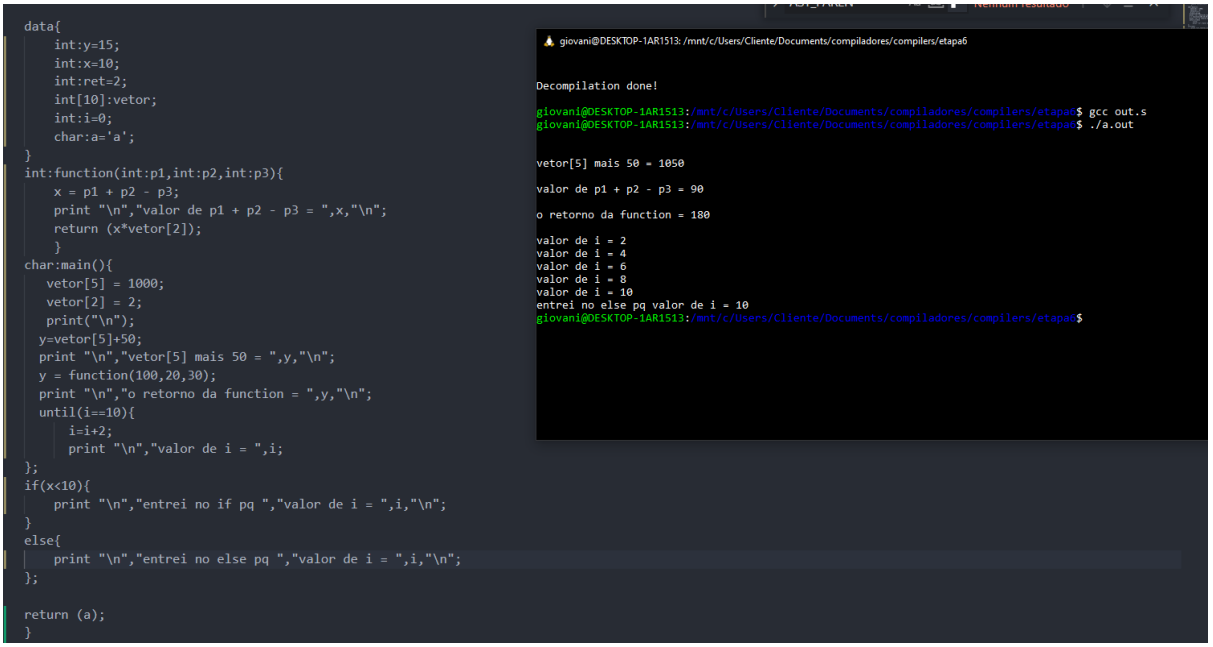
Relatório Etapa 6 Compiladores - Giovani da Silva

Exemplo 1:



```
data{
    int:y=15;
    int:x=10;
    int:ret=2;
    int[10]:vetor;
    int:i=0;
    char:a='a';
}
int:function(int:p1,int:p2,int:p3){
    x = p1 + p2 - p3;
    print "\n","valor de p1 + p2 - p3 = ",x,"\n";
    return (x*vetor[2]);
    }
char:main(){
   vetor[5] = 1000;
   vetor[2] = 2;
   print("\n");
  y=vetor[5]+50;
  print "\n","vetor[5] mais 50 = ",y,"\n";
  y = function(100,20,30);
  print "\n","o retorno da function = ",y,"\n";
  until(i==10){
      i=i+2;
      print "\n","valor de i = ",i;
};
if(x<10){
    print "\n","entrei no if pq ","valor de i = ",i,"\n";
}
else{
    print "\n","entrei no else pq ","valor de i = ",i,"\n";
};

return (a);
}
```

```
giovani@DESKTOP-1AR1513: /mnt/c/Users/Cliente/Documents/compiladores/compilers/etapa6


Decompilation done!

giovani@DESKTOP-1AR1513:/mnt/c/Users/Cliente/Documents/compiladores/compilers/etapa6$ gcc out.s
giovani@DESKTOP-1AR1513:/mnt/c/Users/Cliente/Documents/compiladores/compilers/etapa6$ ./a.out


vetor[5] mais 50 = 1050

valor de p1 + p2 - p3 = 90

o retorno da function = 180

valor de i = 2
valor de i = 4
valor de i = 6
valor de i = 8
valor de i = 10
entrei no else pq valor de i = 10
giovani@DESKTOP-1AR1513:/mnt/c/Users/Cliente/Documents/compiladores/compilers/etapa6$
```

Exemplo 2 de demonstração de um programa compilável:

```
data{
int[5]:vetor;
int:y=0;
int:x=100;
int:i=0;
int:operacao=0;
}

int:function(int:parametro1,int:parametro2,int:parametro3){
    parametro2 = 200;
    parametro3 = 500;
    parametro1 = parametro1 + parametro3 - parametro2;
    print("valor do p1 eh ");
    print(parametro1);
    print("\n");
    return (parametro1);
}
int:main(){
vetor[2] = 77;
y=vetor[2];
print("\n");
print("o valor de y eh ");
print(y);
print("\n");
print("\n");
x = function(100,0,0) + 50;
print("retorno mais 50 eh ");
print(x);
print("\n");
print("oi vou entrar no until");
print("\n");
```

```
until(i==7){
    print("valor de i eh ");
    print(i);
     print("\n");
    i=i+1;
};
print("\n");
print("oi sai do until ");
operacao = vetor[2] + 50;
if(operacao>50){
    print("\n");
    print("estou dentro do if");
    print("\n");
}       You, seconds ago • Uncommitted changes
else{
    print("\n");
    print("estou dentro do else");
    print("\n");
};









}
```

POR PARTES:

```
vetor[2] = 77;
y=vetor[2];
print("\n");       You, a week ago
print("o valor de y eh ");
print(y);
print("\n");
print("\n");
```

```
o valor de y eh 77
```

```
x = function(100,0,0) + 50;
```

```
int:function(int:parametro1,int:parametro2,int:parametro3){


    parametro2 = 200;
    parametro3 = 500;
    parametro1 = parametro1 + parametro3 - parametro2;
    print("valor do p1 eh ");
    print(parametro1);
    print("\n");

    return (parametro1);
}
```

```
x = function(100,0,0) + 50;
print("retorno mais 50 eh ");
print(x);
print("\n");
```

```
valor do p1 eh 400
retorno mais 50 eh 450
```

Parametro1 = 100 + 500 - 200 = 400
Retorno = Parametro1 + 50 = 450

```
print("oi vou entrar no until");
print("\n");
until(i==7){

    print("valor de i eh ");
    print(i);
     print("\n");
    i=i+1;

};
print("\n");       You, a week ago • re

print("oi sai do until ");
```

```
oi vou entrar no until
valor de i eh 0
valor de i eh 1
valor de i eh 2
valor de i eh 3
valor de i eh 4
valor de i eh 5
valor de i eh 6

oi sai do until
```

```
operacao = vetor[2] + 50;
if(operacao>50){
    print("\n");
    print("estou dentro do if");
    print("\n");
}
else{
    print("\n");
    print("estou dentro do else");
    print("\n");
};
```

```
estou dentro do if
```

Algumas explicações sobre a implementação:
Operações aritméticas foram divididas em dois grupos, um com expressões que usam o registrador do assembler eax antes de edx, como addl, e outras que usam edx antes de eax, como o subl.

Em ambas foi necessário usar um método is_function que observa se algum dos parâmetros é uma função, caso seja, irá pegar o retorno dessa função, adicionando um _Return no nome e colocando no registrador correspondente.
Quando se inicia uma função é sempre adicionado nesse array o nome da função para que essa verificação seja feita.

```
void asm_beginfun(TAC *tac)
{

    fprintf(fout, "## TAC_BEGINFUN\n"
                  ".globl   _%s\n"
                  "_%s:\n"
                  "\tpushq  %%rbp\n"
                  "\tmovq   %%rsp, %%rbp\n\n",

            tac->res->text, tac->res->text);

    arraydefuncoes[funcoes_index] = tac->res->text;
    funcoes_index++;
}
```

```
void asm_aritm_operation_edx_eax(TAC *tac, char *instruction)        You, a wee
{

    if (is_function(tac->op2->text) == 1 || is_function(tac->op1->text) == 1)
    {
        fprintf(fout,"## OPERATION WITH FUNCTION CALL EDX EAX\n");
        fprintf(fout, "\tmovl\t_%s_Return(%%rip), %%eax\n"
                      "\tmovl\t_%s(%%rip), %%edx\n"
                      "\t%s\t%%edx, %%eax\n"
                      "\tmovl\t%%eax, _%s(%%rip)\n",
                tac->op1->text, tac->op2->text,
                instruction, tac->res->text);
    }

    else
    {
        fprintf(fout,"## OPERATION EDX EAX\n");
        fprintf(fout, "\tmovl\t_%s(%%rip), %%edx\n"
                      "\tmovl\t_%s(%%rip), %%eax\n"
                      "\t%s\t%%edx, %%eax\n"
                      "\tmovl\t%%eax, _%s(%%rip)\n",
                tac->op1->text, tac->op2->text,
                instruction, tac->res->text);
    }
}
```

Sempre que uma função é declarada essa é adicionada ao vetor de nome de funções, que é utilizado nessa função is_function

```
int is_function(char *functionName){
    char buffer[256];
    sprintf(buffer,"_%s",functionName);
    for(int i=0;i<100;i++){
        //printf("\n\tbuffer:%s\tarraydefuncoes[%d]:%s\n",functionName,i,arraydefuncoes[i]);
        if(arraydefuncoes[i] == functionName){


            return 1;
        }
    }
    return 0;       You, seconds ago • Uncommitted changes
}
```

Em caso de vetores, temos um caso bem parecido,temos também que verificar se é vetor sempre, pegar a posição requisitada e colocar corretamente no asm, tanto no print quanto em operações.
Adicionando ao array de nome de vetores

```
TAC *asm_var_declar(TAC *first)
{
    TAC *tac = first;

    fprintf(fout, "\n## VAR DECLARATION\n"
                  ".globl   main\n"
                  "main:\n"
                  "\tpushq  %%rbp\n"
                  "\tmovq   %%rsp, %%rbp\n\n");

    while (tac->type == TAC_DECL_VAR || tac->type == TAC_DECL_VECTOR)
    {

        if (tac)
        {

            asm_move(tac);
            tac = tac->next;
            if (tac->type == TAC_DECL_VECTOR)
            {

                vectorsNames[vectorCount] = tac->res->text;
                vectorCount++;
            }
        }
```

Essa é a função que pega a posição solicitada.

```c
char *str_treatment_vector_lenght(char *str)
{

    const char *PATTERN1 = "[";
    const char *PATTERN2 = "]";

    char *target = NULL;
    char *start, *end;

    if (start = strstr(str, PATTERN1))
    {
        start += strlen(PATTERN1);
        if (end = strstr(start, PATTERN2))
        {
            target = (char *)malloc(end - start + 1);
            memcpy(target, start, end - start);
            target[end - start] = '\0';
        }
    }

    if (target)
    {
       // printf("target:%s\n", target);
        return target;
    }

    return str;
}
```

```c
char *str_treatment_vector(char *str)
{



    const char separator = '[';
    char *const sep_at = strchr(str, separator);
    if (sep_at != NULL)
    {
        *sep_at = '\0';
    }

    return str;
}
```

Essa é o método que pega o nome do vetor que sera colocado naquela lista.

Para o print funcionar corretamente, podendo repetir strings, tive que declarar cada print como uma "variável" com seu nome, isso ocasionou alguns problemas com caracteres inválidos que são tratados na função str_treatment.

```c
void str_treatment(char *str_in)
{
    char *string1;
    string1 = malloc(sizeof(str_in));
    strcpy(string1, str_in);
    //printf("\nstring antes da remocao de espaco: %s\n", string1);
    char *str = str_in;
    char *write = str, *read = str;
    do
    {
        if (*read != ' ')
            *write++ = *read;
    } while (*read++);
    //printf("\nstring dps da remocao de espaco: %s\n", str);
    //trocar /n por barra n
    char *str1 = str_in;

    char *write1 = str1, *read1 = str1;
    do      You, seconds ago • Uncommitted changes
    {
        if (*read1 != '\\')
            *write1++ = *read1;
    } while (*read1++);
    //printf("\nstring dps da remocao de barra: %s\n", str1);
    char *str2 = str_in;
    char *write2 = str2, *read2 = str2;
    do
    {
        if (*read2 != '\"')
            *write2++ = *read2;
    } while (*read2++);
    //printf("\nstring dps da remocao de aspas: %s\n", str2);
}
```

Também há uma verificação para que a mesma string não seja declarada duas vezes e ocasione erros no compilador.

```
void print_asm(FILE *fout)
{

    fprintf(fout, "\n## DATA SECTION\n"
                  ".data\n");

    HASH_NODE *node;
    for (int i = 0; i < HASH_SIZE; i++)
    {
        for (node = Table[i]; node; node = node->next)
        {
            if (!search_already(node->text))
            {
                names[p] = node->text;
                p++;

                switch (node->type)
                {

                case SYMBOL_FUNCTION:
                    fprintf(fout, "_%s_Return: .long\t0\n", node->text);
                    break;
                case SYMBOL_VARIABLE:
                    fprintf(fout, "_%s: .long\t0\n", node->text);
                    break;
```