Routine specification in the C programming language

Concepts

Source code, compilation, separate compilation, object code, link edition, executable file, loading, execution, reference solving.

Exercise

Practice with the ***gcc*** compiler through the provided examples.

Steps

Solve the proposed exercises, practicing with ***gcc***.

Documentation

- Unix notes.
- C programming language notes.
- ***gcc*** compiler notes.
- Source code in C of the provided programs.
- Unix help (*man*).

The *gcc* compiler

Compilation steps (step by step or in a single step)

- Preprocessing: preprocessor instructions (#define…) are interpreted, applying the required replacements in the source code. Example:

```
more example.c
```

```
#define SIZE 50
main(int argc, char *argv[]) {
   int table[TAMAINA];
   int i;

   for (i = 0; i < SIZE; i++) {
      table[i] = i*i;
      printf("the square of %d is %d\n", i, table[i]);
   }
}
```

```
gcc –E example.c > example.pp
more example.pp
```

```
main(int argc, char *argv[]) {
   int table[50];
   int i;

   for (i = 0; i < 50; i++) {
      table[i] = i*i;
      printf("the square of %d is %d\n", i, table[i]);
   }
}
```

- Compilation: the C source code is translated into the machine/processor assembler code.
- Assembling: the assembler code is translated into object code (binary code, which is executable by the processor).
- Link edition: the different object modules that form the program (compiled source files, used libraries …) are linked, creating a single executable file.

gcc compiler syntax

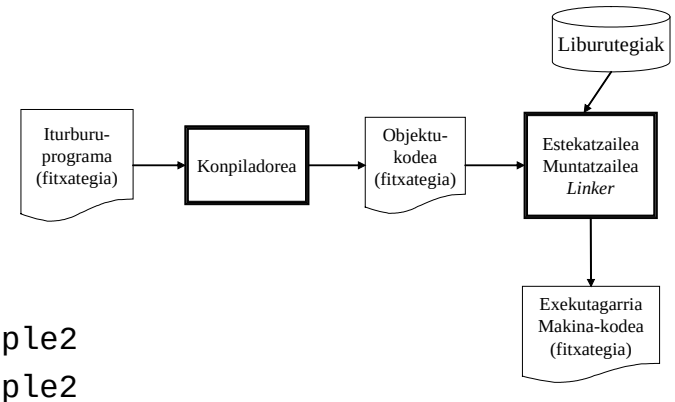
gcc options C_file_name obj_module_name

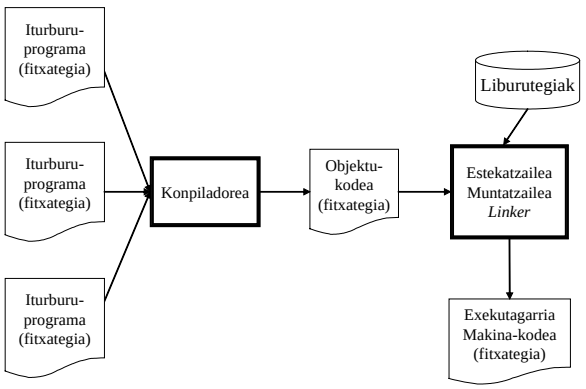
Options:

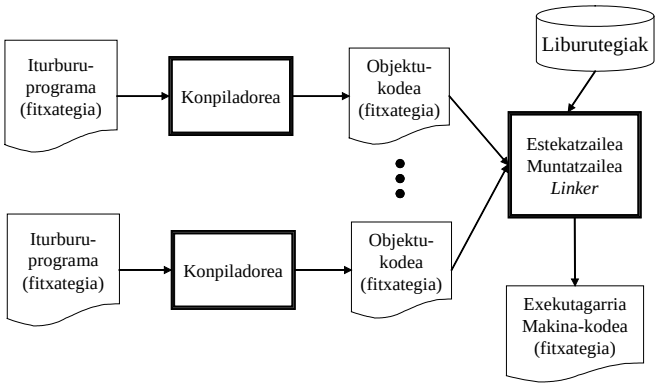
-c	creates object file only (without linking)
-g	generates debugger information
-llib_name	tells the linker to search object modules in the lib_name library
-Ldirectory	indicates the directory where libraries are located
-o exec_name	gives a name to the executable file
-E	only for preprocessing
-S	creates assembler file only
-D	symbol definition (conditional compilation, macros)

Examples

```
gcc example.c (result: a.out)
gcc –c example.c
gcc example.o –o example
gcc example.c –o example
gcc –c example2.c
gcc –c routines.c
gcc example2.o routines.o –o example2
gcc example2.c routines.c –o example2
```







ld tool (*link-editor*)

Links object modules

```
ld   options   obj_module_names
```

Options:

-l`library`	library for searching for object modules
-L`directory`	directory where libraries are located
-o `exec_name`	giving a name to the executable file
-M	gets a map of modules

ar tool

Creating and managing libraries of object modules.

```
ar   options   library   obj_module_names
```

Options:

d	deleting a module
m	moving a module
p	printing a module
r	replacing a module
t	listing a table of contents
x	extracting a module
c	creating a library
u	replacing only updated modules
v	getting information about changes

Libraries

(1) Naming convention: `lib`*`name`*`.a`
Example: `libmod.a`

(2) Creation: `ar rcv lib`*`name`*`.a`
Example: `ar rcv libmod.a`

(3) Adding an object module: `ar r lib`*`name`*`.a `*`module`*`.o`
Example: `ar r libmod.a replace.o`

(4) Compiling using a library:

`gcc -L`*`dir`* `-o `*`executable source`* `-l`*`name`*
Example: `gcc -L. -o test test.c -l`**`mod`**
object modules are searched in the *libmod.a* library

(5) Same thing using the linker:

`ld -L`*`dir`* `-l`*`name`* `-o `*`executable object`*`.o`
Example: `ld -L. -l`**`mod`** `-o test test.o`

gdb tool (debugging)

Syntax: **`gdb`** `executable`

Commands:

help	command summary
run	execute program
next	execute one instruction
step	enter into a function
list	see code
break	define a breakpoint
print	show the value of a variable (once)
display	show the value of a variable on each step/breakpoint
where	show the execution point and the content of the stack

Exercises

1. A set of C functions for managing tables of integers is provided (*tables.c*). Using those functions, write a C program for sorting integer numbers, then compile it and finally test it.

2. A set of C functions for managing queues of integers is provided (*queues.c*). An object module of this set of functions has to be created. Two additional files are provided (*queues.h* and *assign.h*) for defining functions and data types and for memory management, respectively. Finally, a program is provided (*mytest.c*). This program reads integer values, inserts them in a queue and prints them in the standard output. Based on this program, write a new program that prints the numbers sorted.

Steps

1. Write the sortnumbers.c program, including tables.h (#include statement).
2. Compile the sortnumbers.c file, creating the sortnumbers executable. Test it.

```
/* tables.h */

extern void read_table(int table[], int count);
extern void print_table(int table[], int count);
extern void sort_table(int table[], int count);
```

```
/* tables.c */

#include <stdio.h>

void read_table(int table[], int count)
{
   int i;

   for (i = 0; i < count; i++)
      scanf("%d", &table[i]);
}

void print_table(int table[], int count)
{
   int i;

   for (i = 0; i < count; i++)
      printf("%d\n", table[i]);
}
```

```
int smallest(int table[], int count)
{
   int i, index_smallest = 0;

   for (i = 1; i < count; i++)
      if (table[i] < table[index_smallest]) index_smallest = i;
   return(index_smallest);
}

void sort_table(int table[], int count)
{
   int i, index;
   int tmp;

   for (i = 0; i < count-1; i++) {
      index = smallest(&table[i], count-i);
      tmp = table[i];
      table[i] = table[i+index];
      table[i+index] = tmp;
   }
}
```

3. Compile the queues.c file, creating the queues.o file.
4. Create the program mytest. Test it.
5. Modify mytest such that it sorts the numbers.

```
/* queues.h */

struct item {
   struct item *next;
   int number;
   char *info;
};

struct queue {
   struct item *first;
   struct item *last;
};

#define NIL (struct item*)0
#define BOOL short

void create(struct queue *p);
BOOL empty(struct queue *p);
struct item *first(struct queue *p);
struct item *take(struct queue *p, int number);
void add(struct queue *p, struct item *pelem);
void insert(struct queue *p, struct item *pelem);
```

```
/* assign.h */

#define MAXELEM 100

struct item table_elem[MAXELEM];
```

```
struct item *get_elem()
{
   static int i = 0;

   if (i < MAXELEM) return(&table_elem[i++]);
   else return(NIL);
}
```

```
/* queues.c */

#include "queues.h"

void create(struct queue *p)
{
   p->first = NIL;
   p->last = NIL;
}

BOOL empty(struct queue *p)
{
   return(p->first == NIL);
}

struct item *first(struct queue *p)
{
   struct item *paux;

   paux = p->first;
   if (paux != NIL) {
      p->first = paux->next;
      if (p->last == paux)
         p->last = NIL;
   }
   return(paux);
}

void add(struct queue *p, struct item *pelem)
{
   struct item *paux;

   paux = p->last;
   if (paux == NIL)
      p->first = pelem;
   else
      paux->next = pelem;
   pelem->next = NIL;
   p->last = pelem;
}

struct item *take(struct queue *p, int number)
{
   struct item *paux, *paur;

   paur = NIL;

   for (paux = p->first; paux != NIL; paux = paux->next)
   {
```

```
      if (paux->number > number) paur = paux;
      if (paux->number == number)
      {
         if (paur != NIL) paur->next = paux->next;
         else p->first = paux->next;
         if (paux->next == NIL) p->last = paur;
         return(paux);
      }
      if(paux->number < number) return(NIL);
   }
   return(NIL);
}

void insert(struct queue *p, struct item *pelem)
{
   struct item *paux, *paur;

   paur = NIL;

   for (paux = p->first; paux != NIL; paux = paux->next)
   {
      if (pelem->number <= paux->number) paur = paux;
      else break;
   }
   pelem->next = paux;
   if (paur != NIL) paur->next = pelem;
   else p->first = pelem;
   if (paux == NIL) p->last = pelem;
}
```

```
/* mytest.c */

#include <stdio.h>
#include "queues.h"
#include "assign.h"

struct queue my_queue;

main()
{
   int data;
   struct item *elem;

   create(&my_queue);
   while (scanf("%d", &data) != EOF)
   {
      if ((elem = get_elem()) == NIL) break;
      elem->number = data;
      add(&my_queue, elem);
   }
   printf("\n");
   while (!empty(&my_queue))
   {
      elem = first(&my_queue);
      printf("%d\n", elem->number);
   }
}
```

NAME

sortnumbers – Unix filter for sorting integer numbers.

SYNOPSIS

sortnumbers

DESCRIPTION

The sortnumbers Unix filter reads a sequence of integers form the standard input (one number per line), and writes the sequence, ordered from the smallest to the biggest, in the standard output

The input sequence terminates when the EOF condition is satisfied (^D in the keyboard)

The sortnumbers filter has no parameters.

COMPATIBILITY

The sortnumbers filter should work on any Unix system.

KNOWN ERRORS

The sortnumbers filter sorts a maximum of 100 numbers. If more than 100 numbers are entered as input, only the first 100 numbers will be sorted.

The numbers must be of int C type. Otherwise, the result is not guaranteed.

AUTHOR

Anonymous

COPYRIGHT

Copyleft license <http://www.gnu.org/copyleft/>. No guarantee.

SEE ALSO

sort (1)