Diego Souza Lima Marques – TIA: 32039921

Paradigmas de Linguagens de Programação – 05D

Desafio 4

Semana 3 – Vale até 

Programa A:

Código	Layout de memória

//Programa A
#include <stdio.h>
 	
int f(int x, int y)
{
  int z;
  x=1;
  y=1;
  z = x+y;
  return z;
}
 	
int main(void) {
  int x=0,y=0,z=0;
  printf("%d %d %d\n",x,y,z);
  z = f(x,y);
  printf("%d %d %d\n",x,y,z);
  return 0;
}
 	
Passagem por: valor
	<table><tr><td>ARI -> main()</td><td></td></tr><tr><td>x</td><td>0</td></tr><tr><td>y</td><td>0</td></tr><tr><td>z</td><td>0</td></tr><tr><td>ARI -> f()</td><td></td></tr><tr><td>z VL</td><td>0 (ou lixo de memória)</td></tr><tr><td>x (cópia)</td><td>1</td></tr><tr><td>y (cópia)</td><td>1</td></tr><tr><td>z VL</td><td>2</td></tr><tr><td>ARI -> main()</td><td></td></tr><tr><td>x</td><td>0</td></tr><tr><td>y</td><td>0</td></tr><tr><td>z</td><td>2</td></tr></table>

Dentro da função f, visto que x e y são passados como argumentos, são criadas cópias do conteúdo de x e y na main, mas em endereços diferentes. Além disso, a variável z dentro da função f() é declarada como uma variável local, pertencente apenas ao contexto daquela função, e sem qualquer relação com a variável z da main. Por fim, a passagem é feita claramente por valor, visto que o conteúdo de x e y, na main, não foi alterado após a execução da função f(), já que se criaram apenas cópias.

Programa A_trace:

Código

```
//Programa A_trace
#include <stdio.h>

int f(int x, int y)
{
  int z;
  printf("Endereços (l-values) no RA de f:\n");
  printf("%X %X %X\n",&x,&y,&z);
  x=1;
  y=1;
  z = x+y;
  printf("Valores (r-values) no RA de f:\n");
  printf("%d %d %d\n",x,y,z);
  return z;
}

int main(void) {
  int x=0,y=0,z=0;
  printf("Endereços (l-values) no RA da main:\n");
  printf("%X %X %X\n",&x,&y,&z);
  printf("Valores (r-values) no RA da main:\n");
  printf("%d %d %d\n",x,y,z);
  z = f(x,y);
  printf("Novos valores (r-values) no RA da main após chamar f:\n");
  printf("%d %d %d\n",x,y,z);
  return 0;
}
```

Passagem por: valor

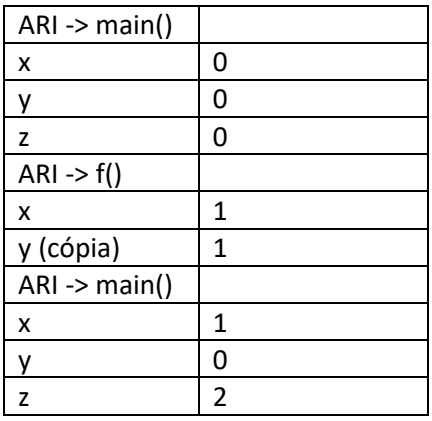
Layout de memória

ARI -> main()		Endereços
x	0	B11458EC
y	0	B11458F0
z	0	B11458F4
ARI -> f()		
z VL	0 ou lixo	B11458C4
x (cópia)	1	B11458BC
y (cópia)	1	B11458B8
z VL	2	B11458C4
ARI -> main()		
x	0	B11458EC
y	0	B11458F0
z	2	B11458F4

Ocorre o mesmo processo utilizado no programa A anterior, apenas explicitando e fazendo um track dos valores e endereços das variáveis. Continua sem mudar as variáveis na main e, portanto, continua sendo passagem por valor.

Programa B:

Código	Layout de memória

//Programa B
#include <stdio.h>
 	
int f(int *x, int y)
{
  *x=1;
  y=1;
  return *x+y;
}
 	
int main(void) {
  int x=0,y=0,z=0;
  printf("%d %d %d\n",x,y,z);
  z = f(&x,y);
  printf("%d %d %d\n",x,y,z);
  return 0;
}
 	
Passagem por: valor (y), referência (x)
	

Em C, não existe passagem por referência com o operador &, por exemplo. Porém, no programa acima, a função f tem como parâmetro um ponteiro, sendo este o modo possível para fazer passagem por referência. Além disso, o ponteiro é uma variável que aponta para um endereço de memória – no caso, o x da função aponta para o endereço de memória do x da main. Ou seja, o ponteiro explicita o l-value da variável x da main. Deste modo, o que acontece dentro da função f() reflete na main para essa variável. Para y, que não usa ponteiro, a passagem é por valor, visto que é criada uma cópia com o r-value da variável y para o escopo da função f().

Programa C:

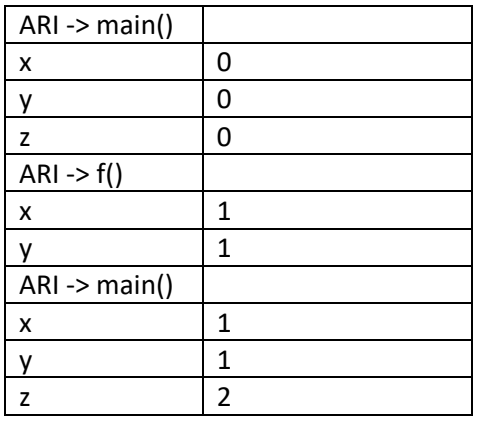
Código	Layout de memória

//Programa C
#include <stdio.h>
 	
int y = 4;
 	
int f(int *x)
{
  *x=1;
  return *x+y;
}
 	
int main(void) {
  int x=0,z=0,*p;
  p = &x;
  printf("%d %d %d\n",x,y,z);
  z = f(p);
  printf("%d %d %d\n",x,y,z);
  return 0;
}
 	
Passagem por: referência
	<table><tr><td>y (global)</td><td>4</td></tr><tr><td>ARI -> main()</td><td></td></tr><tr><td>x</td><td>0</td></tr><tr><td>y</td><td>4</td></tr><tr><td>z</td><td>0</td></tr><tr><td>p</td><td>0 (aponta para x)</td></tr><tr><td>ARI -> f()</td><td></td></tr><tr><td>x</td><td>1</td></tr><tr><td>ARI -> main()</td><td></td></tr><tr><td>x</td><td>1</td></tr><tr><td>y</td><td>4</td></tr><tr><td>z</td><td>5</td></tr><tr><td>p</td><td>1</td></tr></table>

No programa acima, há novamente a passagem por referência utilizando ponteiro para fazer referência a algum endereço (l-value). Porém, desta vez, existe a variável p, um ponteiro que aponta para o endereço (l-value) de x, e é essa variável que é chamada na função f(). No entanto, na prática, o resultado é o mesmo, pois os endereços são o mesmo, alterando o valor de x na main. Além disso, outra particularidade desse código é o uso da variável global y = 4, declarada fora de qualquer função (inclusive a main) e podendo ser utilizada em qualquer parte do código, visto que seu escopo abrange toda parte do código. Ou seja, mesmo que a variável y não tenha sido passada para a função f() como argumento, o escopo da função f() a conhece e a utiliza normalmente.

Programa D:

Código	Layout de memória

//Programa D
#include <iostream>
using namespace std;
 	
int f(int &x, int &y)
{
  x=1;
  y=1;
  return x+y;
}
 	
int main() {
  int x=0,y=0,z=0;
  cout << x << y << z << endl;
  z = f(x,y);
  cout << x << y << z << endl;
  return 0;
}
 	
Passagem por: referência
	

ARI -> main()	
x	0
y	0
z	0
ARI -> f()	
x	1
y	1
ARI -> main()	
x	1
y	1
z	2

O programa acima é feito em C++, linguagem que permite o uso do operador & para fazer passagem por referência. Nesse caso, os parâmetros x e y da função f() já fazem referência ao endereço de memória das variáveis passadas como argumento, sem criar uma cópia e alterando o valor das variáveis na main também.