C Tips 'N' Tricks

Aula 2

Bruno Jurkovski Kauê Silveira

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Grupo PET Computação

26 de Maio de 2010

Definições

Definições

... no protótipo da função

va_list lista com os argumentos

Inicialização

va_start(va_list params, var);

- Inicializa params, a partir de var

Destruição

va_end(va_list params);

- Destrói params

Funções com número variável de parâmetros (1)

```
1   #include <stdio.h>
2   #include <stdarg.h>
3
4   double media(const int qtdNros, ...)
5   {
6       int i;
7       double result=0;
8       va_list params;
9       va_start(params, qtdNros);
10      for(i=0; i<qtdNros; i++)
11          result += va_arg(params, int);
12      va_end(params);
13      return result/qtdNros;
14  }
```

Funções com número variável de parâmetros (2)

⋮

```
16      int main()
17      {
18          printf("%lf\n", media(4, 1, 2, 3, 4));
19          return 0;
20      }
```

Exercício

Modifique o exemplo anterior para receber uma string com os tipos dos valores ('d' para **double** e 'i' para **int**). Por exemplo, a string "diddi" significa que os parâmetros são, respectivamente, **double**, **int**, **double**, **double**, **int**.
Protótipo da função: **double** media(**const char** *format, ...);

vprintf, vfprintf, vsprintf

vprintf, vfprintf, vsprintf - <stdio.h>

```
int vprintf ( char *format, va_list arg_ptr );
int vfprintf ( FILE *stream, const char *format, va_list arg_ptr );
int vsprintf ( char *buffer , char *format, va_list arg_ptr );
```

- Semelhantes à printf(), fprintf() e sprintf(), respectivamente, mas devem ser encapsuladas por uma outra função que recebe um número variável de parâmetros.

vscanf, vfscanf, vsscanf

vscanf, vfscanf, vsscanf - <stdio.h>

```
int vscanf( char *format, va_list arg_ptr );
int vfscanf( FILE *stream, const char *format, va_list arg_ptr );
int vsscanf( char *buffer, char *format, va_list arg_ptr );
```

- Semelhantes à scanf(), fscanf() e sscanf(), respectivamente, mas devem ser encapsuladas por uma outra função que recebe um número variável de parâmetros.

Exemplo (1)

```
1   #include <stdio.h>
2   int main()
3   {
4       int i;
5       char str[100];
6       for(i = 0; i < 5; i++)
7           sprintf(str, "file%i.txt", i),
8           fclose(fopen(str, "w"));
9       return 0;
10  }
```

Exemplo (2)

```
1   #include <stdio.h>
2   #include <stdarg.h>
3
4   FILE* sfopen(const char* filename,
5                const char* mode, ...)
6   {
7       char fname[FILENAME_MAX];
8       va_list params;
9       va_start(params, mode);
10      vsprintf(fname, filename, params);
11      va_end(params);
12      return fopen(fname, mode);
13  }
```

Exemplo (2)

```
                                    ⋮

15      int main()
16      {
17          int i;
18          for(i=0; i<5; i++)
19                  fclose(sfopen("file%d.txt", "w", i));
20          return 0;
21      }
```

Macros

Definidas pelo pré-processador

`__FILE__`	nome do arquivo atual
`__LINE__`	número da linha atual

Operações

`#`	transforma em string
`##`	concatena

Definida pelo compilador (não é macro)

`__FUNCTION__`	nome da função atual

Exemplo (1)

```c
#include <stdio.h>

#define DEBUG                                   \
  printf("%s:%s:%i\n", __FILE__, __FUNCTION__, __LINE__)

int main()
{
    DEBUG;
    //printf("%s:%s:%i\n", "main.c", "main", 8);
    return 0;
}
```

Exemplo (2)

```
1  #include <stdio.h>
2
3  #define DEBUGintVAR(var)        \
4      printf("%i: %s = %i\n", __LINE__, #var, var)
5
6  int main()
7  {
8      int i = 77;
9      DEBUGintVAR(i);
10     //printf("%i: %s = %i\n", 9, "i", i);
11     return 0;
12 }
```

Exemplo (3)

```c
#include <stdio.h>

#define DEBUGfloatVAR(var)      \
    printf("%i: %s = %f\n", __LINE__, #var, var)

int main()
{
    float f = 0.67;
    DEBUGfloatVAR(f);
    //printf("%i: %s = %f\n", 10, "f", f);
    return 0;
}
```

Exemplo (4)

```
1  #include <stdio.h>
2
3  #define DEBUGMSG(msg)                \
4          printf("%i: %s", __LINE__, msg)
5
6  int main()
7  {
8          DEBUGMSG("batata");
9          //printf("%i: %s", 9, "batata");
10         return 0;
11 }
```

Exemplo (5)

```
1  #define DEBUGVAR(tipo, var)    \
2      DEBUG##tipo##VAR(var)
3
4  int main()
5  {
6      DEBUGVAR(int, i);
7      //DEBUGintVAR(i);
8
9      DEBUGVAR(float, f);
10     //DEBUGfloatVAR(f);
11
12     return 0;
13 }
```

Exercício

Faça um programa que imprima o próprio código. O programa deve ser composto por um único arquivo.

Estrutura Geral

```
1  # (isto eh um comentario) Importante:
2  # o espaco antes de 'gcc' em cada linha deve ser um TAB
3  all: executavel
4
5  executavel: main.c fonte1.o fonte2.o
6      gcc main.c fonte1.o fonte2.o -o executavel -Wall
7
8  fonte1.o: fonte1.c fonte1.h
9      gcc fonte1.c -c -Wall
10
11 fonte2.o: fonte2.c fonte2.h
12     gcc fonte2.c -c -Wall
```

Definição de Variáveis

```
1 CC = gcc
2 FLAGS = -Wall
3
4 exec: main.c fonte.c fonte.h
5     $(CC) main.c fonte.c -o exec $(FLAGS)
```

Variáveis Especiais

$@ nome da regra

$< primeira dependência da regra

$^ todas as dependências da regra

Exemplo Completo

```
1  CC = gcc
2  FLAGS = -Wall
3  OBJS = fonte1.o fonte2.o
4
5  all: executavel
6
7  executavel: main.c $(OBJS)
8      $(CC) $^ -o $@ $(FLAGS) #todas as dependências
9
10 fonte1.o: fonte1.c fonte1.h
11     $(CC) $< -c $(FLAGS) #primeira dependência
```

make -B

make -B

Força o make a recompilar tudo.

Regras Implícitas (1)

Regras

```
f.o: f.c [param_opt]*
    $(CC) $(CFLAGS) $(CPPFLAGS) -c -o f.o $^
f: f.o [param_opt]*
    $(CC) $(LDFLAGS) $^ $(LOADLIBES) $(LDLIBS) -o f
```

Regras Implícitas (2)

Variáveis

CC	compilador c
CPPFLAGS	flags para o pré-processador
CFLAGS	flags para o compilador c
LDFLAGS	flags para o ligador
LOADLIBES	bibliotecas
LDLIBS	bibliotecas

Exemplo

```
1 PROG = program
2 OBJS = main.o fonte1.o fonte2.o
3
4 CFLAGS = -Wall
5
6 $(PROG): main
7     mv main $(PROG)
8
9 main: $(OBJS)
```

Case Ranges

```
1   const char *get_char_type(char c)
2   {
3       switch(c) {
4           case '0' ... '9':
5               return "algarismo";
6           case 'a' ... 'z':
7               return "minusculo";
8           case 'A' ... 'Z':
9               return "maiusculo";
10          default:
11              return "desconhecido";
12      }
13  }
```

Function Attributes

Function Attibutes

- constructor, constructor(priority)
- destructor, destructor(priority)
- warn_unused_result
- deprecated

```
1  void before_main() __attribute__((constructor));
```

Variable Attributes

Variable Attributes

- cleanup(cleanup_function)
- deprecated
- unused

```
1  void myfclose(FILE **f) { fclose(*f); }
2
3  void write_results()
4  {
5      FILE *f __attribute__((cleanup(myfclose)));
6      f = fopen(//...
7  }
```

Type Attributes

Type Attributes

- deprecated

```
1  typedef struct
2  {
3      int x, y;
4  } Point __attribute__((deprecated));
5
6  int main() {
7      Point p; // warning aqui
8  }
```

Diagnostic Pragmas

```
1    #pragma message "Compiling " __FILE__ "..."
```

Binary Constants

```
1    i = 42;  // decimal: 4 * 10 + 2
2    i = 0x2a; // hexadecimal: 2 * 16 + 10
3    i = 052;  // octal: 5 * 8 + 2
4    i = 0b101010; // binário: 1 * 32 + 1 * 8 + 1 * 2
```

Designated Initializers (1)

```
1   /* [index] = value */
2   int a[6] = { [4] = 29, [2] = 15 },
3         // = { 0, 0, 15, 0, 29, 0 };
4       b[6] = { [1] = v1, v2, [4] = v4 },
5         // = { 0, v1, v2, 0, v4, 0 };
6       c[6] = { [1 ... 4] = 1 },
7         // = { 0, 1, 1, 1, 1, 0 }
8   /* also works with chars */
9       isWhitespace[256] = {
10        [' '] = 1, ['\t'] = 1, ['\n'] = 1
11      };
```

Designated Initializers (2)

```
1  typedef struct { int x, y; } Point;
2
3  /* .fieldname = value */
4  Point p = { .y = yvalue, .x = xvalue };
5
6  /* [index].fieldname = value */
7  Point ptarray[10] = {
8      [2].y = yv2, [2].x = xv2, [0].x = xv0
9  };
```

Passando matrizes por parâmetro

```
1  #include <stdio.h>
2
3  void imprime(int m, int n, int mat[m][n]) {
4      int i, j;
5      for(i = 0; i < m; i++) {
6          for(j = 0; j < n; j++)
7              printf("%d ", mat[i][j]);
8          printf("\n");
9      }
10 }
```

Dúvidas?

Contato:

- bruno.jurkovski@inf.ufrgs.br
- kaue.silveira@inf.ufrgs.br
- Sala 202 do prédio 43424

Boa semana acadêmica!

Referências


C++ : Reference : C Library. http://www.cplusplus.com/reference/clibrary/.


DRAFT of the C99 standard with corrigenda TC1, TC2, and TC3 included. http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf.


GNU Make Manual. http://www.gnu.org/software/make/manual/.


GCC online documentation. http://gcc.gnu.org/onlinedocs/.