Lecture13: Other C Topics
12/17/2012

Topics

- Variable-length argument lists
- Pointers to functions
- Command-line arguments
- Suffixes for integer and floating-point constants
- More memory allocation: `calloc()` and `realloc()`
- Unconditional branching: `goto`
- Compiling multiple-source file with `Makefile`

Variable-length argument lists

- Have you wondered how these are implemented?

```
int printf(const char *format, …);
int scanf(const char * format, ... );
```

```
printf("%d", i, j);
printf("%d %f", i, j);
printf("%d %f %c", i, j, k);
```

- Pass **variable number** of arguments to a function
- Also **variable type** in the passed arguments

```c
#include <stdio.h>
#include <stdarg.h>

int sum(int n, ...)
{
    int total = 0;
    int i;
    va_list ap;
    va_start(ap, n);
    for (i=0; i<n; i++)
    {
        total += va_arg(ap, int);
    }
    va_end(ap);
    return total;
}

int main()
{
    printf("%d\n", sum(2, 1, 2));
    printf("%d\n", sum(3, 1, 2, 3));
}
```

- `<stdarg.h>`: include this header file.
- `va_list`: a type for getting variable arguments
- `va_start()`: initialize the variable `ap` that holds the variable arguments, and `n` is the last named argument in the function.
- `va_arg(ap, type)`: get the next argument of a specific type.
- `va_end()`: end using the va_list

In-Class Exercise 13.1

- Write a variable-length argument function, `minprintf()`, that supports the format string: "`%d`", "`%s`".

```
void miniprintf(char *format, …);

i = 10;
s = "hello";
minprintf("%d", i);          // will print 10
minprintf("%d:%s", i, s);  // will print 10:hello
```

Hint: the number of "`%x`" in the format string will tell you (1) the number of additional arguments, and also (2) the type of each argument.

Pointers to Functions

- A pointer to a function contains the address of the function in memory.

```
int x;
int *p;                 // pointer to an integer
p = &x;

int max(int a, int b);
int (*pf)(int, int);    // pointer to a function, this function
                        // has a prototype int f(int, int)
pf = max;
(*pf)(10, 20);          // call max(10, 20)
```

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

int foo(int do_min)
{
   int (*pf)(int, int); // declaring function ptr
   if (do_min)
   {
        pf = min;
   }
   else
   {
        pf = max;
   }
   return pf(10, 20);
}
```

qsort() example

```
void qsort(void * base, size_t num, size_t size,
           int ( * compar ) ( const void *, const void * ) );


/* qsort example */
#include <stdio.h>
#include <stdlib.h>
int values[] = { 40, 10, 100, 90, 20, 25 };

int compare (const void * a, const void * b)
{
   return ( *(int*)a - *(int*)b );
}

int main () {
   int n; qsort (values, 6, sizeof(int), compare);
   for (n=0; n<6; n++)
        printf ("%d ",values[n]);
   return 0;
}
```

In-Class Exercise 13.2

- Define the function `compare()` to sort blackjack hands from small to large.

```
#include <stdio.h>
#include <stdlib.h>
char *hands[] = { "AK", "T8", "45", "JQ", "8T", "39" };

int compare(const void * a, const void * b)
{
        // fill in your code
}

int main ()
{
  int n;
  qsort(hands, 6, sizeof(char *), compare);
  for (n=0; n<6; n++)
     printf ("%s ", hands[n]);  // print 45 39 8T T8 JQ AK
  return 0;
}
```

Command-line arguments

- The function `main()` can be defined with or without parameters, using the following forms:

```
int main();
int main(int argc, char *argv[]);
int main(int argc, char **argv);
```

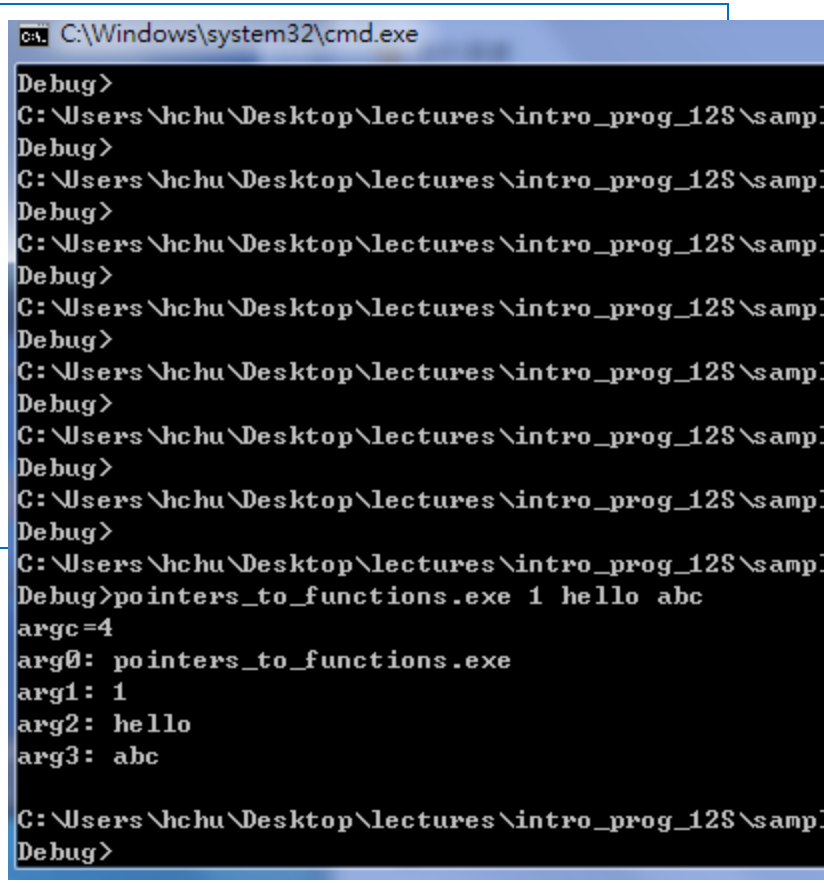
`argc` (argument count) is an integer that indicates how many arguments were entered on the command line.

`argv` (argument vector), is an array of pointers to arrays of character objects.

Command line arguments

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i;
    printf("argc=%d\n", argc);
    for (i=0; i<argc; i++)
    {
        printf("arg%d: %s\n", i, argv[i]);
    }
}
```



Suffixes for integer and floating-point constants

```
unsigned int ui = 174u;
long int li = 8358L;
unsigned long int uli = 28373ul;
long long int lli = 12345678912345678 9LL;

float f = 1.28f;
double ld = 3.14159L;
```

More memory allocation: `calloc()` and `realloc()`

```
void *malloc(size_t size);

void *calloc(size_t nmemb, size_t size);
```

Initialize the elements of the array to zero.

```
void *realloc(void *ptr, size_t size);
```

Change the size of the memory allocation (pointed to by *ptr).

`goto` : Unconditional branching (avoid using this)

```
#include <stdio.h>

int main()
{
    int count = 1;

  start:      // goto label
    if (count > 10)
        goto end;
    printf("%d ", count);
    count++;
    goto start;

  end:        // goto label
    putchar('\n');

    return 0;
}
```

In-Class Exercise 13.3

- In the following program that calculates the Fibonacci number, replace the `for` loop with `goto`.

```
void main()
{
    int n, first = 0, second = 1, next=0, i;
    printf("Enter n: ");
    scanf("%d",&n);
    for (i = 1; i < n; i++)
    {
         next = first + second;
         first = second;
         second = next;
    }
    printf("%d\n",next);
    return 0;
}
```

Custom Makefile in Codeblocks

- 1. New a empty project
- 2. Add two source file (ex:test.c, test2.c)
- Main() is in one of the file, and the implementation of function is in the another file(check the next slide)
- 3. Download the makefile in the course site,

and put it in your project directory.

Custom Makefile in Codeblocks

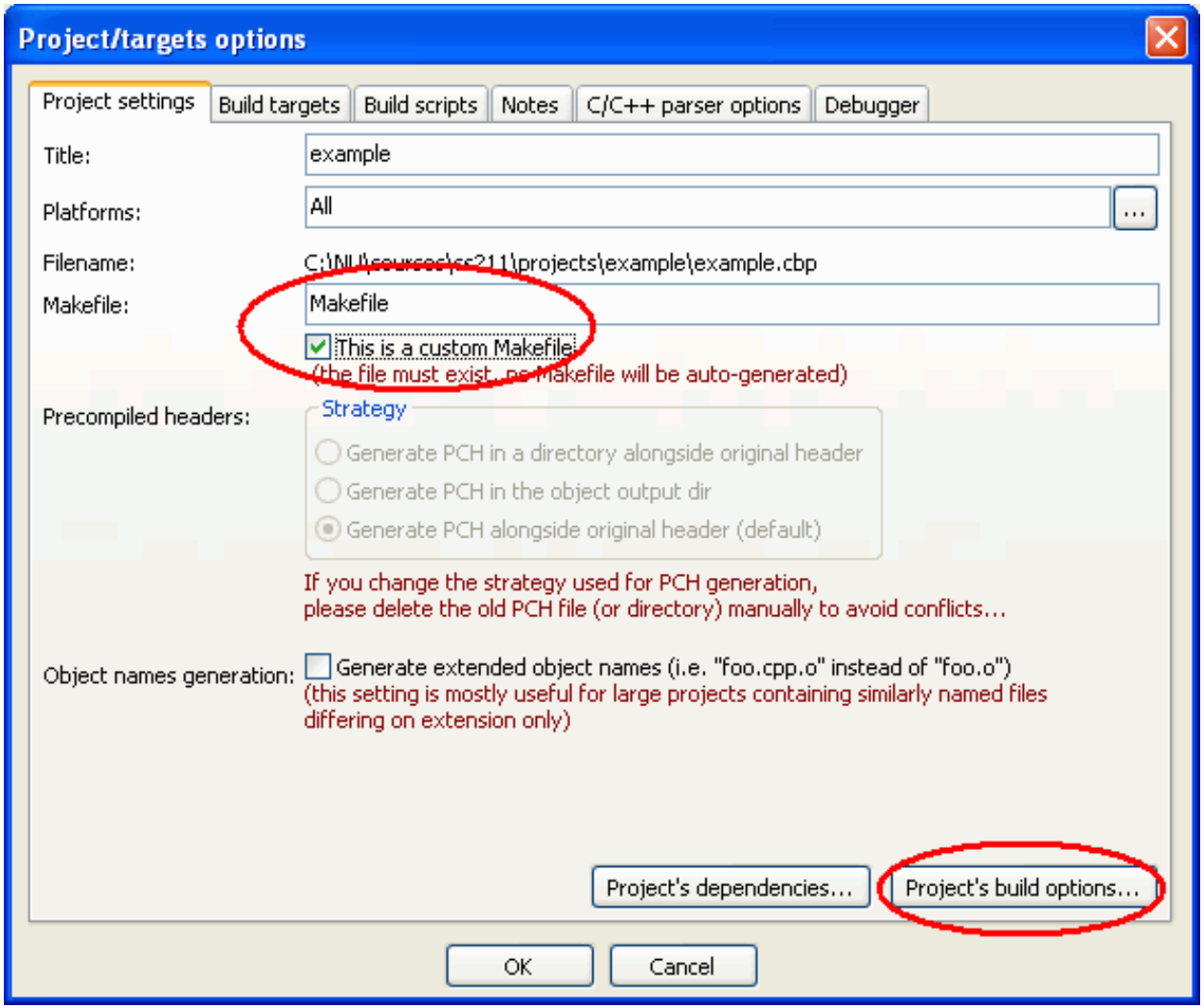
- **test.c:**

```
#include <stdio.h>
void func();
int main() {
printf("main.c\n");
func();
return 0;
}
```

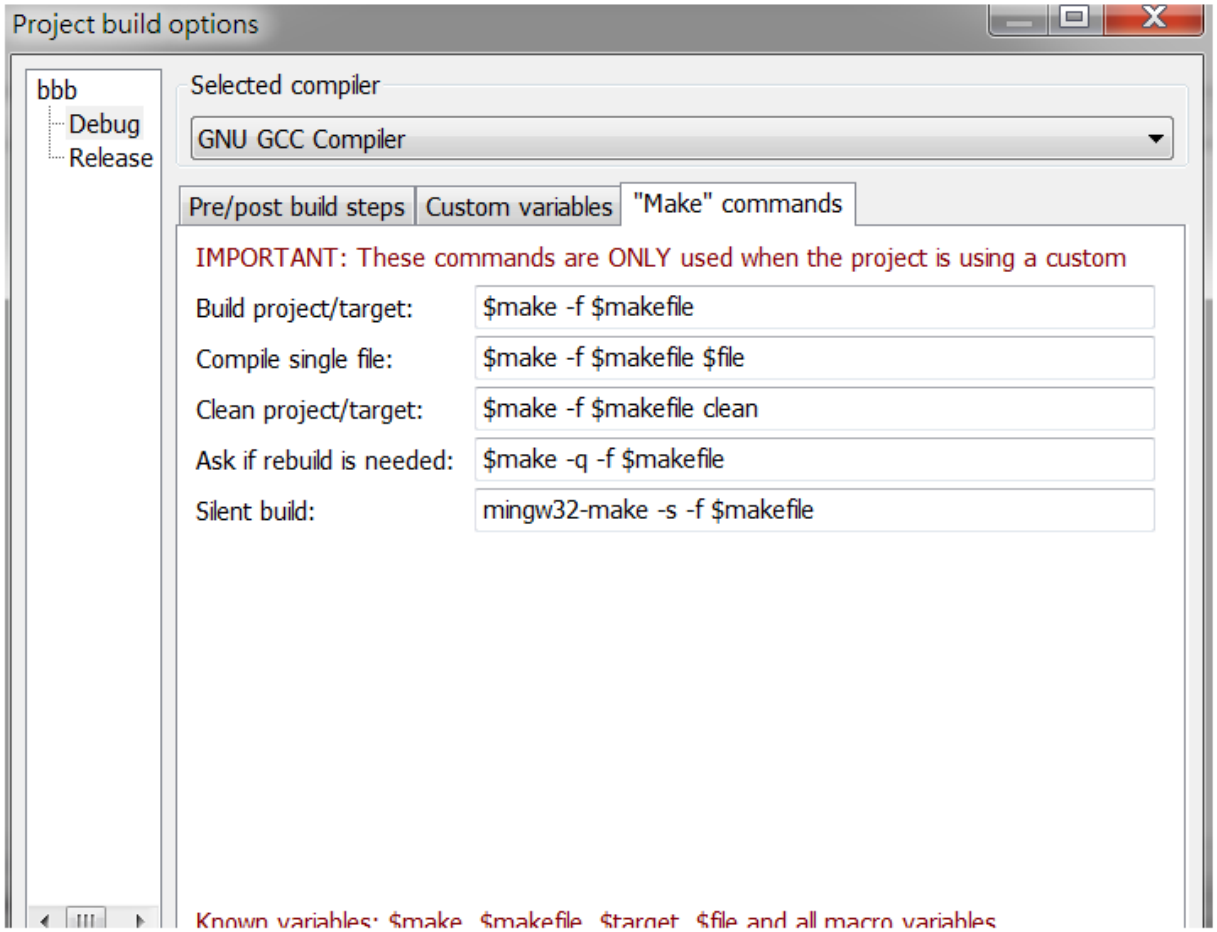
test2.c:

```
#include <stdio.h>
void func() {
printf(“I am implementation”);
}
```

Custom Makefile in Codeblocks



Custom Makefile in Codeblocks



```
4
5  # Project name and version
6  Proj := project_name
7  Version := Debug
8
9  #paths for Project (Ppath) Object files (Opath) and binary path (Bpath)
10 Ppath := project_path (Z:\\??)
11 Opath := $(Ppath)\obj\$(Version)
12 Bpath := $(Ppath)\bin\$(Version)
13
14 flags = -pipe -mthreads -D__GNUWIN32__ -Wall -g
15
16 CXX = "C:\Program Files (x86)\CodeBlocks\MinGW\bin\mingw32-gcc.exe" (gcc's path)
17
18 Obj := test.o test2.o
19
20 test.exe : $(Obj)
21         $(CXX) test.o test2.o -o test.exe
22
23 test2.o : test2.c
24         $(CXX) -c $(flags) $(Ppath)\test2.c
25 test.o : test.c
26         $(CXX) -c $(flags) $(Ppath)\test.c
27
28 .PHONEY : clean
29
30 clean:
31         del $(Bpath)\$(Proj).exe $(Obj)
```