

Apoio à Ficha de Trabalho N.º 3

Versão 2024/24

**Objectivos:** Estudo de Funções: Passagem por Valor.

Exercícios Introdutórios

Para ajudar a uma melhor compreensão do que é realmente uma função, vai utilizar-se o Pyton Tutor para executar um conjunto de programas que mostram os vários tipos de funções e em que consistem os parâmetros e o valor de retorno, se este existir.

O Pyton Tutor mostra a simulação da execução de um programa de forma visual. Ora, como é sabido, "uma imagem vale mais do que mil palavras". É o que pretenderam os criadores desta ferramenta: ajudar os docentes a explicar conceitos complicados de uma forma visual que mostra como se fosse a própria memória do computador, mostrando a execução, passo a passo do programa, sendo apresentado um diagrama com vários elementos: rectângulos (blocos de memória) que simbolizam representam variáveis, ponteiros, dados estruturados ou blocos de memória genéricos e setas, que representam as ligações entre os blocos de memória (normalmente ligando ponteiros aos respectivos blocos de dados onde residem os valores apontados).

Para introduzir a compreensão da utilidade do Pyton Tutor, num browser, ir a (<https://pythontutor.com/>) que, apesar do seu nome, aceita também código em [**Python**](https://pythontutor.com/python-compiler.html)**,**[**JavaScript**](https://pythontutor.com/javascript.html)**,**[**C**](https://pythontutor.com/c.html)**,**[**C++**](https://pythontutor.com/cpp.html) e[**Java**](https://pythontutor.com/java.html), e usando o bloco de código infra, efectuar a execução do código linha a linha e ir explicando o que vai acontecendo:

Na caixa de texto, vão surgindo os outputs e por baixo desta, vai sendo desenhado o diagrama dos dados (as variáveis) e respectivos endereços.

Será importante que alterem a visualização dos dados a serem apresentados em cada rectângulo do diagrama, para ser lá mostrado o 1.º endereço da memória onde reside (ver Figura 1) na combobox assinalada com a figura oval, em cor azul.

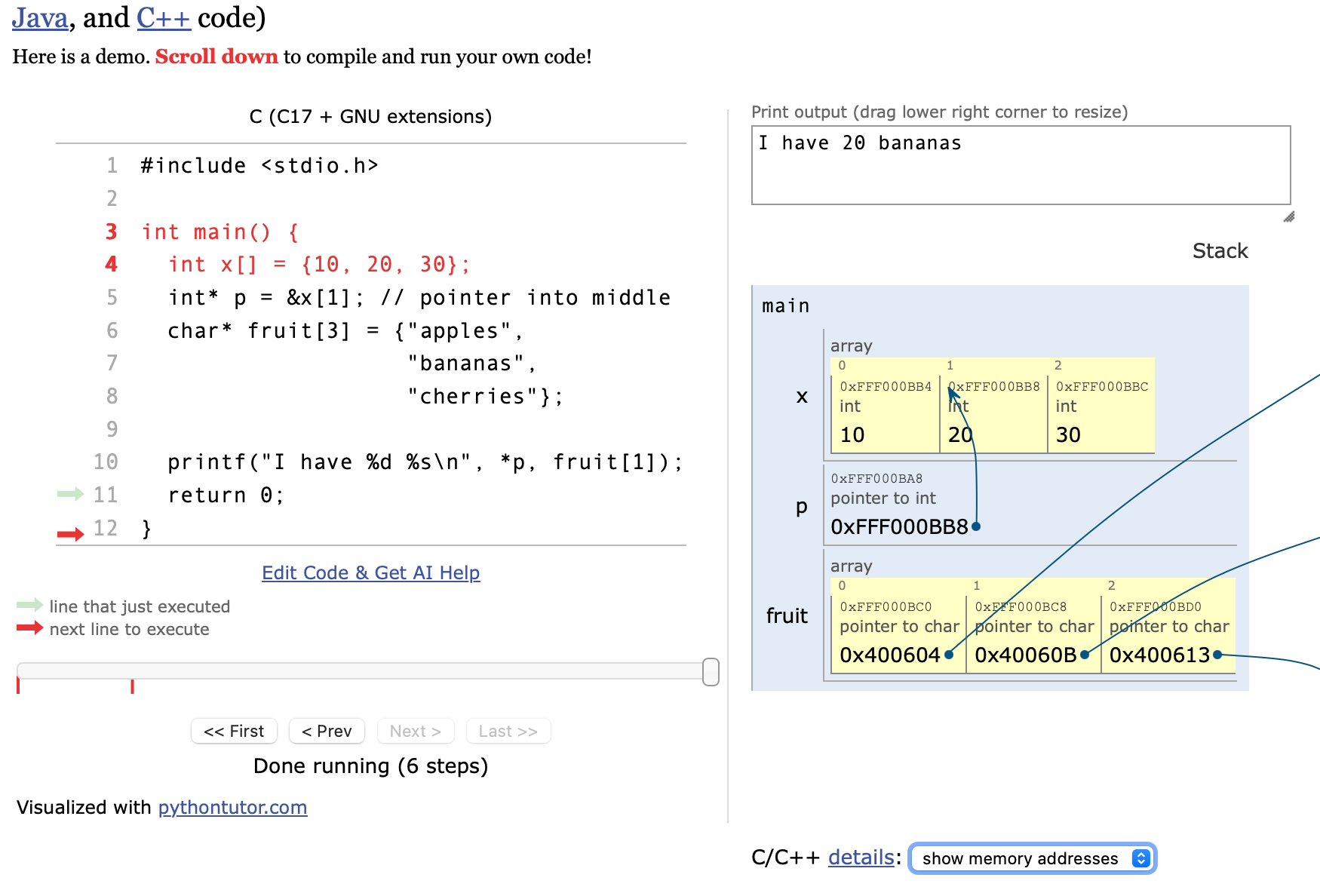


Figura – Demo do Pyton Tutor, com o programa em C, na metade direita e a mostra da sua execução, na metade direita.

Para mostrar as funcionalidades mais importantes da ferramentas aos estudantes e assim poderem percebem perfeitamente o que acontece na execução do exemplos futuros, clicar em Edit Code (o link sinalizado com a oval a cor verde) e inserir o código seguinte:

#include <stdio.h>

int main(){

int x;

int y;

x=10;

y=20;

}

Iniciar a execução do código, linha a linha e, neste caso, mostra-se visualmente a memória onde reside cada variável e, depois, a colocação dos valores nessas memórias quando são executadas as instruções de atribuição. Ou seja, os estudantes, agora passam a ver o que acontece realmente na memória quando é executado um programa, o que estimula o processo de aprendizagem e o facilita.

Depois da explicação breve do Pyton Tutor, para começar esta ficha introdutória, devem relembrar o conceito de função aos estudantes, com foco especial no valor de retorno ou a sua inexistência e a diferença entre um e outro tipo, empregando uma ênfase especial no conceito de parâmetro e a necessidade de ser especificado um tipo para ele. Referir igualmente a sua utilidade em termos de criação de uma aplicação, como um todo composto de várias partes, sendo os parâmetros a forma principal de efectuar a comunicação entre os diversos componentes.

Depois, lançar o Pyton Tutor e copiar para a área de código o programa apresentado infra:

1.º Caso Prático

#include <stdio.h>

void fSoma(int parcela1, int parcela2){

int res;

res=parcela1 + parcela2;

printf("\nA operação %d + %d = %d", parcela1, parcela2, res);

parcela1=30;

parcela2=40;

printf("\nparcela1 = %d; parcela2= %d", parcela1, parcela2);

}

int main() {

int x, y;

x=10;

y=20;

// invocação da função

printf("\nx = %d; y = %d", x, y);

fSoma(x, y);

printf("\nx = %d; y = %d", x, y);

return 0;

}

Executar este programa linha a linha e ir perguntando aos estudantes o que está a acontecer no diagrama e na caixa de texto que mostra as mensagens de saída.

Focar especialmente quando a linha de invocação da função é efectuada, tendo como consequência o aparecimento das novas caixas que surgem no diagrama, que representam os parâmetros da função fSoma que foi invocada no main(), parâmetros estes são considerados como variáveis locais na função. Salientar que, como se pode ver nos diagramas dos dados, ao haver a invocação da função, a correspondência entre as variáveis que são os argumentos quando a função é invocada e os parâmetros da função fSoma(), quando esta é carregada no *stack* e inicia a sua execução. Ou seja, o valor de x (o 1.º argumento) vai ser copiado para o parâmetro parcela1 (o 1.º parâmetro) e o valor de y (o 2.º argumento) será copiado para o parâmetro parcela2 (o 2.º parâmetro).

Não esquecer de mostrar que, quando se altera o valor de parcela1 e parcela2, os valor das variáveis que foram copiadas para os parâmetros continuam com o mesmo valor, ou seja, as variáveis x e y são distintas dos parâmetros parcela 1 e parcela 2.

Frisar que, quando a função é invocada, a única operação que é efectuada é a cópia dos valores de x e y para os parâmetros parcela1 e parcela2, sendo estes e pares completamente distintos. Aliás, olhando o diagrama, quando a função está a ser executada, é fácil observar que os endereços das variáveis x e y são distintos dos endereços de parcela1 e parcela2.

Já agora, importa também mostrar que, quando termina a execução da função (fSoma), os blocos que representavam os dados usados na função desaparecem do diagrama, mostrando que, quando a função termina, toda a memória que ela estava a usar é liberta de uma zona de memória denominada *stack*, cuja utilidade se irá explicar um pouco mais abaixo.

2.º Caso Prático

Vai passar agora a mostrar-se execução da mesma função, mas agora usando um valor de retorno.

Para isso, copiar agora o bloco de código apresentado infra e repetir as operações realizadas para o primeiro programa.

#include <stdio.h>

int fSoma(int parcela1, int parcela2){

int res;

res=parcela1 + parcela2;

printf("\nA operação %d + %d = %d", parcela1, parcela2, res);

return res;

}

int main() {

int x, y, valSoma;

x=10;

y=20;

valSoma=0;

// invocação da função

printf("\nx = %d; y = %d", x, y);

valSoma=fSoma(x, y); // atribuição do valor de retorno da função fSoma à variável valSoma

printf("\nx = %d; y = %d; x + y = %d", x, y, valSoma);

x=50;

y=100;

printf("\nx = %d; y = %d; x + y = %d", x, y, fSoma(x, y)); // apresentação do valor de retorno

return 0; // numa mensagem

}

Neste caso, além de se mostrar o resultado da execução bastante semelhante ao programa anterior, salientar apenas que a execução da linha "return res", vai disponibilizar o valor de res à função que invocou a função fSoma(), neste caso o main(), como sendo o resultado da execução da função. Ou seja pode: 1) atribuir-se esse valor a uma variável existente no main() (o 1.º caso da invocação da função); 2) mostrar-se o valor directamente, usando a invocação da função (o 2.º caso da invocação da função). Em suma, o valor de retorno pode usar-se como outro qualquer dado.

3.º Caso Prático

Vai passar agora a mostrar-se execução da mesma função, mas que vai invocar ainda outra função, para mostrar que o processo de invocação pode ser sucessivo e que, quando uma função termina, a função que a invocou passa a ser executada na linha seguinte à invocação feita anteriormente, e assim sucessivamente. Explicar que o que acontece é que as sucessivas funções invocadas e respectivos dados vão sendo carregadas numa área de memória denominada *stack*, umas por cima das outras.

Ora, quando a última termina a execução, a área de memória que usava é libertada e a execução passa à função que estava imediatamente em baixo. Por isso é que a área de memória se chama *stack*, pois que a última função a ser carregada fica no cimo e, quando termina, ela é eliminada (retirada). É como quando temos um prato com panquecas: a última a ser colocada no prato é a primeira a ser retirada e possivelmente comida com muito apetite!

Para mostrar agora as invocação sucessiva das funções, copiar para o Pyton Tutor o código infra:

#include <stdio.h>

int fMult(int factor1, int factor2){

printf("\n\nInício da execução da função fMult...");

int res;

res=factor1 \* factor2;

printf("\nNa função fMult, a operação %d \* %d = %d", factor1, factor2, res);

return res; // o valore de res é devolvido à função fSomaMul que a invocou,

//recebendo esta o valor

}

int fSomaMult(int parcela1, int parcela2, int factMult){

printf("\n\nInício da execução da função fSomaMult...");

int res1, res2;

res1=parcela1 + parcela2;

printf("\nNa função SomaMult, a operação %d + %d = %d", parcela1, parcela2, res1);

res2=fMult(res1, factMult); // invocação da função fMult

printf("\nNa função fSomaMult, depois de invocda a função fMult, a operação %d \* %d = %d", res1, factMult, res2);

return res2; // o valore de res2 é devolvido à função main() que a invocou,

// recebendo esta o valor

}

int main() {

int x, y, valSomaM, factM;

x=10;

y=20;

factM=5;

valSomaM=0;

printf("\nNo main, inicialmente, x = %d; y = %d", x, y);

// invocação da função fSomaMult

valSomaM=fSomaMult(x, y, factM); // atribuição do valor de retorno da função fSomaMult à variável valSomaM

printf("\nNo main(), depois de invocada a função jSomaMult, x = %d; y = %d; (x + y) \* ultiplicador(factM) (%d) = %d", x, y, factM, valSomaM);

x=50;

y=100;

printf("\nNo main(), com x = %d; y = %d; (x + y) \* multiplicador(factM) (%d) = %d",

x, y, factM, fSomaMult(x, y, factM)); // apresentação do valor de retorno numa

// numa mensagem

return 0;

}

Neste caso, o main() invoca a função fSomaMult (aparecendo no diagrama as respectivas variáveis e parâmetros) e, esta, por sua vez, invoca a função fMult (que faz aparecer no diagrama os rectângulos correspondentes às respectivas variáveis e parâmetros). Esta última função mais não faz do que multiplicar o parâmetro factor1 pelo factor2 e devolver à função fSomaMult o valor da variável res e terminar a execução. Quando isso acontece, os rectângulos que correspondiam aos dados da função fMult são eliminados do diagrama, mostrando que a função foi removida do *stack*.

Como poderá ser observado, a execução retorna à função fSomaMult (a que tinha invocado a fMult), na linha seguinte à correspondente à invocação da fMult.