
1	2 (2.47%)	9 (11.11%)	8 (9.88%)	1 (1.23%)	5 (6.17%)
0	2 (2.47%)	9 (11.11%)	2 (2.47%)	2 (2.47%)	5 (6.17%)
Sleeping	13 (16.05%)	36 (44.44%)	45 (55.56%)	63 (77.78%)	44 (54.32%)

3

Take-home assignment #7

- For those who came to take the midterm exam (i.e., submitted something on the judge system on the exam day), you have a chance to earn 1/2 credit back on the lost points for problems 3-5 (not problems 1-2 because I will talk about the solutions today).
- Say midterm score is 5 on the 3rd problem.
  - Get 10 points on the 3rd problem in assignment #7
  - Your midterm score for the 3rd problem = (5 + 10)/2 = 7.5

4

Program development

- A Common problem
  - Know all C commands but don't know how to use them to write a program.
- Top-down approach – always work!
  - Start by writing down the **main steps** **(in words/comments, not code)** of a program.
    - Consider each step to be a function
  - For each difficult main step, break it down into smaller **steps** **(again in words/comments, not code).**
  - Apply this "divide-and-conquer" approach until the steps are **simple enough** to code.
  - Write code

5

Midterm Problem 1

Problem 1

Given N distinct points on a two dimensional plane, find the two points which have the longest distance.

Input Format

The first line contains an integer N. (2 <= N <= 100)
Next N lines contain two integers Xi and Yi that represent the position. (-10000 <= Xi, Yi <= 10000)

Output Format

Output the positions of the farthest pair in two lines with the same format of input.
Print the point with smaller x first. If they have same x, print the point with smaller y first.

Assume that there is only one farthest pair.

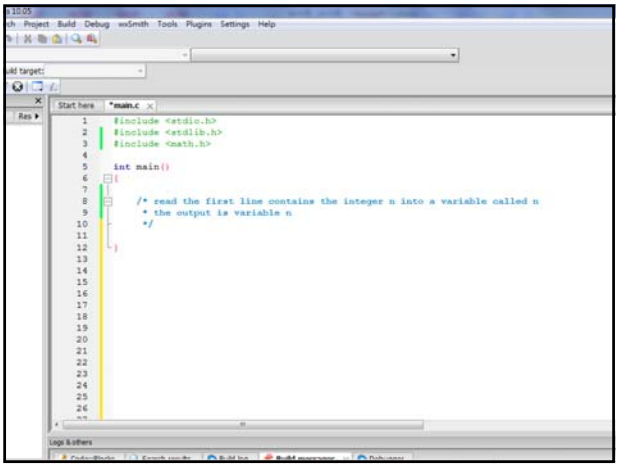
Sample Input	Sample Output
4 -1 0 0 0 1 0 1 10	-1 0 1 10

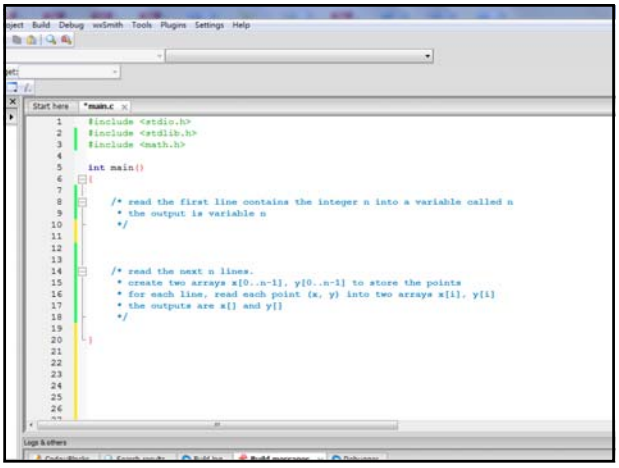
6

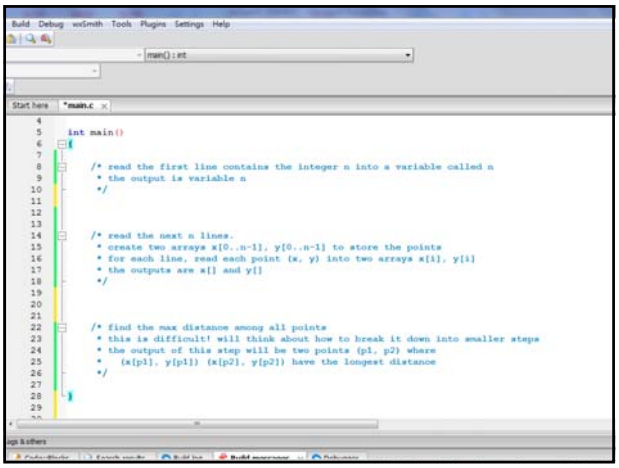
Program development

- A Common problem
  - Know all C commands but don't know how to use them to write a program.
- Top-down approach – always work!
  - Start by writing down the **main steps** (in words/comments, not code) of a program.
    - Consider each step to be a function
  - For each difficult main step, break it down into smaller **steps** (again in words/comments, not code).
  - Apply this "divide-and-conquer" approach until the steps are **simple enough** to code.
  - Write code

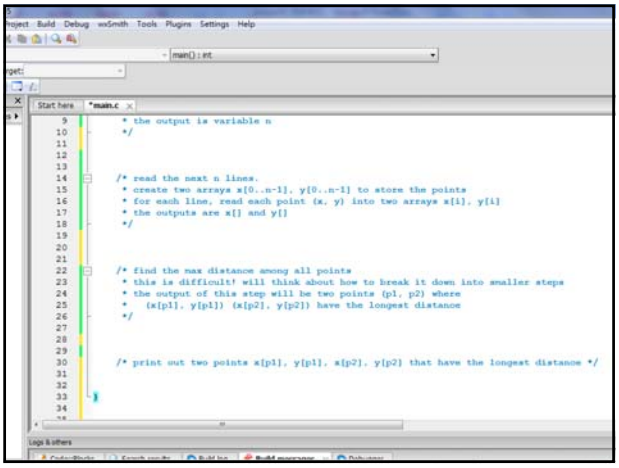
7







```
int main()
{
    /* read the first line contains the integer n into a variable called n
     * the output is variable n
     */


    /* read the next n lines.
     * create two arrays x[0..n-1], y[0..n-1] to store the points
     * for each line, read each point (x, y) into two arrays x[i], y[i]
     * the outputs are x[] and y[]
     */


    /* find the max distance among all points
     * this is difficult! will think about how to break it down into smaller steps
     * the output of this step will be two points (p1, p2) where
     *   (x[p1], y[p1]) (x[p2], y[p2]) have the longest distance
     */
}
```

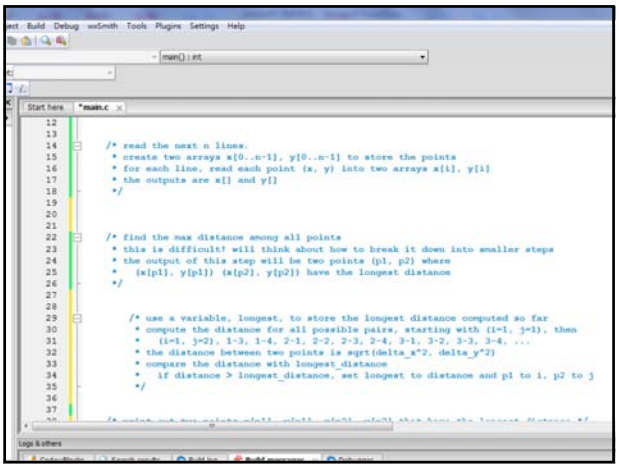


```
     * the output is variable n
     */


    /* read the next n lines.
     * create two arrays x[0..n-1], y[0..n-1] to store the points
     * for each line, read each point (x, y) into two arrays x[i], y[i]
     * the outputs are x[] and y[]
     */


    /* find the max distance among all points
     * this is difficult! will think about how to break it down into smaller steps
     * the output of this step will be two points (p1, p2) where
     *   (x[p1], y[p1]) (x[p2], y[p2]) have the longest distance
     */


    /* print out two points x[p1], y[p1], x[p2], y[p2] that have the longest distance */

}
```

Program development

- A Common problem
  - Know all C commands but don't know how to use them to write a program.
- Top-down approach – always work!
  - Start by writing down the **main steps** (in words/comments, not code) of a program.
    - Consider each step to be a function
  - For each difficult main step, break it down into smaller **steps** (again in words/comments, not code).
  - Apply this "divide-and-conquer" approach until the steps are **simple enough** to code.
  - Write code

12

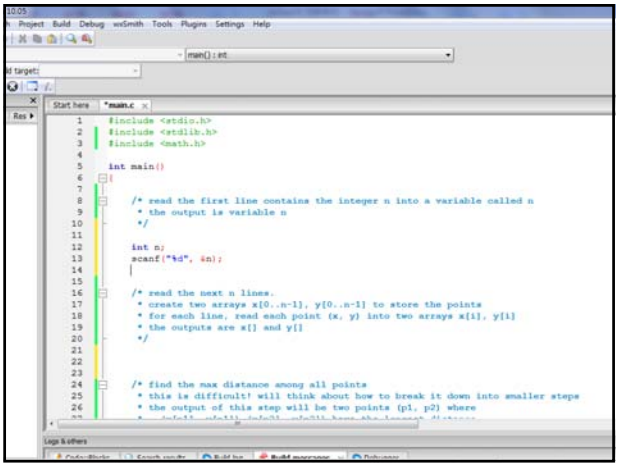


```
12
13
14      /* read the next n lines.
15       * create two arrays x[0..n-1], y[0..n-1] to store the points
16       * for each line, read each point (x, y) into two arrays x[i], y[i]
17       * the outputs are x[] and y[]
18       */
19
20
21
22      /* find the max distance among all points
23       * this is difficult! will think about how to break it down into smaller steps
24       * the output of this step will be two points (p1, p2) where
25       *   (x[p1], y[p1]) (x[p2], y[p2]) have the longest distance
26       */
27
28
29          /* use a variable, longest, to store the longest distance computed so far
30           * compute the distance for all possible pairs, starting with (i=1, j=1), then
31           *   (i=1, j=2), 1-3, 1-4, 2-1, 2-2, 2-3, 2-4, 3-1, 3-2, 3-3, 3-4, ...
32           * the distance between two points is sqrt(delta_x^2, delta_y^2)
33           * compare the distance with longest_distance
34           *   if distance > longest_distance, set longest to distance and p1 to i, p2 to j
35           */
36
37
```

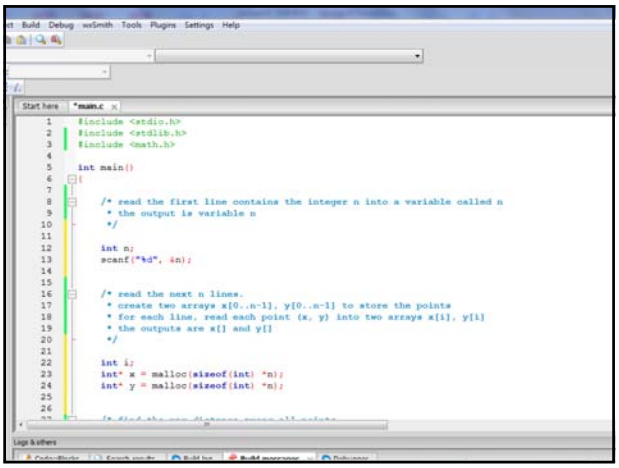
Program development

- A Common problem
  - Know all C commands but don't know how to use them to write a program.
- Top-down approach – always work!
  - Start by writing down the **main steps** (in words/comments, not code) of a program.
    - Consider each step to be a function
  - For each difficult main step, break it down into smaller **steps** (again in words/comments, not code).
  - Apply this "divide-and-conquer" approach until the steps are **simple enough** to code.
  - Write code!

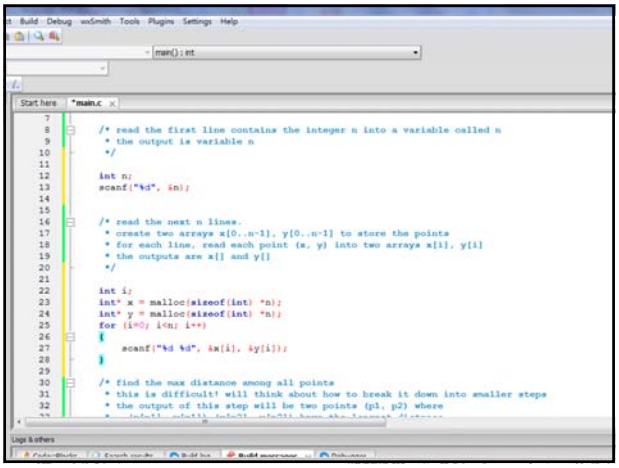
14



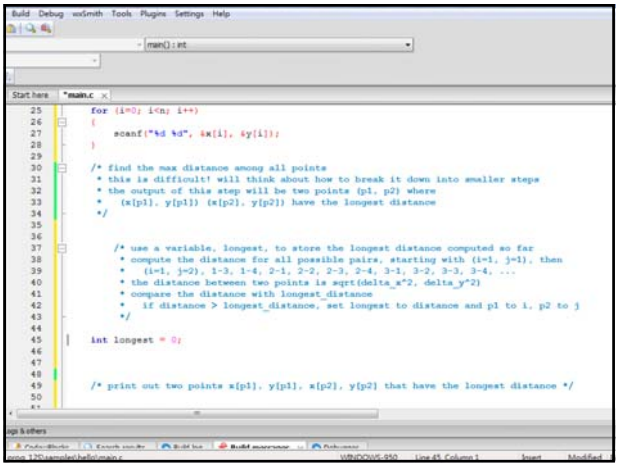
```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4
5   int main()
6   {
7
8       /* read the first line contains the integer n into a variable called n
9        * the output is variable n
10       */
11
12      int n;
13      scanf("%d", &n);
14
15
16      /* read the next n lines.
17       * create two arrays x[0..n-1], y[0..n-1] to store the points
18       * for each line, read each point (x, y) into two arrays x[i], y[i]
19       * the outputs are x[] and y[]
20       */
21
22
23
24      /* find the max distance among all points
25       * this is difficult! will think about how to break it down into smaller steps
26       * the output of this step will be two points (p1, p2) where
```



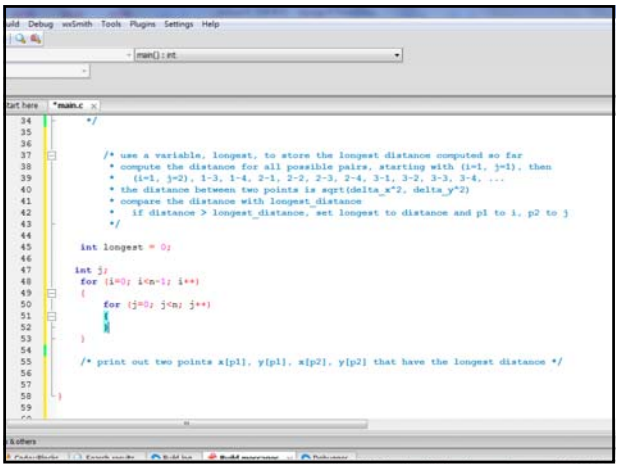
```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{

    /* read the first line contains the integer n into a variable called n
     * the output is variable n
     */

    int n;
    scanf("%d", &n);

    /* read the next n lines.
     * create two arrays x[0..n-1], y[0..n-1] to store the points
     * for each line, read each point (x, y) into two arrays x[i], y[i]
     * the outputs are x[] and y[]
     */

    int i;
    int* x = malloc(sizeof(int) *n);
    int* y = malloc(sizeof(int) *n);
```



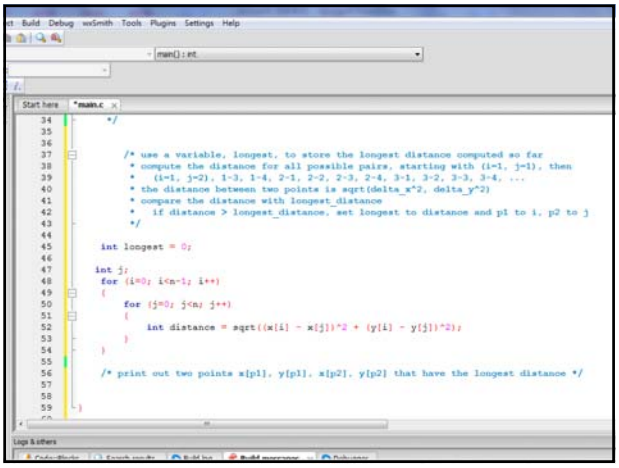
```c
    /* read the first line contains the integer n into a variable called n
     * the output is variable n
     */

    int n;
    scanf("%d", &n);

    /* read the next n lines.
     * create two arrays x[0..n-1], y[0..n-1] to store the points
     * for each line, read each point (x, y) into two arrays x[i], y[i]
     * the outputs are x[] and y[]
     */

    int i;
    int* x = malloc(sizeof(int) *n);
    int* y = malloc(sizeof(int) *n);
    for (i=0; i<n; i++)
    {
        scanf("%d %d", &x[i], &y[i]);
    }

    /* find the max distance among all points
     * this is difficult! will think about how to break it down into smaller steps
     * the output of this step will be two points (p1, p2) where
```



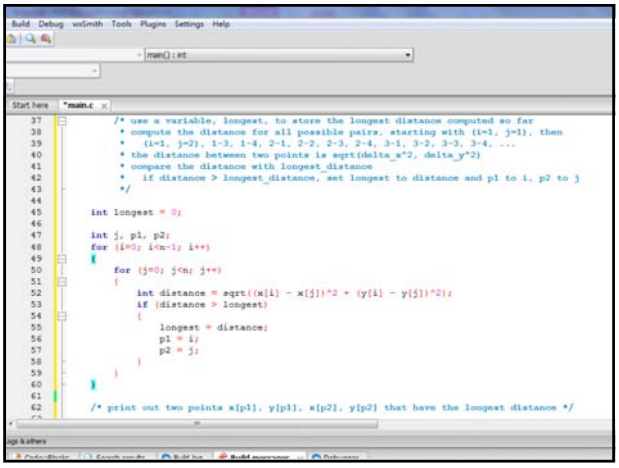
```c
    for (i=0; i<n; i++)
    {
        scanf("%d %d", &x[i], &y[i]);
    }

    /* find the max distance among all points
     * this is difficult! will think about how to break it down into smaller steps
     * the output of this step will be two points (p1, p2) where
     *   (x[p1], y[p1]) (x[p2], y[p2]) have the longest distance
     */


        /* use a variable, longest, to store the longest distance computed so far
         * compute the distance for all possible pairs, starting with (i=1, j=1), then
         *   (i=1, j=2), 1-3, 1-4, 2-1, 2-2, 2-3, 2-4, 3-1, 3-2, 3-3, 3-4, ...
         * the distance between two points is sqrt(delta_x^2, delta_y^2)
         * compare the distance with longest_distance
         *   if distance > longest_distance, set longest to distance and p1 to i, p2 to j
         */

    int longest = 0;


    /* print out two points x[p1], y[p1], x[p2], y[p2] that have the longest distance */
```



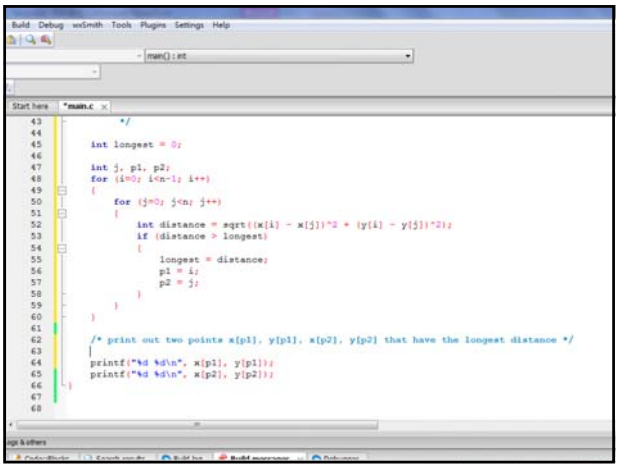
```
*/

/* use a variable, longest, to store the longest distance computed so far
 * compute the distance for all possible pairs, starting with (i=1, j=1), then
 *   (i=1, j=2), 1-3, 1-4, 2-1, 2-2, 2-3, 2-4, 3-1, 3-2, 3-3, 3-4, ...
 * the distance between two points is sqrt(delta_x^2, delta_y^2)
 * compare the distance with longest_distance
 *   if distance > longest_distance, set longest to distance and p1 to i, p2 to j
 */

int longest = 0;

int j;
for (i=0; i<n-1; i++)
{
    for (j=0; j<n; j++)
    {
    }
}

/* print out two points x[p1], y[p1], x[p2], y[p2] that have the longest distance */
```



```
*/

/* use a variable, longest, to store the longest distance computed so far
 * compute the distance for all possible pairs, starting with (i=1, j=1), then
 *   (i=1, j=2), 1-3, 1-4, 2-1, 2-2, 2-3, 2-4, 3-1, 3-2, 3-3, 3-4, ...
 * the distance between two points is sqrt(delta_x^2, delta_y^2)
 * compare the distance with longest_distance
 *   if distance > longest_distance, set longest to distance and p1 to i, p2 to j
 */

int longest = 0;

int j;
for (i=0; i<n-1; i++)
{
    for (j=0; j<n; j++)
    {
        int distance = sqrt((x[i] - x[j])^2 + (y[i] - y[j])^2);
    }
}

/* print out two points x[p1], y[p1], x[p2], y[p2] that have the longest distance */
```



```
/* use a variable, longest, to store the longest distance computed so far
 * compute the distance for all possible pairs, starting with (i=1, j=1), then
 *   (i=1, j=2), 1-3, 1-4, 2-1, 2-2, 2-3, 2-4, 3-1, 3-2, 3-3, 3-4, ...
 * the distance between two points is sqrt(delta_x^2, delta_y^2)
 * compare the distance with longest_distance
 *   if distance > longest_distance, set longest to distance and p1 to i, p2 to j
 */

int longest = 0;

int j, p1, p2;
for (i=0; i<n-1; i++)
{
    for (j=0; j<n; j++)
    {
        int distance = sqrt((x[i] - x[j])^2 + (y[i] - y[j])^2);
        if (distance > longest)
        {
            longest = distance;
            p1 = i;
            p2 = j;
        }
    }
}

/* print out two points x[p1], y[p1], x[p2], y[p2] that have the longest distance */
```



Systematic way of writing programs

- Like writing an essay.
- Start with an outline that describes what you are going to write.
- Okay to write the comments in Chinese.

23

Midterm Problem 2

Problem 2

Consider a binary string which contains only '0' or '1', print this string with a recursive way.

First we can divide the string into two parts(left and right) of the same length, then we can calculate how many '1' there are in each part, then you should print the part with more '1' first , then print the part with less '1', then recursively print those parts in the same way.
For example, consider a string '00110111', then we should print '0111', then print '0011'.
For '0111', the printed result is 1110, for '0011', the printed result is 1100 , so the final result is 11101100.

Note that the length of the string must be 2^n.
Print the left part first if two parts have same number of '1'.

Input Format

One line contains a string S. (length of S = 1, 2, 4, 8, 16, ..., 2^20)

Output Format

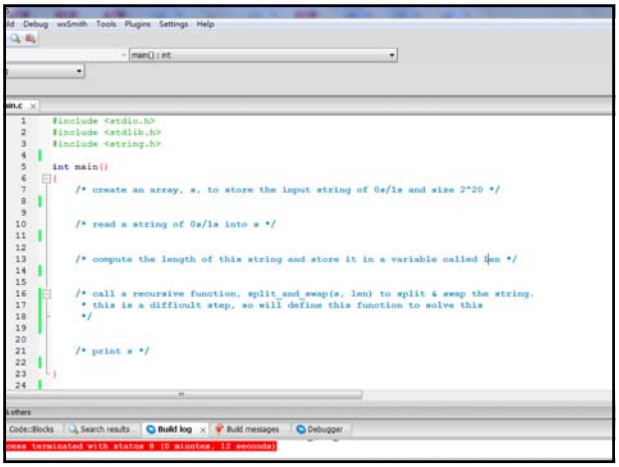
Print the string by the way in the description.

Sample Input	Sample Output
00110111	11101100

Program development

- A Common problem
  - Know all C commands but don't know how to use them to write a program.
- Top-down approach – always work!
  - Start by writing down the **main steps** (in words/comments, not code) of a program.
    - Consider each step to be a function
  - For each difficult main step, break it down into smaller **steps** (again in words/comments, not code).
  - Apply this "divide-and-conquer" approach until the steps are **simple enough** to code.
  - Write code

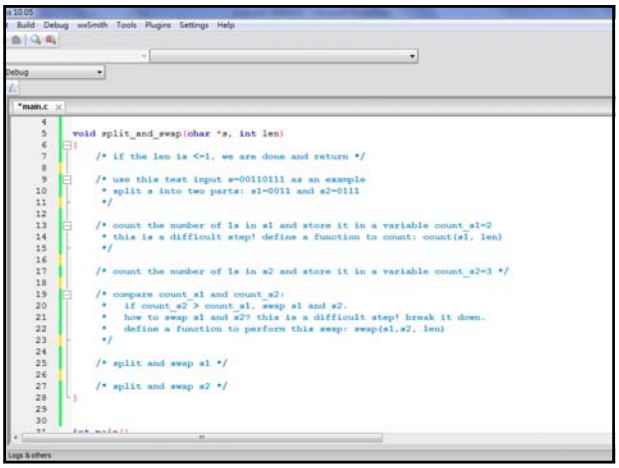
25



Program development

- A Common problem
  - Know all C commands but don't know how to use them to write a program.
- Top-down approach – always work!
  - Start by writing down the **main steps** (in words/comments, not code) of a program.
    - Consider each step to be a function
  - For each difficult main step, break it down into smaller **steps** (again in words/comments, not code).
  - Apply this "divide-and-conquer" approach until the steps are **simple enough** to code.
  - Write code

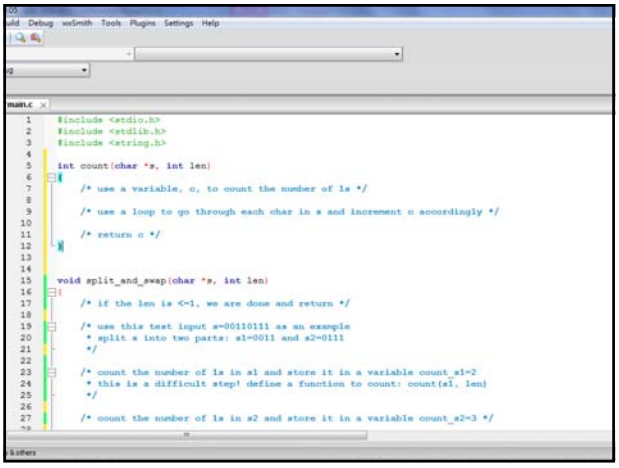
27



Program development

- A Common problem
  - Know all C commands but don't know how to use them to write a program.
- Top-down approach – always work!
  - Start by writing down the **main steps** (in words/comments, not code) of a program.
    - Consider each step to be a function
  - For each difficult main step, break it down into smaller **steps** **(again in words/comments, not code).**
  - Apply this "divide-and-conquer" approach until the steps are **simple enough** to code.
  - Write code

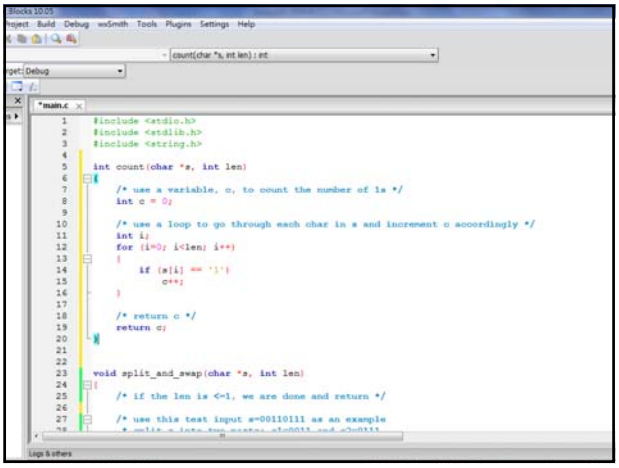
29

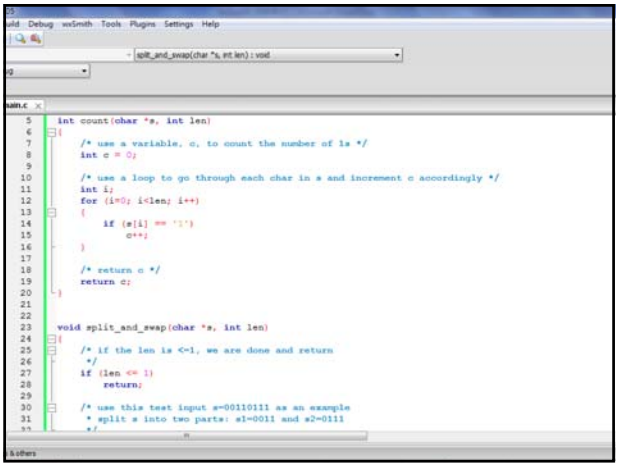


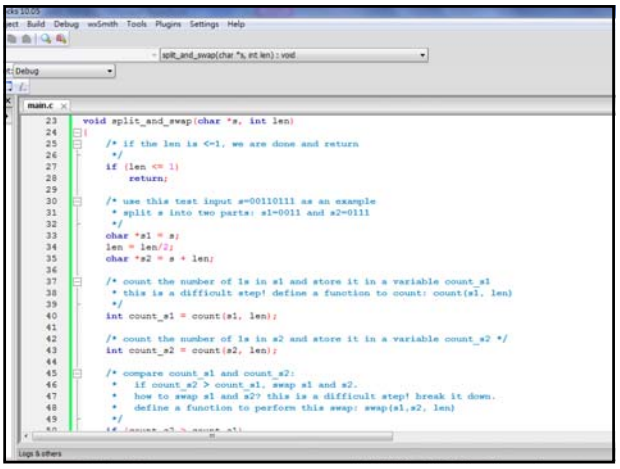
Program development

- A Common problem
  - Know all C commands but don't know how to use them to write a program.
- Top-down approach – always work!
  - Start by writing down the **main steps** (in words/comments, not code) of a program.
    - Consider each step to be a function
  - For each difficult main step, break it down into smaller **steps** (again in words/comments, not code).
  - Apply this "divide-and-conquer" approach until the steps are **simple enough** to code.
  - Write code!

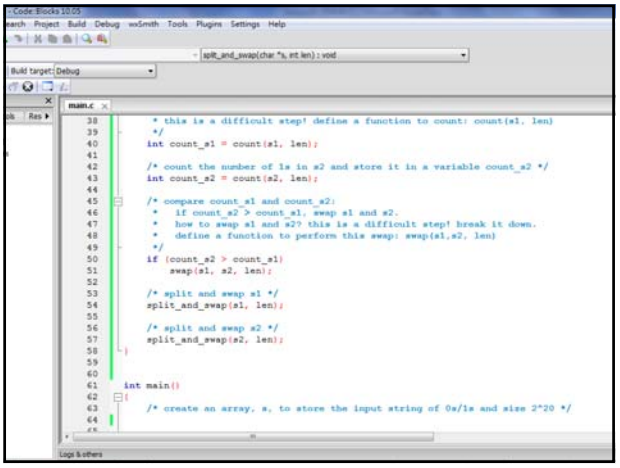
31



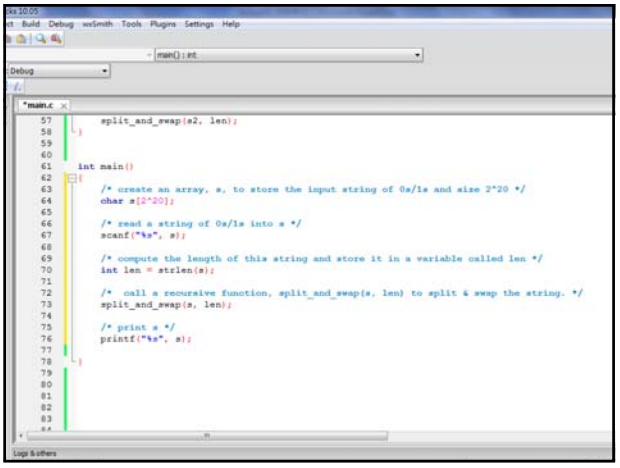




```
void split_and_swap(char *s, int len)
{
    /* if the len is <=1, we are done and return
     */
    if (len <= 1)
        return;

    /* use this test input s=00110111 as an example
     * split s into two parts: s1=0011 and s2=0111
     */
    char *s1 = s;
    len = len/2;
    char *s2 = s + len;

    /* count the number of 1s in s1 and store it in a variable count_s1
     * this is a difficult step! define a function to count: count(s1, len)
     */
    int count_s1 = count(s1, len);

    /* count the number of 1s in s2 and store it in a variable count_s2 */
    int count_s2 = count(s2, len);

    /* compare count_s1 and count_s2:
     *   if count_s2 > count_s1, swap s1 and s2.
     *   how to swap s1 and s2? this is a difficult step! break it down.
     *   define a function to perform this swap: swap(s1,s2, len)
     */
```



```
     * this is a difficult step! define a function to count: count(s1, len)
     */
    int count_s1 = count(s1, len);

    /* count the number of 1s in s2 and store it in a variable count_s2 */
    int count_s2 = count(s2, len);

    /* compare count_s1 and count_s2:
     *   if count_s2 > count_s1, swap s1 and s2.
     *   how to swap s1 and s2? this is a difficult step! break it down.
     *   define a function to perform this swap: swap(s1,s2, len)
     */
    if (count_s2 > count_s1)
        swap(s1, s2, len);

    /* split and swap s1 */
    split_and_swap(s1, len);

    /* split and swap s2 */
    split_and_swap(s2, len);
}

int main()
{
    /* create an array, s, to store the input string of 0s/1s and size 2^20 */
```



```
    split_and_swap(s2, len);
}

int main()
{
    /* create an array, s, to store the input string of 0s/1s and size 2^20 */
    char s[2^20];

    /* read a string of 0s/1s into s */
    scanf("%s", s);

    /* compute the length of this string and store it in a variable called len */
    int len = strlen(s);

    /*  call a recursive function, split_and_swap(s, len) to split & swap the string. */
    split_and_swap(s, len);

    /* print s */
    printf("%s", s);
}
```

Preprocessors

C program -> Preprocessor -> Modified C program -> Compile

```
#include <stdio.h>
#define FREEZING_PT 32.0
#define SCALE_FACTOR (5.0 / 9.0)
int main()
{
   float fahrenheit, celsius;
   printf("Enter Fahrenheit temperature: ");
   scanf("%f", &fahrenheit);
   celsius = (fahrenheit – FREEZING_PT) * SCALE_FACTOR;
   printf("Celsius equivalent is: %.1f\n", celsius);
}
```

```
#include <stdio.h>
int main()
{
   float fahrenheit, celsius;
   printf("Enter Fahrenheit temperature: ");
   scanf("%f", &fahrenheit);
   celsius = (fahrenheit – 32.0) * (5.0 / 9.0);
   printf("Celsius equivalent is: %.1f\n", celsius);
}
```

37

Preprocessor

- Commands to the compiler
- Include files, shortcuts, conditional compilation
- Common Preprocessor Commands

```
#include
#define
#ifdef / #ifndef
#
##
```

#include: Header Files

- Includes files: Liberally copy-paste
- Typically header files
- Header files:
  - Declares
    - External functions
    - Variable types
    - External global variables
  - Typically named *.h

#include: Header Files

- mylib.h

```
int max(int a, int b);
```

- Mylib.c

```
#include "mylib.h"
int max(int a, int b)
{
    return (a > b ? a : b);
}
```

#define: Macros

- Blind substitution inside file

```
#define MALLOC mymalloc
#define MAXSIZE 100
p = MALLOC(MAXSIZE);
printf("Allocated %d bytes", MAXSIZE);
```

- is exactly the same as

```
p = mymalloc(100);
printf("Allocated %d bytes", 100);
```

#define: Parameterized Macros

```
#define PROD(a, b)      PROD2(a, b * 10)

PROD(5, 6)  => PROD2(5, 6 * 10)
PROD(5, 6 + 7) => PROD2(5, 6 + 7 * 10) BUG!!
```

42

#define: More usage
#undef

```
#define OLDFUNC(a, b)   NEWFUNC1(a); NEWFUNC2(b)
OLDFUNC(5,6)
becomes
NEWFUNC1(5); NEWFUNC2(6)

for (i = 0; i < 5; i++) OLDFUNC(5, 6);
becomes
for (i = 0; i < 5; i++) NEWFUNC1(5); NEWFUNC2(6);

// can also undefine
#undef OLDFUNC
```

43

operator

```
#define PRINT_INT(n)    printf(#n " = %d\n", n);
```

Preprocessor creates a string literal from PRINT_INT's argument.

```
PRINT_INT(i/j)
->
printf("i/j" " = %d\n", n);
->
printf("i/j = %d\n", n);
```

44

operator

```
#define MK_ID(n)  i##n
```

Preprocessor combines `i` and `n` together into one token.

```
int MK_ID(1);
int MK_ID(2);
->
int i1;
int i2;
```

45

operator

```
#define MK_ID(n)  i##n
```

Preprocessor combines `i` and `n` together into one token.

```
int MK_ID(1);
int MK_ID(2);
->
int i1;
int i2;
```

46

#ifdef, #ifndef, #if, #elif, #else

```
#ifdef identifer
Lines to be included if identifier is defined as a macro
#endif

#ifndef identifer
Lines to be included if identifier is not defined as a macro
#endif

#if expr1
Lines to be included if expr1 is nonzero
#elif expr2
Lines to be included if expr1 is zero and expr2 is nonzero
#else
Lines to be included otherwise
#endif
```

In-class Exercise 8.1

Write a macro DISP(f,x) that expands into a call of printf that displays the value of the function f when called with argument x. For example,

```
DISP(sqrt, 3.0)
```

should expand into

```
printf("sqrt(%f) = %f\n", 3.0, sqrt(3.0));
```

48