

VISUAL PROGRAMMER

Contextualização

1. A acessibilidade nas escolas regulares é um marco recente para estudantes surdos;
2. Ainda existe uma falta de preparo nas escolas;

```
printf ("/begline"); break; case endlne: printf ("/endlne"); break; case on_failure_ -  
extract_number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt);  
case on_failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf  
failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-  
_number_and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break;  
case push_dummy_failure: printf ("/push_dummy_failure"); break; case may-  
_number_and_incr (&mcnt, &p); printf
```

Justificativa


1. São abordados assuntos complexos e abstratos em disciplinas de programação;
2. Há dificuldade por parte dos alunos na interpretação de problemas e formação de raciocínio lógico;
3. Discentes com necessidades especiais tendem a ter suas dificuldades potencializadas;
4. Existe uma alta taxa de evasão em cursos relacionados à área de computação.

```
printf ("/begline"); break; case endlne: printf ("/endlne"); break; case on_failure -  
extract_number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt);  
case on_failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf  
failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-  
tract_number_and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break;  
case push_dummy_failure: printf ("/push_dummy_failure"); break; case may-  
be_dummy_failure: extract_number_and_incr (&mcnt, &p); printf
```

Objetivos

Criação de uma plataforma web educacional com o objetivo de auxiliar o processo de aprendizado de programação

1. Prover uma metodologia alternativa de ensino baseada em elementos visuais;
2. Utilização de símbolos que representam sintaxes dos comandos da linguagem C;
3. Plataforma que possibilita estudar conceitos da linguagem C;



```
printf("/begline"); break; case endlime: printf("/endlime"); break; case on_failure -  
extract_number_and_incr(&mcnt, &p); printf("/on_failure_jump/0/%d", mcnt);  
case on_failure_keep_string_jump: extract_number_and_incr(&mcnt, &p); printf  
failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-  
tract_number_and_incr(&mcnt, &p); printf("/dummy_failure_jump/0/%d", mcnt); break;  
case push_dummy_failure: printf("/push_dummy_failure"); break; case may-  
be_dummy_failure: extract_number_and_incr(&mcnt, &p); printf
```

```
der(destination, source); *source += 1; #endif #define EXTRACT_MACRO-
ROS #undef EXTRACT_NUMBER_AND_INCR #define EXTRACT_NUM-
BER_AND_INCR(dest, src) \extract_number_and_incr (&dest, &src) #endif /*
ot EXTRACT_MACROS */ #endif /* DEBUG */ 0 /* If DEBUG is defined, Regex prints
y voluminous messages about what it is doing (if the variable `debug' is nonzero). If
t with the main program in `iregex.c', you can enter patterns and strings interactively.
linked with the main program in `main.c' and the other test files, you can run the al-
ritten tests. */ #ifdef DEBUG /* We use standard I/O for debugging. */ #include <stdio.h>
l to test things that "must" be true when debugging. */ #include <assert.h> static int
0; #define DEBUG_STATEMENT(e) e #define DEBUG_PRINT1(x) if (debug) printf (x) #define
PRINT2(x1, x2) if (debug) printf (x1, x2) #define DEBUG_PRINT3(x1, x2, x3) if (debug) printf
3) #define DEBUG_PRINT4(x1, x2, x3, x4) if (debug) printf (x1, x2, x3, x4) #define DE-
NT_COMPILED_PATTERN(p, s, e) if (debug) print_partial_compiled_pattern (s, e) #define DE-
DOUBLE_STRING(s1, s2) if (debug) print_double_string(s1, s2) #define DO_FASTMAP(s1, sz1, s2, sz2)
ntchar_t * Print_double_string(s1, s2) { if (debug) print_double_string(s1, s2); return s1; }
unsigned int a_range = 1; unsigned int i; while (i < (1 << BYTEWIDTH)) { if (fastmap[i++])
printchar (i - 1); while (i < (1 << BYTEWIDTH)) { was_a_range = 1; i++; } if
f ("-"); print_double_string(s1, s2); } #endif #define PRINT_PATTERN_STRING_IN_HU-
ing at the end of the pattern string. The pointer END. */ void
pattern (start, end) unsigned char *start; unsigned char *end; { int mcnt, mcnt2; un-
signed char *end; while (start != NULL) { printf ("(null)"); return; } /* Loop over
ids. */ while (p < pend) { switch ((re_opcode_t) *p++) { case no_op: printf ("/no_op");
cnt, mcnt = *p++; printf ("/exactn/%d", mcnt); do { putchar ('/'); printchar (*p++); }
break; case start_memory: mcnt = *p++; printf ("/start_memory/%d/%d", mcnt,
e stop_memory: mcnt = *p++; printf ("/stop_memory/%d/%d", mcnt, *p++);
cate; printf ("/duplicate/%d", *p++); break; case anychar: printf ("/anychar");
rset; case charset_not; { register int c; printf ("/charset%s", (re_opcode_t) *(p
t ? "_not": ""); assert (p + *p < pend); for (c = 0; c < *p; c++) { unsigned bit;
map_byte = p[1 + c]; putchar ('/'); for (bit = 0; bit < BYTEWIDTH; bit++) if
(1 << bit)) printchar (c * BYTEWIDTH + bit); } p += 1 + *p; break; } case beg-
egline"); break; case newline: printf ("/newline"); break; case on_failure_-
number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt);
failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf
ep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-
and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break;
```

Uma versão renovada

Outras soluções



VISUALG.3
O melhor interpretador de algoritmos

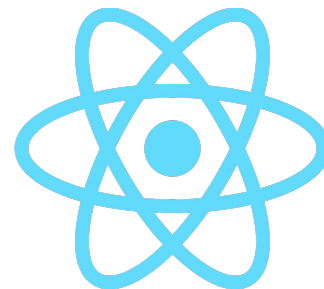
```
printf("/begline"); break; case endlime: printf("/endlime"); break; case on_failure_ -  
extract_number_and_incr(&mcnt, &p); printf("/on_failure_jump/0/%d", mcnt);  
case on_failure_keep_string_jump: extract_number_and_incr(&mcnt, &p); printf  
failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex -  
number_and_incr(&mcnt, &p); printf("/dummy_failure_jump/0/%d", mcnt); break;  
case push_dummy_failure: printf("/push_dummy_failure"); break; case may -  
for_dummy_extract_number_and_incr(&mcnt, &p); printf
```



```
ber(destination, &source); #endif #define EXTRACT_MACRO-
ROS #undef EXTRACT_NUMBER_AND_INCR #define EXTRACT_NUM-
BER_AND_INCR(dest, src) \extract_number_and_incr (&dest, &src) #endif /*
ot EXTRACT_MACROS */ #endif /* DEBUG */ #endif /* If DEBUG is defined, Regex prints
y voluminous messages about what it is doing (if the variable `debug' is nonzero). If
t with the main program in `iregex.c', you can enter patterns and strings interactively.
linked with the main program in `main.c' and the other test files, you can run the al-
ritten tests. */ #ifdef DEBUG /* We use standard I/O for debugging. */ #include <stdio.h>
l to test things that "must" be true when debugging. */ #include <assert.h> static int
0; #define DEBUG_STATEMENT(e) e #define DEBUG_PRINT1(x) if (debug) printf (x) #define
PRINT2(x1, x2) if (debug) printf (x1, x2) #define DEBUG_PRINT3(x1, x2, x3) if (debug) printf
3) #define DEBUG_PRINT4(x1, x2, x3, x4) if (debug) printf (x1, x2, x3, x4) #define DE-
NT_COMPILED_PATTERN(p, s, e) if (debug) print_partial_compiled_pattern (s, e) #define DE-
DOUBLE_STRING(w, s1, sz1, s2, sz2) \ if (debug) print_double_string (w, s1, sz1, s2, sz2)
ntchar(); /* Print the fastmap in human-readable form. */ void print_fastmap (fastmap)
unsigned was_a_range = 0; unsigned i = 0; while (i < (1 << BYTEWIDTH)) { if (fastmap[i++])
printf ("%d", i); while (i < (1 << BYTEWIDTH) && fastmap[i]) { was_a_range = 1; i++; } if
f (" "); print_double_string ("Fastmap", fastmap); } #endif /* DEBUG */
ting at the end of the file. If you are using the library, you can compile the library
pattern. If you are using the library, you can compile the library. If you are using
insignificant, but it is not a problem. If you are using the library, you can compile
ids. */ while (p < pend) { switch ((re_opcode_t) *p++) { case no_op: printf ("/no_op");
ctn; mcnt = *p++; printf ("/exactn/%d", mcnt); do { putchar ('/'); putchar (*p++); }
break; case start_memory: mcnt = *p++; printf ("/start_memory/%d/%d", mcnt,
e stop_memory: mcnt = *p++; printf ("/stop_memory/%d/%d", mcnt, *p++);
cate; printf ("/duplicate/%d", *p++); break; case anychar: printf ("/anychar");
rset; case charset_not; { register int c; printf ("/charset%s", (re_opcode_t) *(p
t ? "_not" : ""); assert (p + *p < pend); for (c = 0; c < *p; c++) { unsigned bit;
map_byte = p[1 + c]; putchar ('/'); for (bit = 0; bit < BYTEWIDTH; bit++) if
(1 << bit)) putchar (c * BYTEWIDTH + bit); } p += 1 + *p; break; } case beg-
egline"); break; case endline: printf ("/endline"); break; case on_failure_-
number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt);
failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf
ep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-
and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break;
```

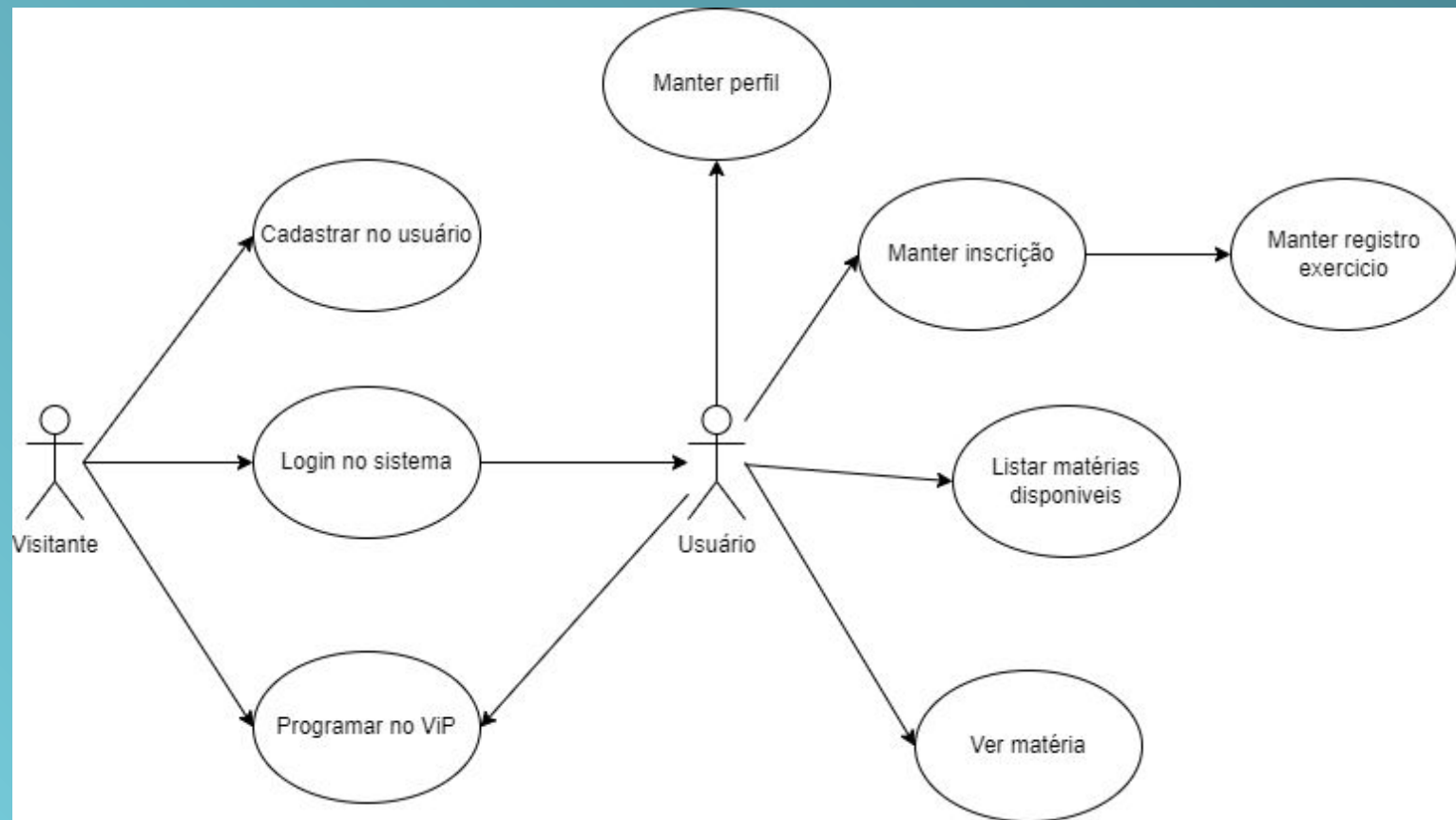
Metodologia

Tecnologias escolhidas



```
printf("/begline"); break; case endlime: printf("/endlime"); break; case on_failure -  
extract_number_and_incr(&mcnt, &p); printf("/on_failure_jump/0/%d", mcnt);  
case on_failure_keep_string_jump: extract_number_and_incr(&mcnt, &p); printf  
failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-  
number_and_incr(&mcnt, &p); printf("/dummy_failure_jump/0/%d", mcnt); break;  
case push_dummy_failure: printf("/push_dummy_failure"); break; case may-  
for_dummy_extract_number_and_incr(&mcnt, &p); printf
```

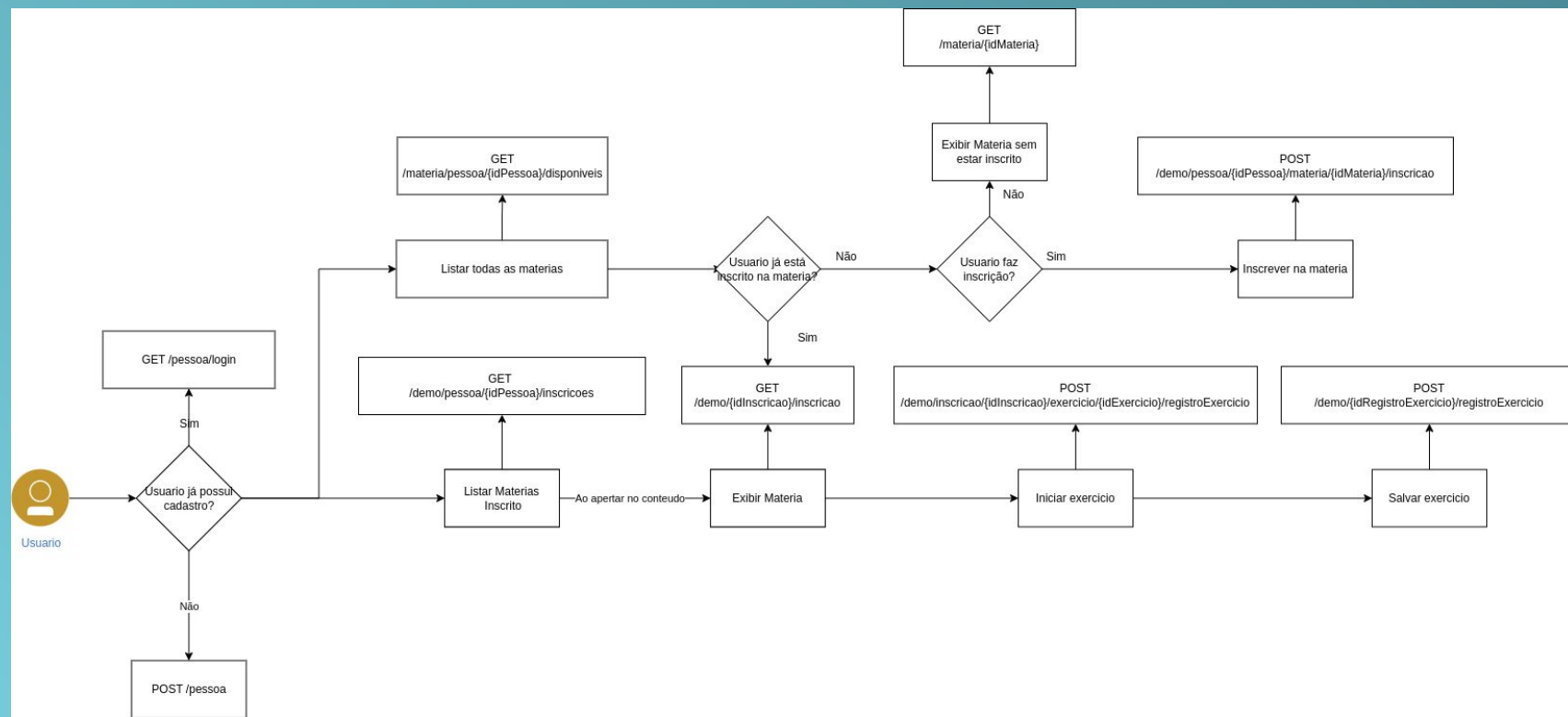

Caso de uso



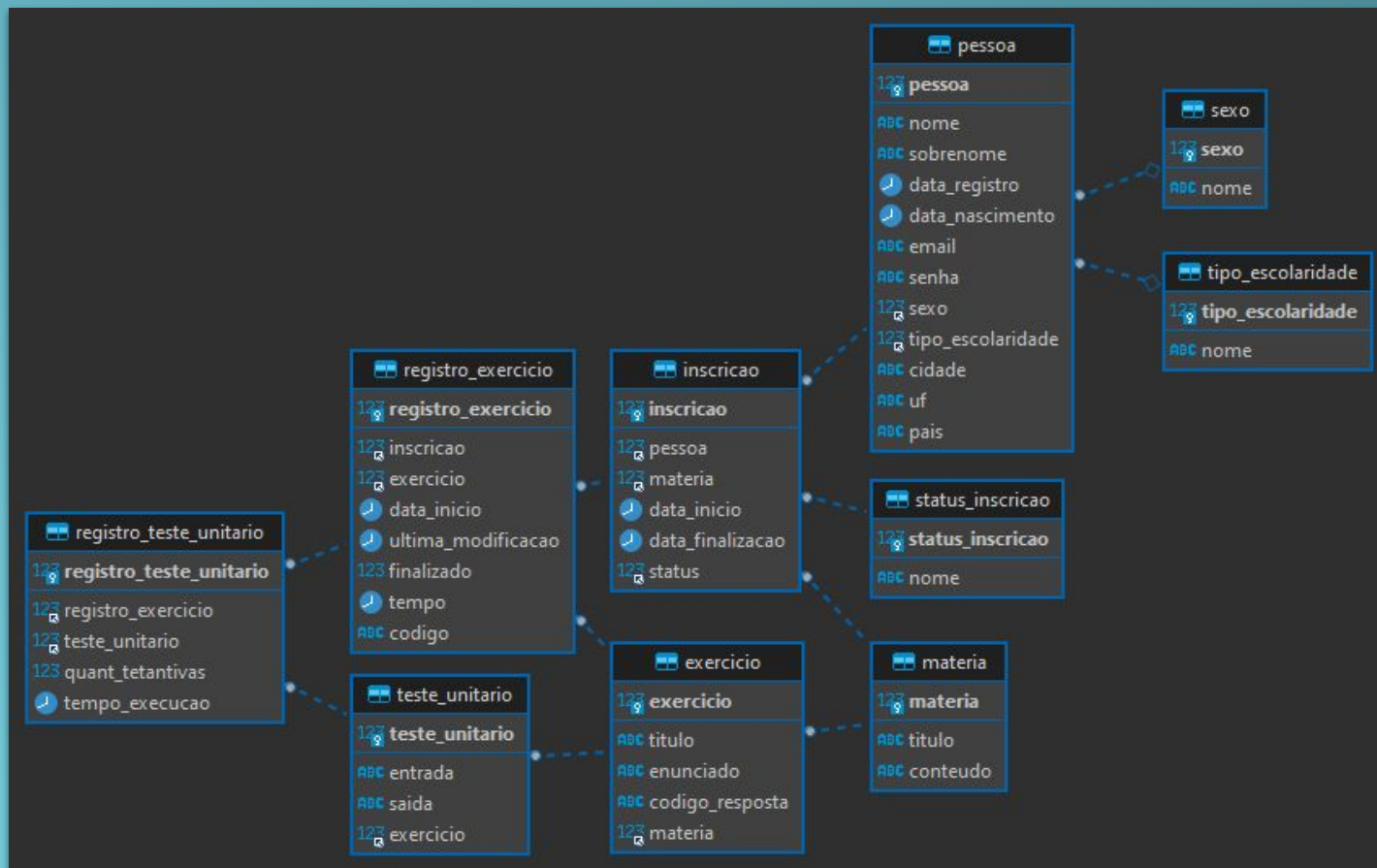
```
ber(destination, source); *source += 1; #endif EXTRACT_MACRO-
ROS #undef EXTRACT_NUMBER_AND_INCR #define EXTRACT_NUM-
BER_AND_INCR(dest, src) \extract_number_and_incr (&dest, &src) #endif /*
ot EXTRACT_MACROS */ #endif /* DEBUG */ 0 /* If DEBUG is defined, Regex prints
y voluminous messages about what it is doing (if the variable `debug' is nonzero). If
t with the main program in `iregex.c', you can enter patterns and strings interactively.
linked with the main program in `main.c' and the other test files, you can run the al-
ritten tests. */ #ifdef DEBUG /* We use standard I/O for debugging. */ #include <stdio.h>
l to test things that "must" be true when debugging. */ #include <assert.h> static int
0; #define DEBUG_STATEMENT(e) e #define DEBUG_PRINT1(x) if (debug) printf (x) #define
PRINT2(x1, x2) if (debug) printf (x1, x2) #define DEBUG_PRINT3(x1, x2, x3) if (debug) printf
3) #define DEBUG_PRINT4(x1, x2, x3, x4) if (debug) printf (x1, x2, x3, x4) #define DE-
NT_COMPILED_PATTERN(p, s, e) if (debug) print_partial_compiled_pattern (s, e) #define DE-
DOUBLE_STRING(w, s1, sz1, s2, sz2) \ if (debug) print_double_string (w, s1, sz1, s2, sz2)
ntchar(); /* Print the fastmap in human-readable form. */ void print_fastmap (fastmap)
unsigned was_a_range = 0; unsigned i = 0; while (i < (1 << BYTEWIDTH)) { if (fastmap[i++]
printchar(' '); while (i < (1 << BYTEWIDTH) && fastmap[i]) { was_a_range = 1; i++; } if
f (" "); printchar(' '); i++; } } void print_double_string (w, s1, sz1, s2, sz2)
ting at the START of the string. If the string is not found, print the string and the
pattern. If the string is found, print the string and the pattern. If the string is
insignificant, don't print it. If (start == NULL) { printf ("(null)\n"); return; } /* Loop over
ids. */ while (p < pend) { switch ((re_opcode_t) *p++) { case no_op: printf ("/no_op");
ctn; mcnt = *p++; printf ("/exactn/%d", mcnt); do { putchar ('/'); printchar (*p++); }
break; case start_memory: mcnt = *p++; printf ("/start_memory/%d/%d", mcnt,
e stop_memory: mcnt = *p++; printf ("/stop_memory/%d/%d", mcnt, *p++);
cate; printf ("/duplicate/%d", *p++); break; case anychar: printf ("/anychar");
rset; case charset_not; { register int c; printf ("/charset%s", (re_opcode_t) *(p
t ? "_not" : ""), assert (p + *p < pend); for (c = 0; c < *p; c++) { unsigned bit;
map_byte = p[1 + c]; putchar ('/'); for (bit = 0; bit < BYTEWIDTH; bit++) if
(1 << bit)) printchar (c * BYTEWIDTH + bit); } p += 1 + *p; break; } case beg-
egline"); break; case endline: printf ("/endline"); break; case on_failure_-
number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt);
failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf
ep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-
and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break;
```

Prototipação

Fluxograma backend



Base de Dados



```
ber(destination, "source"); #endif #define EXTRACT_MACRO-
ROS #undef EXTRACT_NUMBER_AND_INCR #define EXTRACT_NUM-
BER_AND_INCR(dest, src) \extract_number_and_incr (&dest, &src) #endif /*
ot EXTRACT_MACROS */ #endif /* DEBUG */ 0 /* If DEBUG is defined, Regex prints
y voluminous messages about what it is doing (if the variable `debug' is nonzero). If
t with the main program in `iregex.c', you can enter patterns and strings interactively.
linked with the main program in `main.c' and the other test files, you can run the al-
ritten tests. */ #ifdef DEBUG /* We use standard I/O for debugging. */ #include <stdio.h>
l to test things that "must" be true when debugging. */ #include <assert.h> static int
0; #define DEBUG_STATEMENT(e) e #define DEBUG_PRINT1(x) if (debug) printf (x) #define
PRINT2(x1, x2) if (debug) printf (x1, x2) #define DEBUG_PRINT3(x1, x2, x3) if (debug) printf
3) #define DEBUG_PRINT4(x1, x2, x3, x4) if (debug) printf (x1, x2, x3, x4) #define DE-
NT_COMPILED_PATTERN(p, s, e) if (debug) print_partial_compiled_pattern (s, e) #define DE-
DOUBLE_STRING(w, s1, sz1, s2, sz2) \ if (debug) print_double_string (w, s1, sz1, s2, sz2)
ntchar(); /* Print the fastmap in human-readable form. */ void print_fastmap (fastmap)
unsigned was_a_range = 0; unsigned i = 0; while (i < (1 << BYTEWIDTH)) { if (fastmap[i++])
printf ("%d", i); while (i < (1 << BYTEWIDTH) && fastmap[i]) { was_a_range = 1; i++; } if
f ("-"); print_double_string ("Fastmap", fastmap, 1); } void print_compiled_pattern string in hu-
ing at the pointer. All the patterns are terminated by the pointer END. */ void
pattern (start, end) unsigned char *start; unsigned char *end; { int mcnt, mcnt2; un-
signed char *p; while (p < end) { switch ((re_opcode_t) *p++) { case no_op: printf ("/no_op");
cnt, mcnt = *p++; printf ("/exactn/%d", mcnt); do { putchar ('/'); putchar (*p++); }
break; case start_memory: mcnt = *p++; printf ("/start_memory/%d/%d", mcnt,
e stop_memory: mcnt = *p++; printf ("/stop_memory/%d/%d", mcnt, *p++);
cate; printf ("/duplicate/%d", *p++); break; case anychar: printf ("/anychar");
rset; case charset_not; { register int c; printf ("/charset%s", (re_opcode_t) *(p
t ? "_not" : ""), assert (p + *p < pend); for (c = 0; c < *p; c++) { unsigned bit;
map_byte = p[1 + c]; putchar ('/'); for (bit = 0; bit < BYTEWIDTH; bit++) if
(1 << bit)) putchar (c * BYTEWIDTH + bit); } p += 1 + *p; break; } case beg-
egline"); break; case newline: printf ("/newline"); break; case on_failure_-
number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt);
failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf
ep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-
and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break;
```

Swagger


```
ber(destination, source); #endif #define EXTRACT_MACRO-
ROS #undef EXTRACT_NUMBER_AND_INCR #define EXTRACT_NUM-
BER_AND_INCR(dest, src) \extract_number_and_incr (&dest, &src) #endif /*
ot EXTRACT_MACROS */ #endif /* DEBUG */ #if defined, Regex prints
y voluminous messages about what it is doing (if the variable `debug' is nonzero). If
t with the main program in `iregex.c', you can enter patterns and strings interactively.
linked with the main program in `main.c' and the other test files, you can run the al-
ritten tests. */ #ifdef DEBUG /* We use standard I/O for debugging. */ #include <stdio.h>
l to test things that "must" be true when debugging. */ #include <assert.h> static int
0; #define DEBUG_STATEMENT(e) e #define DEBUG_PRINT1(x) if (debug) printf (x) #define
PRINT2(x1, x2) if (debug) printf (x1, x2) #define DEBUG_PRINT3(x1, x2, x3) if (debug) printf
3) #define DEBUG_PRINT4(x1, x2, x3, x4) if (debug) printf (x1, x2, x3, x4) #define DE-
NT_COMPILED_PATTERN(p, s, e) if (debug) print_partial_compiled_pattern (s, e) #define DE-
DOUBLE_STRING(w, s1, sz1, s2, sz2) \ if (debug) print_double_string (w, s1, sz1, s2, sz2)
ntchar(); /* Print the fastmap in human-readable form. */ void print_fastmap (fastmap)
unsigned was_a_range = 0; unsigned i = 0; while (i < (1 << BYTEWIDTH)) { if (fastmap[i++]
printf ("%d", fastmap[i]); while (i < (1 << BYTEWIDTH) && fastmap[i]) { was_a_range = 1; i++; } if
f ("-"); printf ("%d", fastmap[i]); } printf ("%d", fastmap[i]); } /* Print the fastmap in hu-
ting at the end of the fastmap. */ void print_fastmap_end () { void
pattern (p, s, e) { unsigned mcnt1 = 0; unsigned mcnt2; un-
signed char *p; while (p < pend) { switch ((re_opcode_t) *p++) { case no_op: printf ("/no_op");
ctn; mcnt = *p++; printf ("/exactn/%d", mcnt); do { putchar ('/'); putchar (*p++); }
break; case start_memory: mcnt = *p++; printf ("/start_memory/%d/%d", mcnt,
e stop_memory: mcnt = *p++; printf ("/stop_memory/%d/%d", mcnt, *p++);
cate; printf ("/duplicate/%d", *p++); break; case anychar: printf ("/anychar");
rset; case charset_not; { register int c; printf ("/charset%s", (re_opcode_t) *(p
t ? "_not" : ""); assert (p + *p < pend); for (c = 0; c < *p; c++) { unsigned bit;
map_byte = p[1 + c]; putchar ('/'); for (bit = 0; bit < BYTEWIDTH; bit++) if
(1 << bit)) putchar (c * BYTEWIDTH + bit); } p += 1 + *p; break; } case beg-
egline"); break; case endline: printf ("/endline"); break; case on_failure_-
number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt);
failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf
ep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-
and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break;
```

Resultados

Situação do projeto

**Pesquisa e
Modelagem**



**Prototipagem das
telas**



**Criação de banco
de dados e apis**



**Execução do
frontend**



```
printf("/begline"); break; case endlime: printf("/endlime"); break; case on_failure -  
extract_number_and_incr(&mcnt, &p); printf("/on_failure_jump/0/%d", mcnt);  
case on_failure_keep_string_jump: extract_number_and_incr(&mcnt, &p); printf  
failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-  
number_and_incr(&mcnt, &p); printf("/dummy_failure_jump/0/%d", mcnt); break;  
case push_dummy_failure: printf("/push_dummy_failure"); break; case may-  
for_dummy_extract_number_and_incr(&mcnt, &p); printf
```


Considerações finais

1. O projeto está caminhando como previsto com o seu desenvolvimento bem avançado;
2. A parte da modelagem e o seu escopo estão bem definidos;
3. Ainda é necessário realizar alguns apontamentos no artigo;

```
printf ("/begline"); break; case endlne: printf ("/endlne"); break; case on_failure_ -  
extract_number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt);  
case on_failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf  
failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-  
number_and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break;  
case push_dummy_failure: printf ("/push_dummy_failure"); break; case may-  
for_dummy_extract_number_and_incr (&mcnt, &p); printf
```

Obrigada!

Aluna: Ana Paula Marques Barbosa

Orientador: Gil Eduardo de Andrade

Co-orientador: Alexandre Chiarelli

```
printf ("/begin"); break; case endline: printf ("/endline"); break; case on_failure -  
extract_number_and_incr (&mcnt, &p); printf ("/on_failure_jump/0/%d", mcnt);  
case on_failure_keep_string_jump: extract_number_and_incr (&mcnt, &p); printf  
failure_keep_string_jump/0/%d", mcnt); break; case dummy_failure_jump: ex-  
number_and_incr (&mcnt, &p); printf ("/dummy_failure_jump/0/%d", mcnt); break;  
case push_dummy_failure: printf ("/push_dummy_failure"); break; case may-  
for_dummy_extract_number_and_incr (&mcnt, &p); printf
```