GAA Manual

GAA Argument Analyzer

Contents

Chapter 1

Tutorial

This tutorial is based on an example : an imaginary programme named 'sample' . GAA includes instructions that are not described in this tutorial. Please read the GAA Quick reference !

1.1 GAA's input and output

When you call GAA, you give him a file written in GAA language which describes the arguments of your programme. GAA creates a C file that you'll have to add to your Makefile and a header file that you'll need to include in your C/C++ source.

Typical GAA call: **gaa filename.gaa**
two files are created by GAA: *filename_gaa.h* and *filename_gaa.c*

1.2 GAA's principle

GAA provides two functions:

1. **int gaa(int argc, char *argv[ ], gaa info *gaaval)** calls the argument analyser
   - *argc* is the number of arguments of your programme
   - *argv* is the argument list
   - *gaaval* is a pointer to a *gaainfo* instance. It will be filled by the analyser. The gaainfo structure is defined in the GAA language file. The structure pointed by gaaval is the result of the analysis.
   - *return value*: -1 if success
2. **void gaa_help()** shows the help of your programme. This help is generated by GAA, according to your instructions

1.3 How to declare the members of the gaainfo structure

Anywhere, between two instructions, you write after a '#' a structure-member declaration exactly like in C:
example:

```
#int number;
```

You can add to your gaa file as much declarations as you like.

1.4 How to declare options

Option declaration general syntax:
for argless options:

```
option (short_name, long_name) { action } "option help"
```

for options with one argument

```
option (short_name, long_name) ARG_TYPE "arg help" { action } "option help"
```

Note: short_name must be a single character
When gaa() finds an argument begining with '-', it considers that this argument is an option. If an option with the same name has been declared, there are two possibilities:

1. the option doesn't need any argument gaa() executes the assiociated *action*. In this action, you can refer to a data contained in gaaval : you only have to put a '$' character before the name of the variable. example : if you have declared '#char pom', to refer to 'pom' in an action, you must write '$pom'; otherwise, if you write 'pom', it will be considered as a general variable named 'pom'.

2. the option requires an argument gaa() checks if the next argument has the right type, and calculates its value. Then, it executes the action associated with the option in the same way as if there was no argument. In the action, '$1' represents the value of the argument.

Note:

- predefined ARG_TYPEs exist (cf. Quick Reference)
- the type of the argument value depends on the choosen ARG_TYPE. For instance, type for 'INT' is 'int'

For most programmes, the user must provide a filename (for instance) without an option. This is supported by GAA with the 'rest' instruction :

```
rest ARG_TYPE { action }
```

1.5 Description of our sample progamme

Our sample does nothing: he only analyses his arguments and shows the selected options and parameters.
Typical call of the sample : **sample file**
Sample's options:

- **-v** or –**verbose** : activate verbose mode (*default value: off*)
- **-n** or –**num integer** : specifies an integer as parameter of the programme (*default value : 0*)
- **-f** or –**file file1 file2 ... filex** : specifies a list of files
- **-h** or –**help** : shows the help of the sample

1.6 The '*sample.gaa*' file

GAA generates the help of your programme. You can make him write something in the help with the instruction *helpnode*. So, we begin sample.gaa by the line:

```
helpnode "SAMPLE help\nUsage : sample [options] file_name"
```

When GAA generates the help, it follows the order in the gaa file. So, this line will be the first of the help.
Let's declare the sample's options

- Verbose

First, we should declare the gaainfo member that will store the state of the option, 'verbose'.

```
#int verbose;
```

then we must declare the option itself

```
option (v, verbose) { $verbose = 1 } "verbose mode on"
```

GAA changes '$verbose' into the 'verbose' member of gaaval. When GAA returns, gaaval-¿verbose will be equal to 1.

- Num

First, we should declare the gaainfo member that will store the number specified by the user. Let's name it 'n'.

```
#int n;
```

then, the option:

```
option (n, num) INT "integer" { $n = $1 } "specifies the number of totoros"
```

- File

This option needs a list of filenames as argument. So we need two datas : the number of files and a list of filenames. For that, we have the STR predefined type whose C-type is char*. To have a list of STR, the ARG_TYPE must be *STR. The C-type of *STR is of course char**

```
#int size;
#char **input;
option (f, file) *STR "file1 file2...fileN" { $input = $1; $size = @1 } "specifies"
" the output files"
```

'@1' is transformed by GAA into the number of filename given by the user. Note : it's always '@1'

- Help

This option nust show the help and quit. So, let's call gaa_help()

```
option (h, help) { gaa_help(); exit(0); } "shows this help text"
```

Sample must be called with a filename as argument. So we must use the 'rest' instruction, and create a data that will store the filename.

```
#char *file;
rest STR { $file = $1 }
```

'rest' represents the argument(s) which remain when you remove all the options (with their private argument(s)) from the line given by the user.

For example, in the command 'tar -xv zorglub.gif -f toto.tar kiem.jpg bobby.c', the arguments managed by rest are zorglub.gif, kiem.jpg and bobby.c. In this case, if the GAA file specifies 'rest STR ...', an error will occur, because the program wants **one** rest argument only. If the GAA files specifies 'rest *STR', it's OK, because it means that the program needs a list of arguments as rest. Each *gaaval* data must be initialized. To do that we use the 'init' instruction

```
init { $n = 0; $verbose = 0; $file = NULL; $size = 0 }
```

That's all !

1.6.1 sample.gaa

Finally, here is the text of **sample.gaa :**

```
helpnode "SAMPLE help\nUsage : sample [options] file_name"

#int verbose;
option (v, verbose) { $verbose = 1 } "verbose mode on"

#int n;
option (n, num) INT "integer" { $n = $1 } "specifies the number of"
 "totoros"

#int size;
#char **input;
option (f, file) *STR "file1 file2...fileN" { $input = $1; $size = @1 } "specifies"
 " the output files"

option (h, help) { gaa_help(); exit(0); } "shows this help text"

#char *file;
rest STR { $file = $1 }

init { $n = 0; $verbose = 0; $file = NULL; $size = 0 }
```

1.6.2 The 'smain.c' the C program using GAA

```
#include <stdio.h>
#include "gaa.h"

int main(int argc, char **argv)
{
gaainfo info;
int i, v;

 if((v = gaa(argc, argv, info)) != -1)
 // calls GAA. The user gived bag args if it returns -1
 {
    return 0;
 }

 printf("n : %d\nfile : %s\nverbose : %d\n", info.n,
  info.file, info.verbose); // shows the given arguments

 if(info.size > 0)
 for(i = 0; i < info.size; i++)
   printf("%s\n", info.input[i]);

 return 0;
}
```

1.7 How to compile the sample file

Calling GAA:

```
$ gaa sample.gaa
```

Calling GCC:

```
$ gcc sample_gaa.c smain.c -o sample
```

Please read the GAA Reference
In the reference, you will find how to make an option obligatory, or two (or more) options incompatible. You will learn how to define your own argument types...

Chapter 2

Reference

2.1 GAA functions

These functions are declared in the header file generated by GAA

int gaa(int argc, char *argv[], gaainfo *gaaval); gaa() analyses arguments from the command line

int gaa_file(char *name, gaainfo *gaaval); gaa_file() analyses options from a configuration file

void gaa_help(); gaa_help() prints the help of the program

2.2 GAA file structure

- The GAA-ng files consist of two parts:
  1. (optional) C declarations that will be put at the begining of the output. These declarations must be enclosed between '#{' and '#}' brackets.
  2. GAA declarations C file
- The traditional GAA file has the following format:
  1. GAA declarations
  2. (optional) C declarations that will be put at the begining of the output C file. This part is separated by the symbol '##' .

2.3 GAA declarations

- **DATA data-name AS gaa-arg-type** Data declaration for the gaainfo structure. Example :DATA a AS CHAR → 'a' can contain a character DATA textList AS *STR → 'textList' can contain a list of strings (C type : char**)

- **#C-variable-declaration** Data declaration for the gaainfo structure. Example : *#char a;*
- **defitem ARGTYPE function-name #C-ArgType** Declaration of an user-defined named *ARGTYPE*. The *function-name* is the name of the function which will be called by gaa() to check if the argument is an *ARGTYPE* and to get its value. C-ArgType is the C type of *ARGTYPE*. The *function-name* function must declared (in the C-Declaration part) as follows : *C-ArgType function-name*(char*) { } In this function, you must use the **GAAERROR** macro when the argument is incorrect. Example : defitem HEX findhex #int ## int findhex(char*) {...}
- **helpnode "help text"** Writes a line in the help.
- **incomp shortnamelist** *shortnamelist* is a list of options. This instruction makes the options in the list incompatibles. Example : incomp vn → the option verbose and num (of the sample) become incompatible
- **init { action }** Defines the intialization action. This action will be the first executed.
- **obligat shortnamelist** *shortnamelist* is a list of options. This instruction forces the user to give one option in of the list. Example : obligat vn → the user must give the option verbose or/and the option num
- **option (shortname, longname) <ARG_DECL_LIST> { action } "option help"** An ARG_DECL_LIST is a list of argument declarations (surprising, isn't it ?) ARG_DECL syntax : ARGTYPE or ARGTYPE "argument description" ***Note*** : shortname must be a single character When gaa() finds an argument begining with '-', it considers that this argument is an option. If an option with the same name has been declared, there are two possibilities :

  1. the option doesn't need any argument
  2. gaa() executes the assiociated *action*. In this action, you can refer to a data contained in gaaval : you only have to put a '$' character before the name of the variable. *example* : if you have declared 'DATA pom AS CHAR' '#char pom', to refer to 'pom' in an action, you must write '$pom'; otherwise, if you write 'pom', it will be considered as a general variable named 'pom'.
  3. the option requires one or more argument(s) gaa() checks if the next argument has the right type, and calculates its value. Then, it executes the action associated with the option in the same way as if there was no argument. In the action, '$n' represents the value of the nth argument.

  bf Note: predefined ARGTYPEs exist (cf. Quick Reference) the type of the argument value depends on the choosen ARGTYPE. For instance, type for 'INT' is 'int' **rest <ARG_DECL_LIST> { action }** For most programs, the user must provide a filename (for instance) without an option. This is supported by GAA with the 'rest' instruction. 'rest' represents the argument(s) which remain when you remove all the options (with their private argument(s)) from the line given by the user. For example, in the command 'tar -xv zorglub.gif -f toto.tar kiem.jpg bobby.c', the arguments managed by rest are zorglub.gif, kiem.jpg and bobby.c. In this case, if the GAA file specifies 'rest STR ... ', an error will occur, because the program wants one rest argument only. If the GAA files specifies 'rest *STR', it's OK, because it means that the program needs a list of arguments as rest.

Example : rest *STR ... → the rest is a list of name Predefined ARG_TYPEs **ARG_TYPEC-typeRecognized arguments**STRchar*any argumentINTintintegersCHARcharsingle charactersFLOATfloatfloat number

bf Important: if in a declaration, you declare *ARG_TYPE, the user will have to give a list of ARG_TYPE and not a single ARG_TYPE. In the attached action, you can get the number of found arguments with the '@n' symbol.

2.4 Configuration files

GAA can analyse the content of a configuration file. Syntax of a configuration file :

option-long-name arg1 arg2 arg3 ... option-long-name arg1 arg2 arg3