Basic C++

3

Dr. Porkoláb Zoltán Károly

gsd@inf.elte.hu

http://gsd.web.elte.hu

Declarations and definitions

- **Declarations** are introducing names and telling their properties
  - Variables: type, scope, ...
  - Function: signature (how to call it), return type, ...
  - Types: existence, structure, usability
- **Definitions** are declarations sufficient to use the entity
  - Variables: here we allocate the memory
  - Functions: here we define the code of the function body
  - Types: we describe all details, size, internal structure, etc.
- **ODR**: (One Definition Rule)
  - The program must contain **exactly one** definition (...)

Declarations and definitions

- Basic format for declarations/definitions:

```
type   id;                  variable
type  *ptr;                 pointer
type  &ref = var;           reference
type  arr[N];               array of N elements
type  fun( type1 par1, … , typeN parN);      function
```

Definitions

```
void f()
{
    int i; // i is an integer variable, undefined value
    const double pi = 3.14; // pi is a constant double variable
    auto d = pi; // d is a double variable (since pi is double)

    int *ip; // ip is a pointer to an int variable
    const int *cip; // cip is a pointer to a constant integer

    int &ir = i; // ir is a reference to an integer variable

    int t[10];   // t is a 10 element array of integers

    double fahr2cels( double f) { return 5./9.*(f-32); } // function
    void fun() { }  // function with no parameters no return

    enum class color_t : char { red, white, blue }; // enumeration type
    struct complex_t { double re; double im; };   // record/struct type
}
```

Definitions 2

```
void f()
{
    int   i;   // i is an integer variable, undefined value
    int  *ip;  // ip is a pointer to int
    int **ipp; // ipp is a pointer to a pointer to int

    int *&ipr = ip; // ipr is a reference to a pointer to int

    int t[10][20];  // t is a 10 element array of 20 integers

    char *getenv(const char p) {...}  // function returning a pointer




}
```

Definitions 2

```
void f()
{
    int   i;   // i is an integer variable, undefined value
    int  *ip;  // ip is a pointer to int
    int **ipp; // ipp is a pointer to a pointer to int

    int *&ipr = ip; // ipr is a reference to a pointer to int

    int t[10][20];  // t is a 10 element array of 20 integers

    char *getenv(const char p) {...}  // function returning a pointer

    int  *p[10];   // array of pointers or pointer to array of 10 ints?


}
```

Definitions 2

```
void f()
{
    int   i;   // i is an integer variable, undefined value
    int  *ip;  // ip is a pointer to int
    int **ipp; // ipp is a pointer to a pointer to int

    int *&ipr = ip; // ipr is a reference to a pointer to int

    int t[10][20];  // t is a 10 element array of 20 integers

    char *getenv(const char p) {...}  // function returning a pointer

    int  *p[10];   // array of pointers to int
    int (*p2)[10]; // pointer to array of 10 ints


}
```

Definitions 2

```
void f()
{
    int   i;   // i is an integer variable, undefined value
    int  *ip;  // ip is a pointer to int
    int **ipp; // ipp is a pointer to a pointer to int

    int *&ipr = ip; // ipr is a reference to a pointer to int

    int t[10][20];  // t is a 10 element array of 20 integers

    char *getenv(const char p) {...}  // function returning a pointer

    int  *p[10];   // array of pointers to int
    int (*p2)[10]; // pointer to array of 10 ints

    double (*funptr)(double); // pointer to a function double f(double)
}
```

Declarations

```
void f()
{
    extern int i; // i is an integer variable, defined somewhere else
    extern const double pi; // pi is a constant, defined somewhere else

    extern int *ip;  // ip is a pointer to int, defined somewhere else
    extern const int *cip; // cip ptr to const, defined somewhere else

    extern int t[10][20]; // t is array of 10x20, defined somewhere else
    extern int   t[][20]; // t is array of ?x20, defined somewhere else

   extern double fahr2cels( double f); // func, defined somewhere else
   extern double fahr2cels( double);   // func, defined somewhere else

   enum class color_t : char; // enumeration, defined somewhere else
   struct complex_t;   // record/struct type, defined somewhere else
}
```

Declarations

```
void f()
{
    extern int i; // i is an integer variable, defined somewhere else
    extern const double pi; // pi is a constant, defined somewhere else

    extern int *ip;  // ip is a pointer to int, defined somewhere else
    extern const int *cip; // cip ptr to const, defined somewhere else

    extern int t[10][20]; // t is array of 10x20, defined somewhere else
    extern int   t[][20]; // t is array of ?x20, defined somewhere else

    extern double fahr2cels( double f); // func, defined somewhere else
    extern double fahr2cels( double);   // func, defined somewhere else

    enum class color_t : char; // enumeration, defined somewhere else
    struct complex_t;   // record/struct type, defined somewhere else
}
```

SAME

Declarations

```
void f()
{
    extern int i; // i is an integer variable, defined somewhere else
    extern const double pi; // pi is a constant, defined somewhere else

    extern int *ip;  // ip is a pointer to int, defined somewhere else
    extern const int *cip; // cip ptr to const, defined somewhere else

    extern int t[10][20]; // t is array of 10x20, defined somewhere else
    extern int   t[][20]; // t is array of ?x20, defined somewhere else

    extern double fahr2cels( double f); // func, defined somewhere else
    extern double fahr2cels( double);   // func, defined somewhere else

    enum class color_t : char; // enumeration, defined somewhere else
    struct complex_t;   // record/struct type, defined somewhere else
}
```

SAME

SAME

Declarations

```
void f()
{
    extern int i; // i is an integer variable, defined somewhere else
    extern const double pi; // pi is a constant, defined somewhere else

    extern int *ip;  // ip is a pointer to int, defined somewhere else
    extern const int *cip; // cip ptr to const, defined somewhere else

    extern int t[10][20]; // t is array of 10x20, defined somewhere else
    extern int   t[][20]; // t is array of ?x20, defined somewhere else

    extern double fahr2cels( double f); // func, defined somewhere else
    extern double fahr2cels( double);   // func, defined somewhere else

    enum class color_t : char; // enumeration, defined somewhere else
    struct complex_t;   // record/struct type, defined somewhere else
}
```

SAME

SAME

Declarations

```
void f()
{
    extern int i; // i is an integer variable, defined somewhere else
    extern const double pi; // pi is a constant, defined somewhere else

    extern int *ip;  // ip is a pointer to int, defined somewhere else
    extern const int *cip; // cip ptr to const, defined somewhere else

    extern int t[10][20]; // t is array of 10x20, defined somewhere else
    extern int   t[][20]; // t is array of ?x20, defined somewhere else

    extern double fahr2cels( double f); // func, defined somewhere else
    extern double fahr2cels( double);   // func, defined somewhere else

    enum class color_t : char;
    struct complex_t;
}
```

SAME

SAME

Incomplete

One Definition Rule (ODR)

- **ODR**: (One Definition Rule)
  - The program must contain **exactly one** definition (...)

One Definition Rule (ODR)

- **ODR**: (One Definition Rule)
  - The program must contain **exactly one** definition (...)

```cpp
// main.cpp



int i = 42;
extern int answer();

int main()
{
   return answer();
}
```

```cpp
// a.cpp



extern int i;

int meaningOfLife()
{
    return i;
}
```

```cpp
// b.cpp



extern int meaningOfLife();

int answer()
{
    return meaningOfLife();
}
```

One Definition Rule (ODR)

```
// mol.h



extern int i;
extern int meaningOfLife();
extern int answer();
```

```
// main.cpp

#include "mol.h"

int i = 42;
extern int answer();

int main()
{
   return answer();
}
```

```
// a.cpp

#include "mol.h"

extern int i;

int meaningOfLife()
{
    return i;
}
```

```
// b.cpp

#include "mol.h"

extern int meaningOfLife();

int answer()
{
    return meaningOfLife();
}
```

One Definition Rule (ODR)

```
// mol.h

#ifndef MOL_H
#define MOL_H

extern int i;
extern int meaningOfLife();
extern int answer();

#endif // MOL_H
```

```
// main.cpp

#include "mol.h"

int i = 42;
extern int answer();

int main()
{
   return answer();
}
```

```
// a.cpp

#include "mol.h"

extern int i;

int meaningOfLife()
{
    return i;
}
```

```
// b.cpp

#include "mol.h"

extern int meaningOfLife();

int answer()
{
    return meaningOfLife();
}
```

Forward declaration

- Break circular references
- Reduce recompilations

```
// employee.h



struct manager;

struct employee
{
  std::string name;
  manager    *boss;
};
```

```
// manager.h



#include "employee.h"

struct manager
{
  std::string name;
  manager    *boss;
  std::vector<employee *> team;
};
```

Forward declaration

- Break circular references
- Reduce recompilations

```
// employee.h
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

struct manager;

struct employee
{
  std::string name;
  manager    *boss;
};

#endif // EMPLOYEE_H
```

```
// manager.h
#ifndef MANAGER_H
#define MANAGER_H

#include "employee.h"

struct manager
{
  std::string name;
  manager    *boss;
  std::vector<employee *> team;
};

#endif // MANAGER_H
```

Forward declaration

- Break circular references
- Reduce recompilations

```
// employee.h
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

struct manager;

struct employee
{
  std::string name;
  manager    *boss;
};

#endif // EMPLOYEE_H
```

```
// manager.h
#ifndef MANAGER_H
#define MANAGER_H

#include "employee.h"

struct manager
{
  std::string name;
  manager    *boss;
  std::vector<employee *> team;
};

#endif // MANAGER_H
```

Initialization

- Variables can be initialized when defined
- Static lifetime variables are zero initialized by default
- References and (most of the) constants must be initialized

```
int   i = 42;
int *ip = &i;   // ip "points to" i
int &ir = ip;   // ir and i are synonyms

int arr1[10] = { 0,1,2,3,4,5,6,7,8,9 };
int arr2[10] = { 0,1,2,3,4,5,6,7,8 };  // arr2[9] is 0
int arr3[]   = { 0,1,2,3,4,5,6,7,8 };  // int arr3[9]

std::vector<int> v = { 0,1,2,3,4,5,6,7,8,9 };
int arr4[10] = { 0,1,2,3,4,5,6,7,8,9,10 };  // Compile error

char welcome1[] = { 'H','e','l','l','o','\0' };
char welcome2[] = "Hello";

extern double sin(double);         // declaration
double (*funcptr)(double) = sin;  // funcptr "points to" sin()
```

Type synonyms

- Creates alias names not new types
- Can simplify syntactical mess

```
typedef int length_t;  // before C++11
using length_t = int;  // since C++11

using trigfp_t = double (*)(double); // can point to e.g. sin()
using trigfp_t = double    (double); // Same as above


trigfp_t inverse( trigfp_t fun)
{
    static std::map<trigfp_t,trigfp_t> table = {
       {sin, asin}, {cos, acos}, {tan, atan} // etc...
                                            };
    return table[map];
}
```

Traps

```
int  *ptr;
int*  ptr;    // same as above

int  *p1, *p2;  // int *p1; int *p2;
int*  p1, p2;   // int *p1; int  p2;


int t[10][20][30];  // definition or declaration

int t[][20][30];    // declaration
int t[][][30];      // invalid
int t[][][];        // invalid


using length_t = double;

double   f(double);
length_t f(length_t);  // same as above, do not overload
```

Traps

```
int  *ptr;
int*  ptr;    // same as above

int  *p1, *p2;  // int *p1; int *p2;
int*  p1, p2;   // int *p1; int  p2;


int t[10][20][30];  // definition or declaration

int t[][20][30];    // declaration
int t[][][30];      // invalid
int t[][][];        // invalid


using length_t = double;

double   f(double);
length_t f(length_t);  // same as above, do not overload
```

Traps 2

```
void g(int par[10][20][30]);

void f()
{
    int t[10][20][30];  // sizeof(t) == 6000*sizeof(int)
    g(t);
}

void g(int par[10][20][30])
{
    // sizeof(par) ==
}
```

Traps 2

```
void g(int par[10][20][30]);

void f()
{
    int t[10][20][30];  // sizeof(t) == 6000*sizeof(int)
    g(t);
}

void g(int par[10][20][30])
{
    // sizeof(par) == sizeof(int (*)[20][30]) == sizeof(int*)
}

void g(int par[][20][30])
{
    // sizeof(par) == sizeof(int (*)[20][30]) == sizeof(int*)
}

void g(int (*par)[20][30])
{
    // sizeof(par) == sizeof(int (*)[20][30]) == sizeof(int*)
}
```

Use std::array

```
#include <array>

int main()
{
    // construction uses aggregate initialization
    std::array<int, 3> a1 = {1, 2, 3};  // array of 3 ints
    std::array<std::string, 2> a2 = {std::string{"a"}, "b"};
    std::array<int, 3> a3{1, 2, 3}; // double-braces required before C++14

    // container operations are supported
    std::sort(a1.begin(), a1.end());
    std::cout << a1.size() << " " << a1[1];

    // ranged for loop is supported
    for(const auto& s: a3)
        std::cout << s << ' ';

    std::array  a1{"foo"};  // C++17 CTAD: std::array<const char*,1>{"foo"}
    auto a2 = std::to_array{"foo"}; // std::array<char,4>{'f','o','o','\0'};
}
```

Scope and life

- Scope: static property

  The area in the source where a name has a specific meaning

- Life: dynamic property

  The time under running the program when an entity (specifically a memory area) is valid

Scope

- Each name is only visible in some portion of the program
  - Block/Local scope
  - Namespace/Global scope
  - (Class scope)
  - ~~Enum scope~~
  - ~~Template parameter scope~~

Local scope

```
void f()
{
    int i = 42;      // scope of outer i starts here
    ++i;             // outer i
    ++k;             // k is not in scope: error
    {
        int k = i;   // scope of k starts here
        ++i;         // outer i
        ++k;         // k
        int i = k;   // scope of inner i starts here, hides outer i
        ++i;         // inner i
    }    // scope of (inner) i and k ends here
    ++i;  // outer i
    ++k;  // k is not in scope: error
}  // scope of (outer) i ends here
```

Local scope

```
void f()
{
    int i = 42;      // scope of outer i starts here
    ++i;             // outer i
    ++k;             // k is not in scope: error
    {
        int k = i;   // scope of k starts here
        ++i;         // outer i
        ++k;         // k
        int i = k;   // scope of inner i starts here, hides outer i
        ++i;         // inner i
    }    // scope of (inner) i and k ends here
    ++i;  // outer i
    ++k;  // k is not in scope: error
}  // scope of (outer) i ends here
```

Function parameters

```
void f(int k)        // scope of parameter k starts here
{
    int i = 42;      // scope of outer i starts here
    ++i;             // outer i
    ++k;             // parameter k
    {
        int k = i;   // scope of k starts here, hides parameter k
        ++i;         // outer i
        ++k;         // inner k
        int i = k;   // scope of inner i starts here
        ++i;         // inner i
    }    // scope of (inner) i and (inner) k ends here
    ++i;  // outer i
    ++k;  // parameter k
}  // scope of (outer) i and parameter k ends here
```

Function parameters 2

```
void f(int k)        // scope of parameter k starts here
try
{
    int i = 42;      // scope of outer i starts here
    ++i;             // outer i
    ++k;             // parameter k
    {
        int k = i;   // scope of k starts here, hides parameter k
        ++i;         // outer i
        ++k;         // inner k
        int i = k;   // scope of inner i starts here
        ++i;         // inner i
    }    // scope of (inner) i and (inner) k ends here
    ++i;  // outer i
    ++k;  // parameter k
}  // scope of (outer) i ends here
catch (std::exception& e)
{
  ++k;  // parameter k
}  // scope of parameter k ends here
```

Local scope 2

```
void f()
{
    int i = 42;

    {
        extern void g();  // scope of g() starts here
        ++i;
        g();  // call of g()
    } // scope of g() declaration ends here

    g();  // g() is not in scope: error
}
```

Namespace/global scope

```
int i;          // definition: global with external linkage, init. 0
extern int j;  // declaration: global, defined in other TU, init. 0
static int k;  // definition: global, no linkage, init. 0

namespace N
{

  int i;                    // N::i, init. 0
  void f() { return k+i; } // N::f()   k+N::i
}

void f()
{
    ++i;              // global i
    int i = 42;       // local i hides global i
    ++k;              // global k
    ++i;              // local i
    ++N::i;           // i from namespace N
    ++::i;            // global i
    ++::k;            // global k
}  // scope of local i ends here
```

Namespace/global scope

```
int i;          // definition: global with external linkage, init. 0
extern int j;  // declaration: global, defined in other TU, init. 0
static int k;  // definition: global, no linkage, init. 0

namespace N
{
  void g();                 // N::g()
  int i;                    // N::i, init. 0
  void f() { return k+g(); } // N::f()  N::g()  k+N::i
}

void f()
{
    ++i;              // global i
    int i = 42;       // local i hides global i
    ++k;              // global k
    ++i;              // local i
    ++N::i;           // i from namespace N
    ++::i;            // global i
    ++::k;            // global k
}  // scope of local i ends here

int N::g() { return i; }   // N::g()  N::i
```

Anonymous namespace

```
int i;          // definition: global with external linkage
extern int j;   // declaration: global, defined in other TU
static int k;   // definition: global, no linkage

namespace
{
    int k; // k is specific to the translation unit, visible as ::k
}

void f()
{
    ++k;  // k from anonymous namespace is visible as global
    {
        int k = 42; // local k hides anonymous namespece k
        ++k;        // local k
        ++::k;      // global k from anonymous namespace
    }  // scope of local k ends here
}
```

Namespaces can be nested, extended

```
namespace N
{
    int n;          // N::n
    namespace M
    {
        int m;      // N::M::m
    }
}

namespace N::M      // scope of N::M resumes  // since C++17
{
    int l;   // N::M::l
    int g() { return n+m; }   // N::M:g()   return N::n+N::M::m
}

void f()
{
    N::n = 1;
    N::M::m = 2;
    N::M::l = 3;
    N::n = N::M::g();
}
```

Class scope

```
class date
{
public:
    int get_year()  const { return year_; }
    int get_month() const { return month_; }
    int get_day()   const { return day_; }
    void set( int y, int m, int d);
private:
    int year_;
    int month;
    int day_;
    void check( int y, int m, int d);
};

void date::set( int y, int m, int d) // scope of date resumes
{
    check( y, m, d); // date::check()
    year_  = y;      // date::year_
    month_ = m;      // date::month_
    day_   = d;      // date::day_
}
```

Using namespace

```
namespace K
{
    int k;
}
namespace N
{
    int n;
    int m;
}
namespace L
{
    int k;
}
void f()
{
    using namespace K;  // all names from K are visible
    ++k;        // K::k
    using N::n;         // n from N is visible
    ++n;        // ok, N::n
    ++m;        // error N::m is not visible
    ++N::m;     // ok, N::m
    using namespace L;  // all names from L are visible
    ++k;        // error: ambigous, K::k or L::k?
    n = K::k + L::k; // fine
}
```

Scope tips

- Minimize visibility to the necessary minimum

```
int i;

void f()
{
    int j = 1;
    ...
    ++i; // instead of ++j


    while (i<10) { ... }
}
```

Scope tips

- Minimize visibility to the necessary minimum

```
int i;

void f()
{
    int j = 1;
    ...
    ++i; // i is not visible here: compile error

    int i = 0;
    while (i<10) { ... }
}
```

Scope tips

- Minimize visibility to the necessary minimum

```
int i;

void f()
{
    int j = 1;
    ...
    ++i; // compile error

    for (int i = 0; i<10; ++i) { ... }
    // i is not visible here
}
```

Scope tips

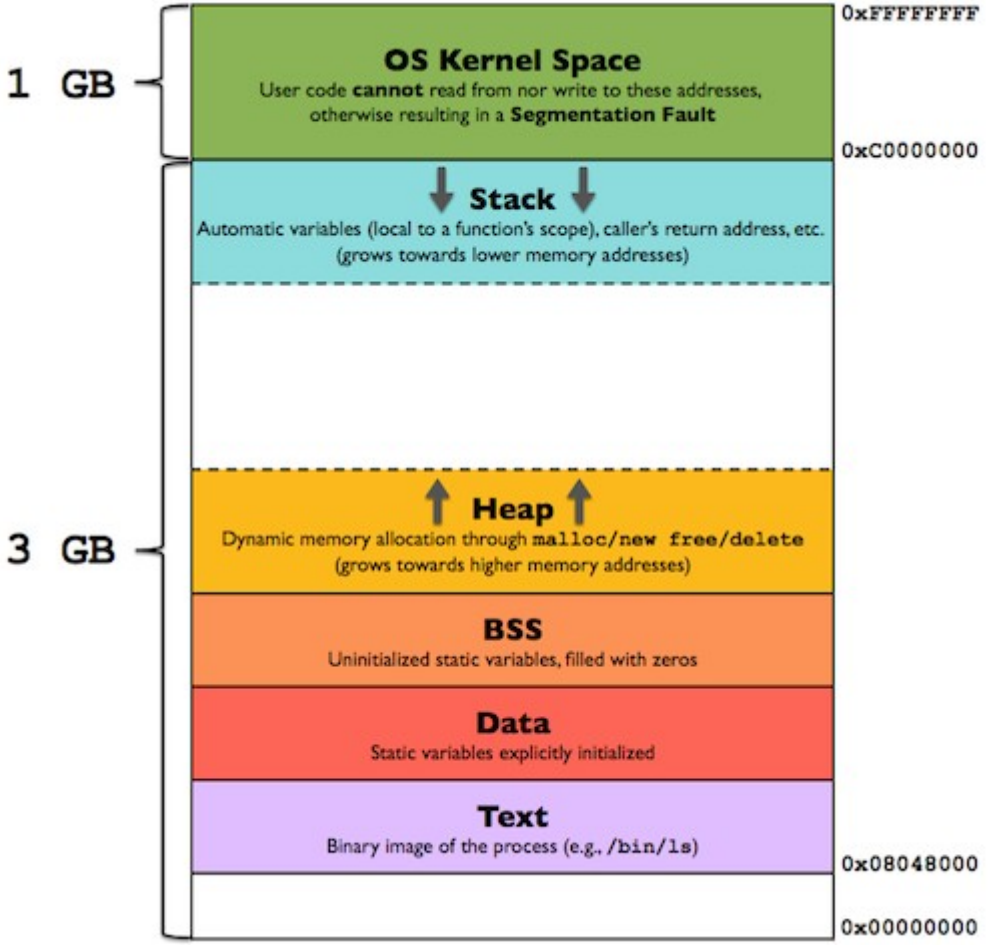
- Never use using namespace in header files
  - Because its effect is intrusive to the files include it

Life

- Each C++ object is valid only for a specific “lifetime” under run
  - Read only objects
  - Static lifetime
  - Automatic lifetime
  - Temporary lifetime
  - Dynamic lifetime

Memory model

- Different platform dependent memory models exist
- Most popular on UNIX: ELF (Executable and Linkable Format)



Read only objects

- String literals
- May placed onto physically read-only memory

```
const char *hello1 = "Hello world";
const char *hello2 = "Hello world";
// ...
hello1[1] = 'a'; // syntax error!

char s = const_cast<char>(hello1);
s[1] = 'a';      // can be run-time error!

if ( hello1 == hello2 )  // likely true
...
```

Static lifetime

- Allocated at the beginning of the run of the program and when the initialization (e.g. by its constructor completed)
- Remains live until the end of the program
- Inside a source file construction is ordered
- Destruction is in reverse order of construction
- Not ordered between source files
- Typical for
  - Global/namespace variables
  - Local static variables
  - Static class members

Static life

```
const int bufsize = 1024;
char buffer[bufsize];

struct X
{
    int i = 1;
    static int si;
};

void main()
{
    X::si = 42; // static data member already lives
    {
        X x;   // all the non-static data member lives from now
        x.i = bufsize;
        x.si = buffer[0];
    }  // non-static data member lifetime ends
    ++X::si;  // static data member still lives
}
// X::si, buffer, bufsize lives end
```

Local static variables

```
int f()
{
    static int cnt = 0;     // live starts when first executed
                            // intialized only once
    return ++cnt;
}


int main()
{
    for ( int i = 0; i < 10; ++i)
    {
        f();    // prints 1 2 3 4 5 6 7 8 9 10
    }
    ++cnt;  // compile error: cnt lives here, but not visible
}
// cnt live ends here
```

Local static variables

```
int f()
{
    if ( some_condition() )
    {
        static int cnt = 0;     // live starts when first executed
                                // intialized only once
        return ++cnt;
    }
    else
    {
        return 0;
    }
}

int main()
{
    for ( int i = 0; i < 10; ++i)
    {
        std::cout << f();     // prints 1 2 3 4 5 6 7 8 9 10 or 0
    }
}
// cnt live ends here if ever started
```

Automatic lifetime

- Begins when the execution reaches the definition in a block
- Ends when the execution leaves the block (and loose value)
- Placed in the execution stack as well as the parameters
- Typical for
  - Objects local to a block (and non-static)
- Lifetime is valid in inner blocks and function calls
  - but the name may be hidden
- Lifetime is not valid when the block finished
  - pointers and references referring to **MUST NOT** be used

Automatic lifetime

```
int *f()
{
    int i = 0; // life of i starts here, initialized to 0 every time

    ++i;
    std::cout << i;  // alwas prints 1

    return &i;  // forbidden and dangerous

} // life of i ends here

int main()
{
    for ( int i = 0; i < 10; ++i)
    {
        int* ptr = f();    // invalid use of address of i
        std::cout << *ptr; // likely run-time error
    }
}
```

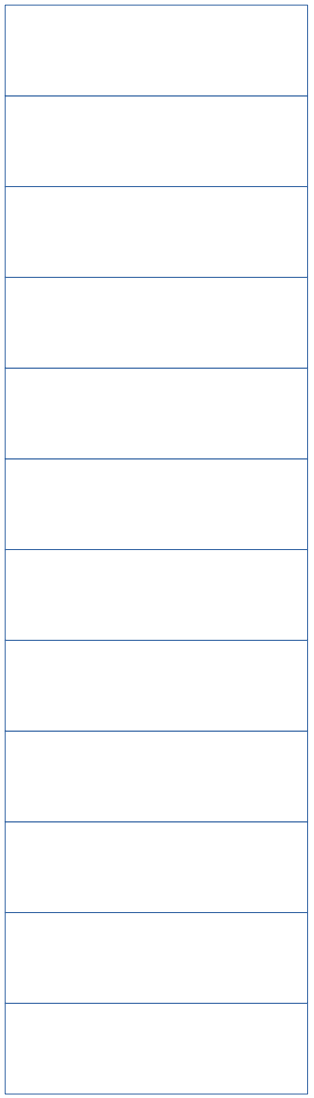
Automatic lifetime

```
int *f()
{
    int i; // life of i starts here, uninitialized

    ++i;   // undefined behavior, using unitialized variable
    std::cout << i;  // always prints ?

    return &i;  // forbidden and dangerous

} // life of i ends here

int main()
{
    for ( int i = 0; i < 10; ++i)
    {
        int* ptr = f();    // invalid use of address of i
        std::cout << *ptr; // likely run-time error
    }
}
```

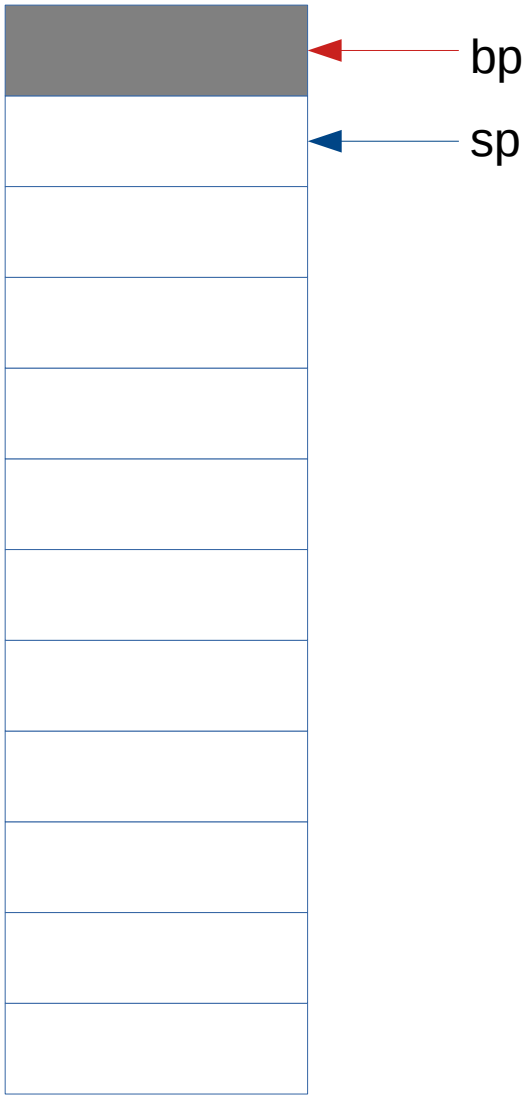
Activation stack

```
void f()
{
    int i;
    double d;
    i = 42;
    d = i;
    d = g( i, d);
    ++i;
}

double g( int par1, double par2)
{
    double t[2]
    t[0] = par1;
    t[1] = par2;
    par1++;
    return t[0]
}
```

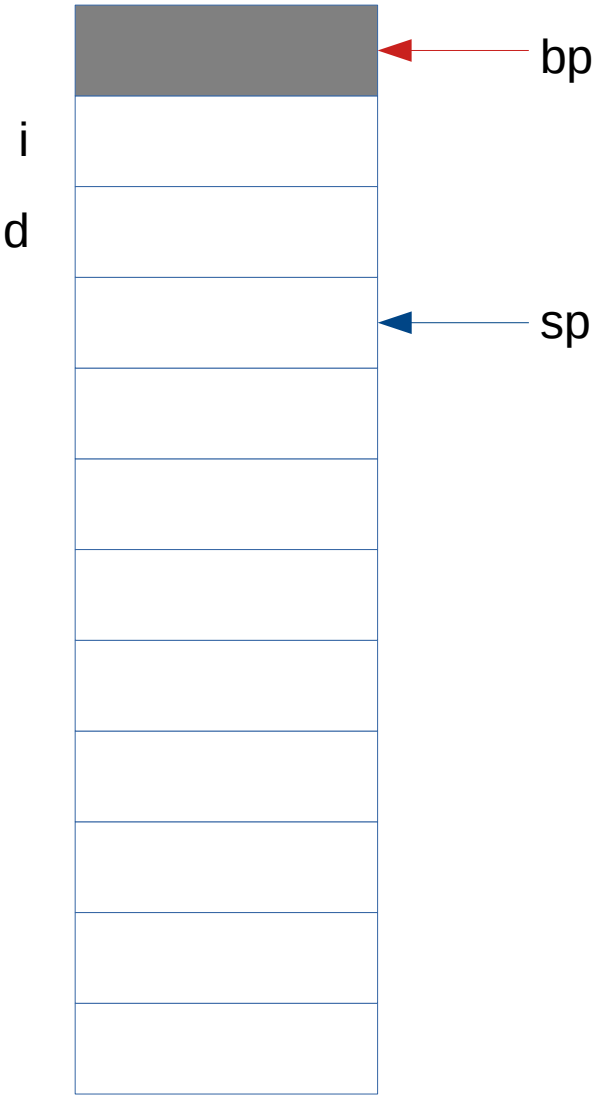
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



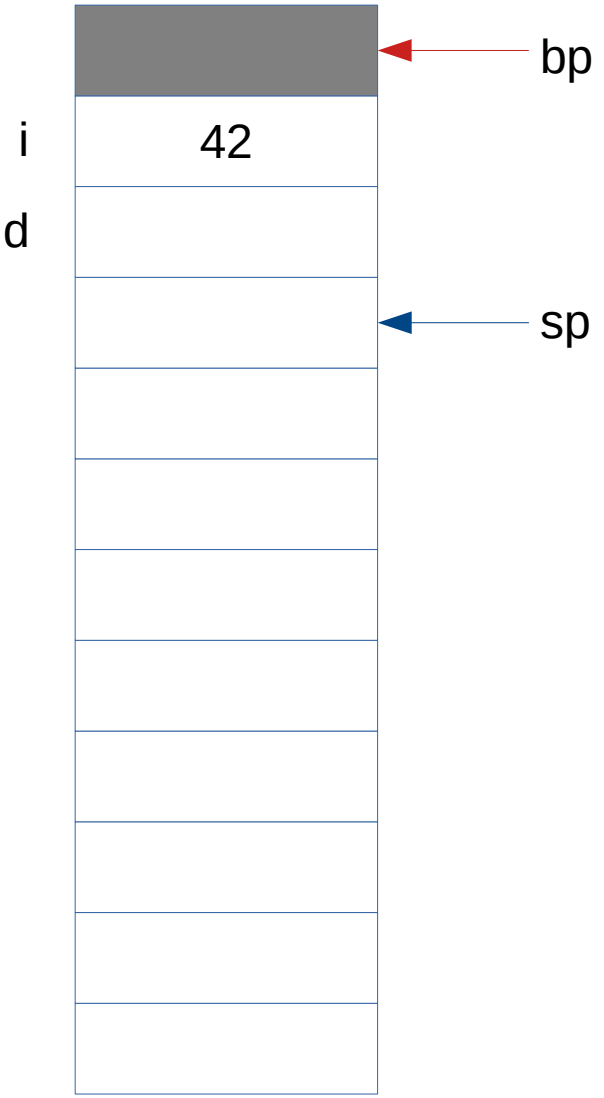
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



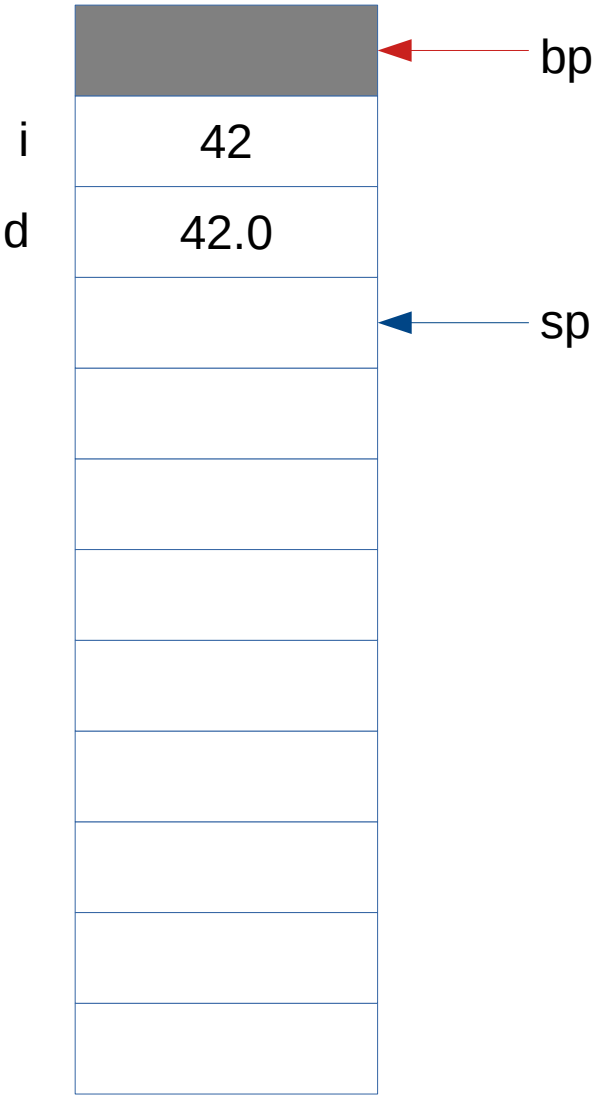
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



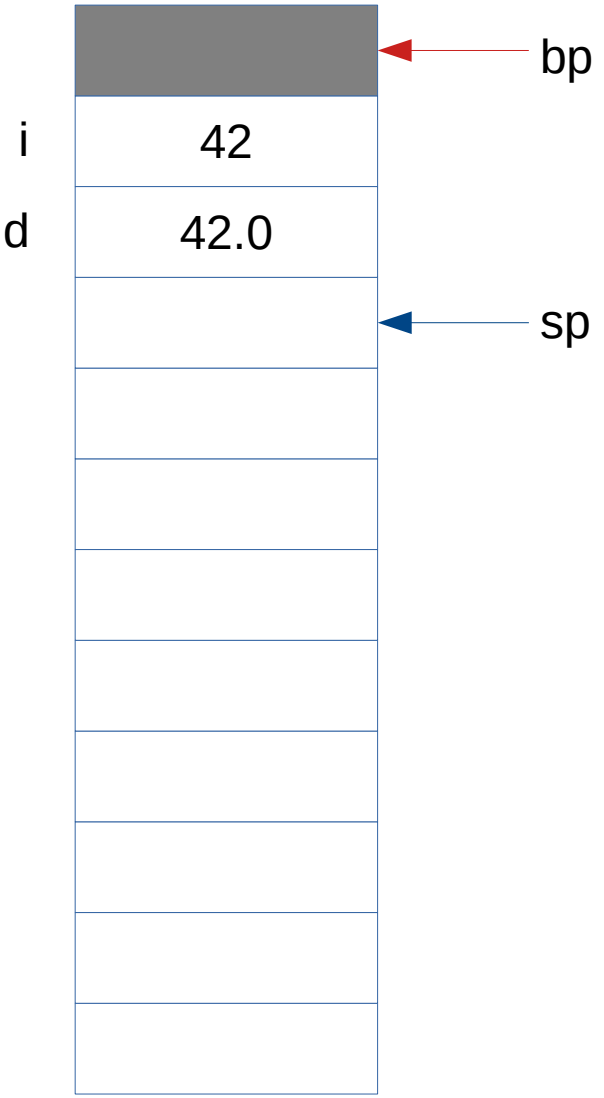
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



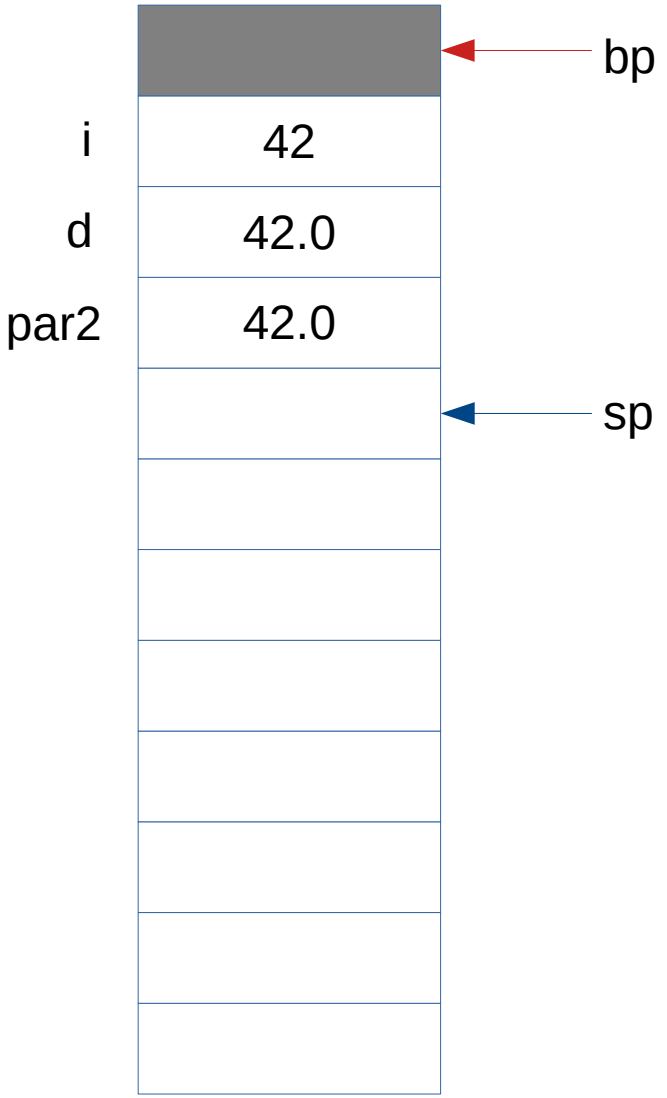
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



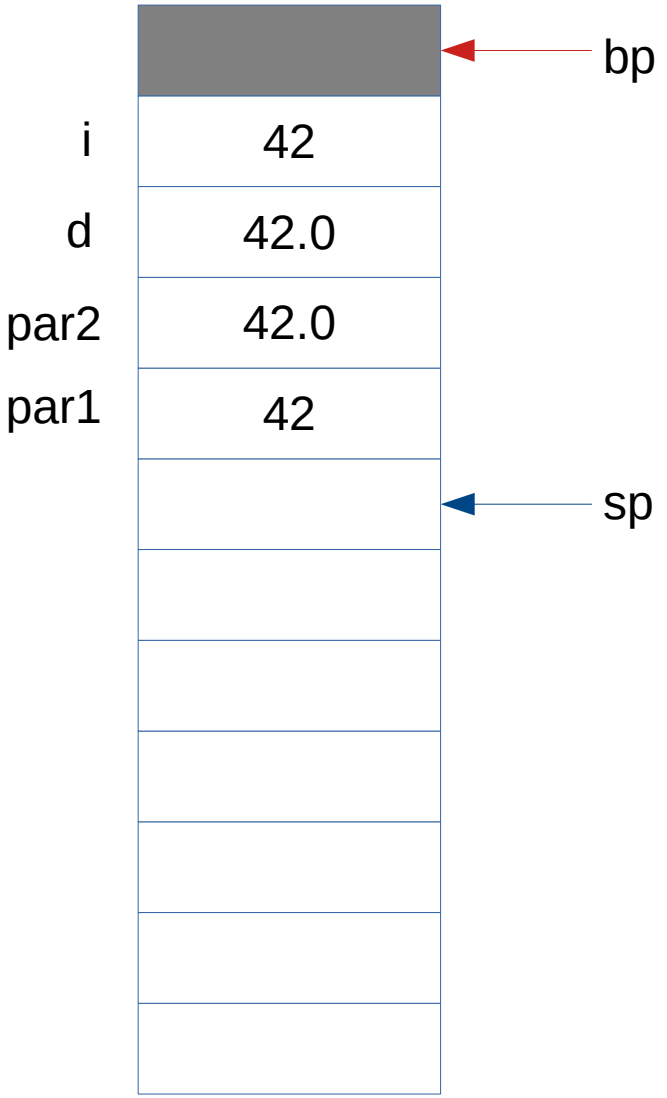
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



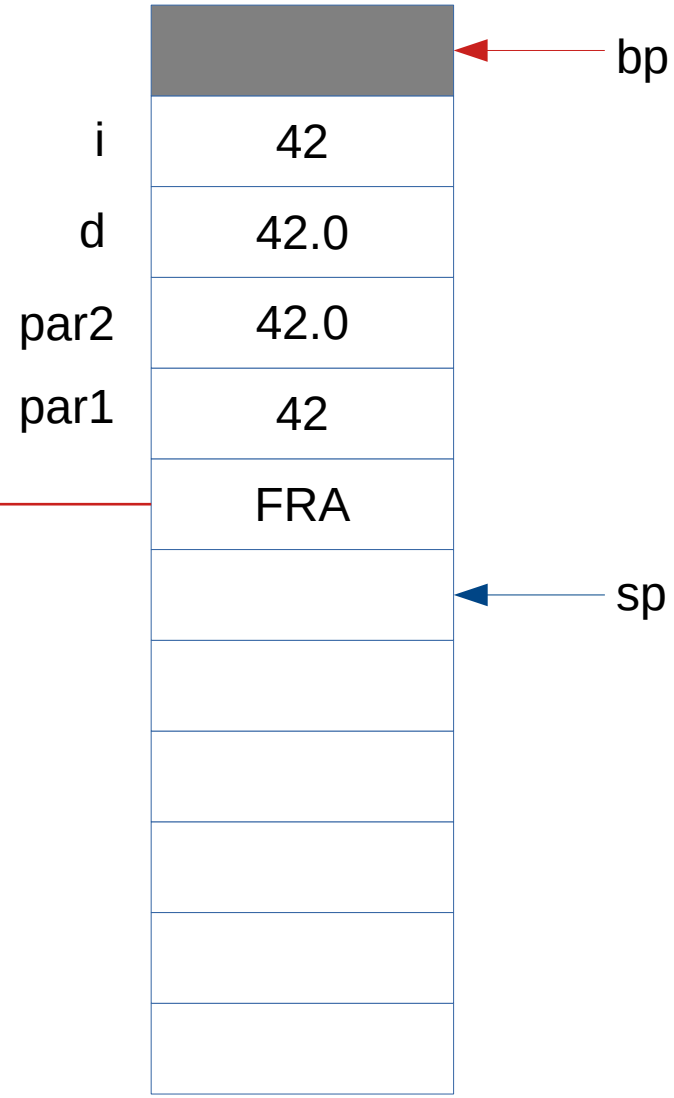
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



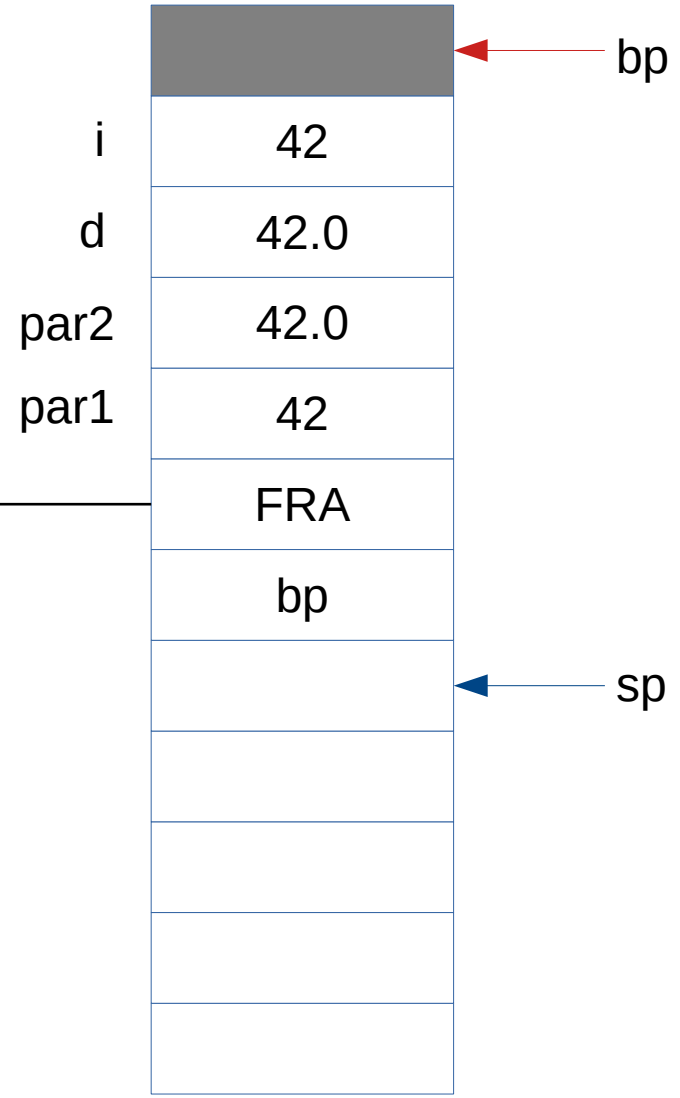
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



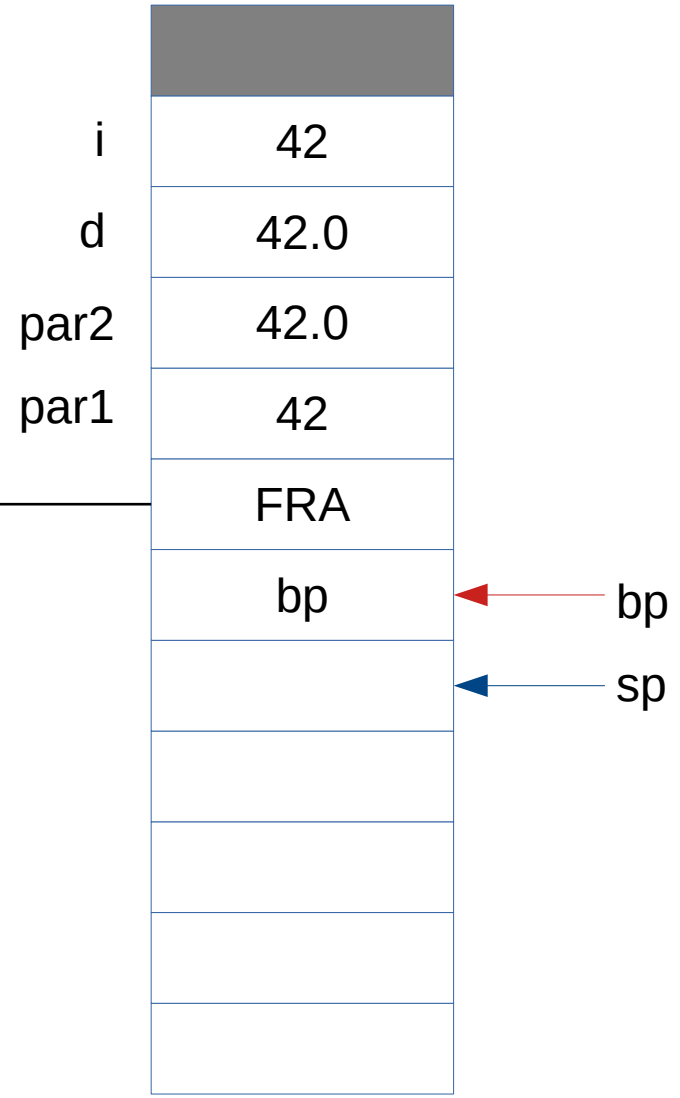
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



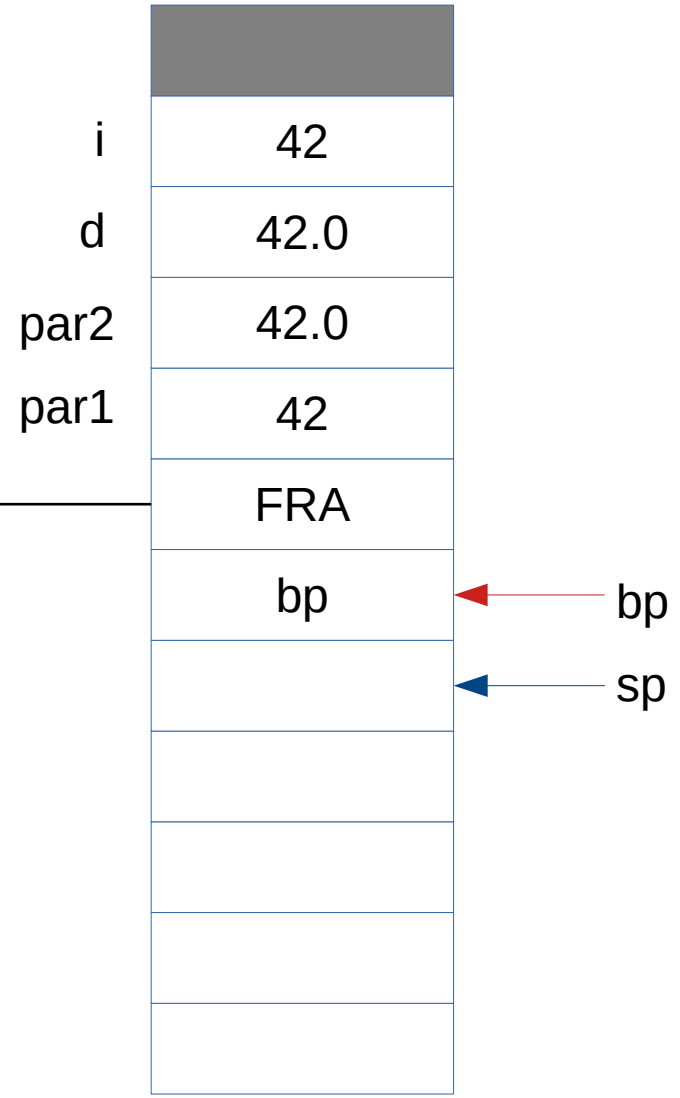
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



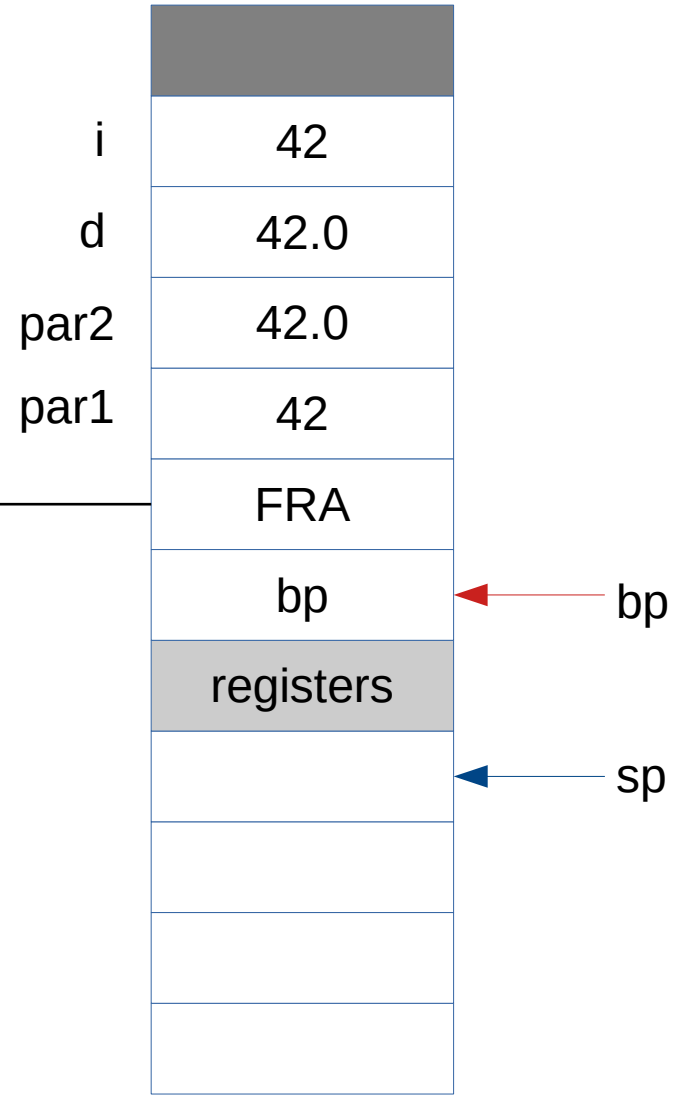
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



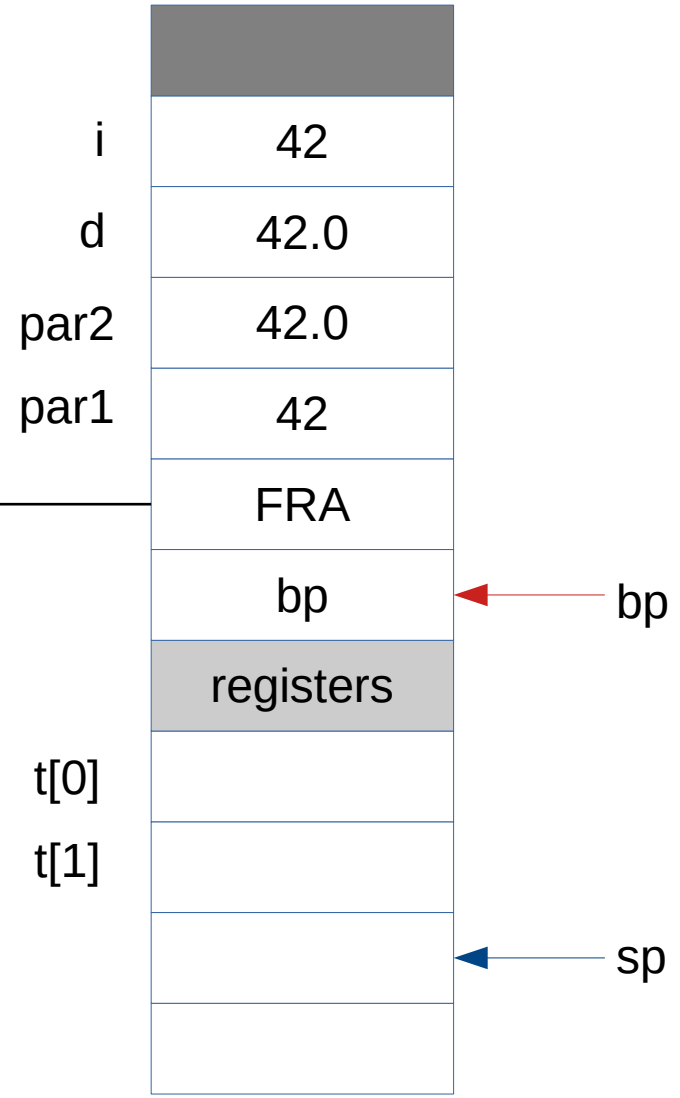
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



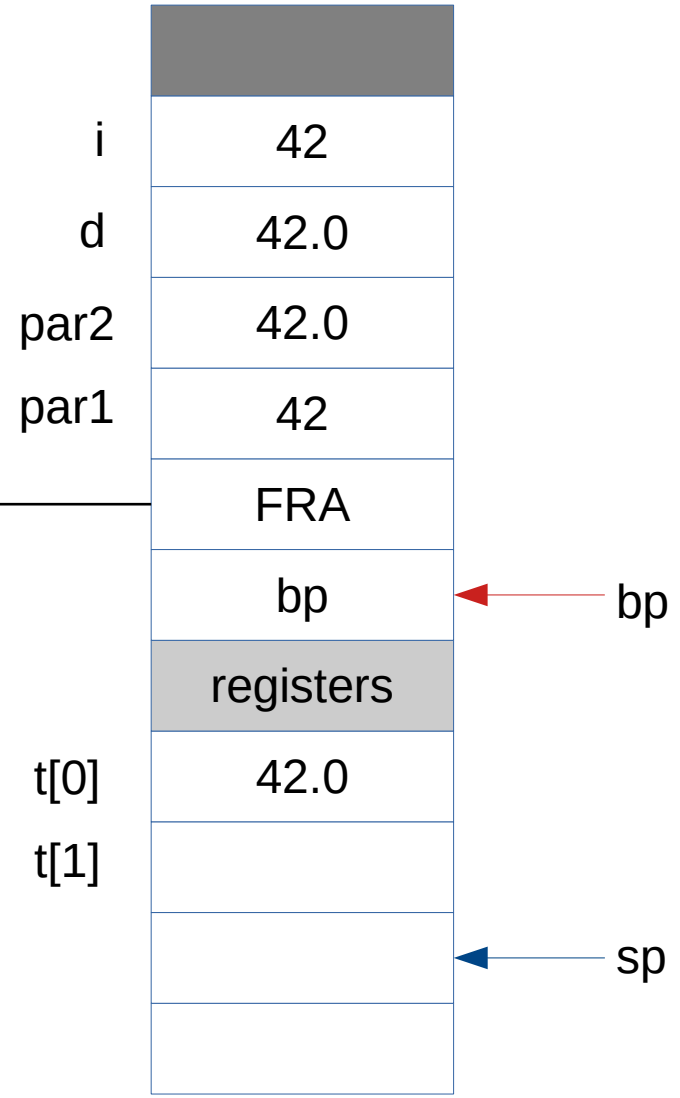
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



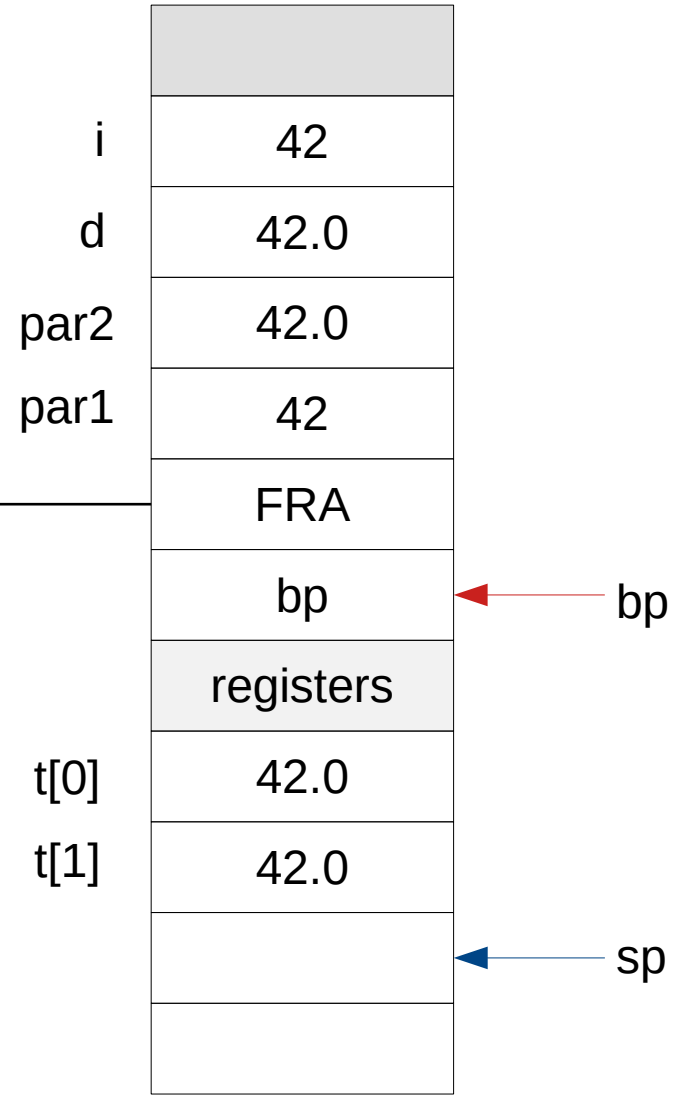
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



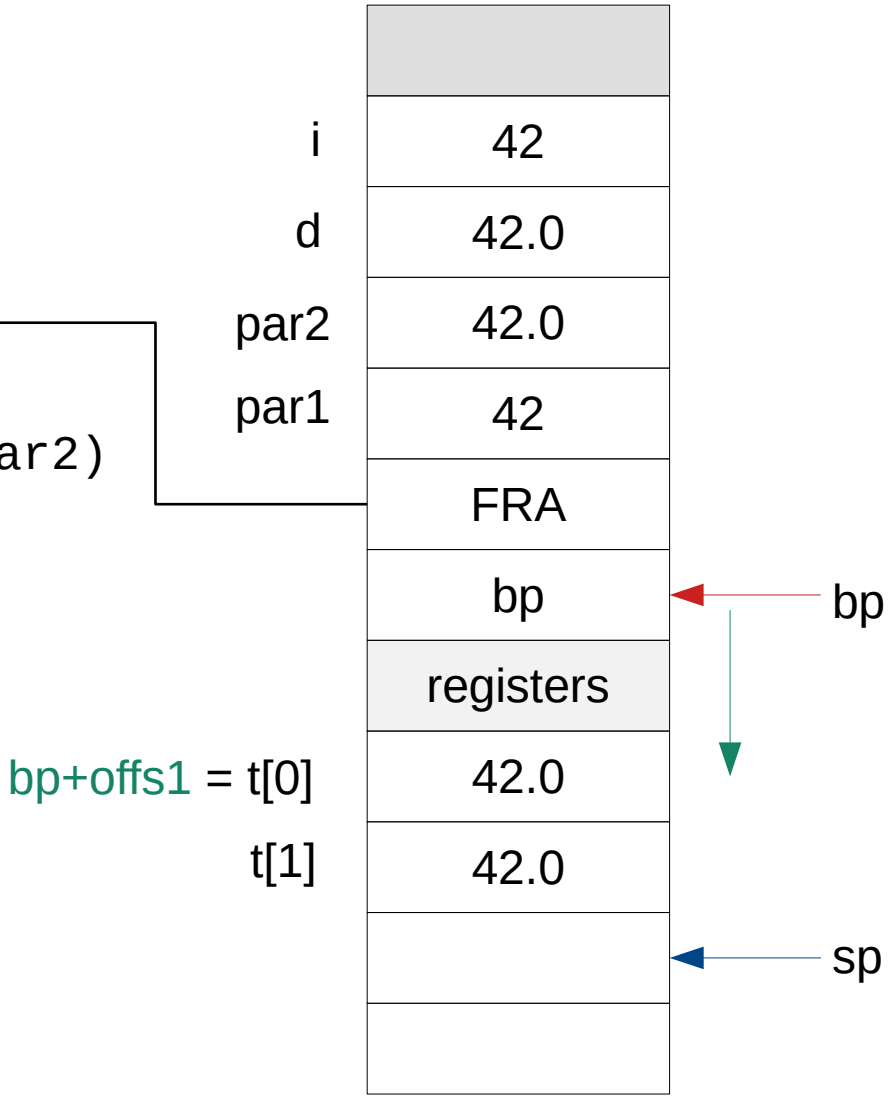
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



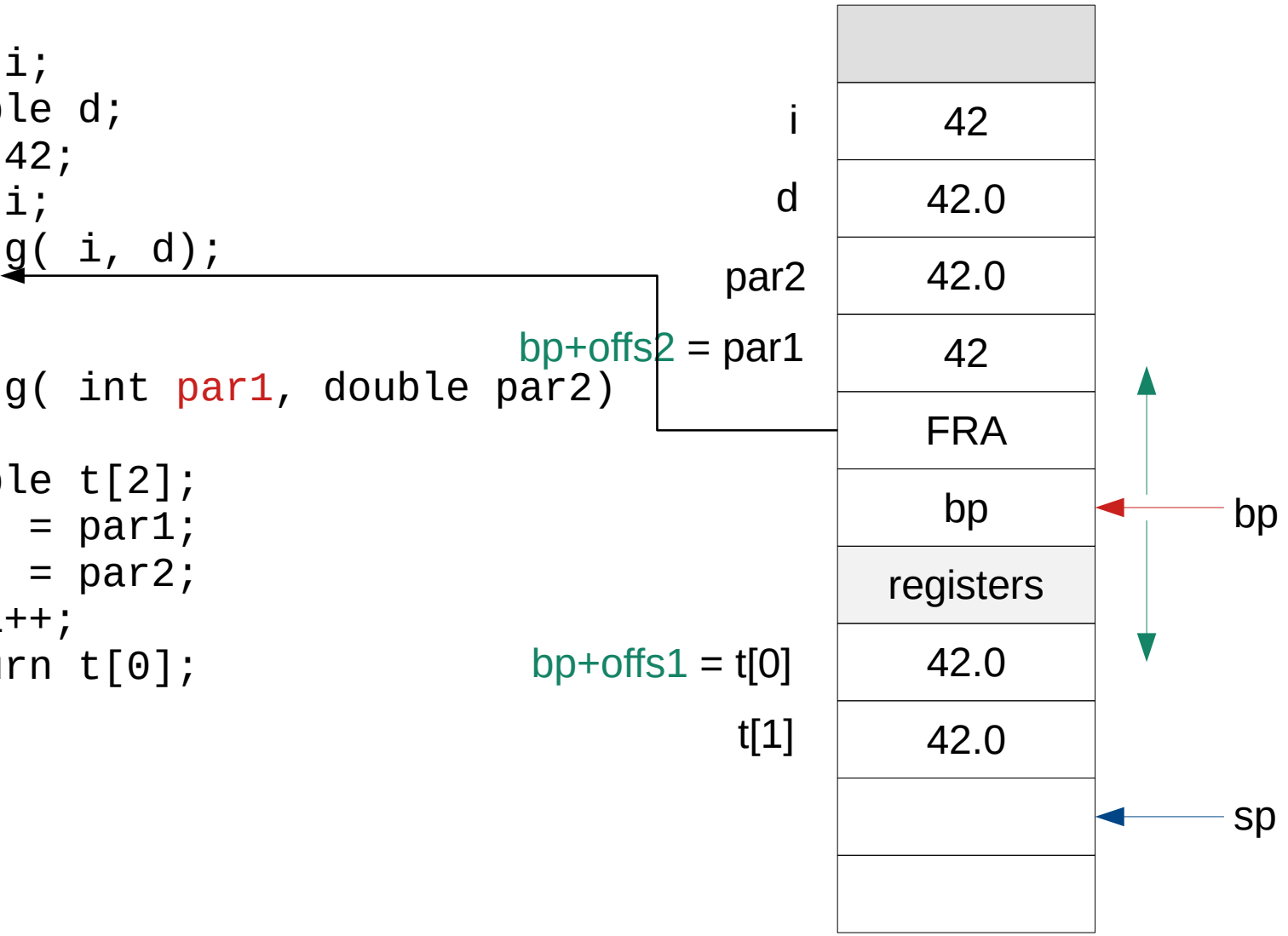
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



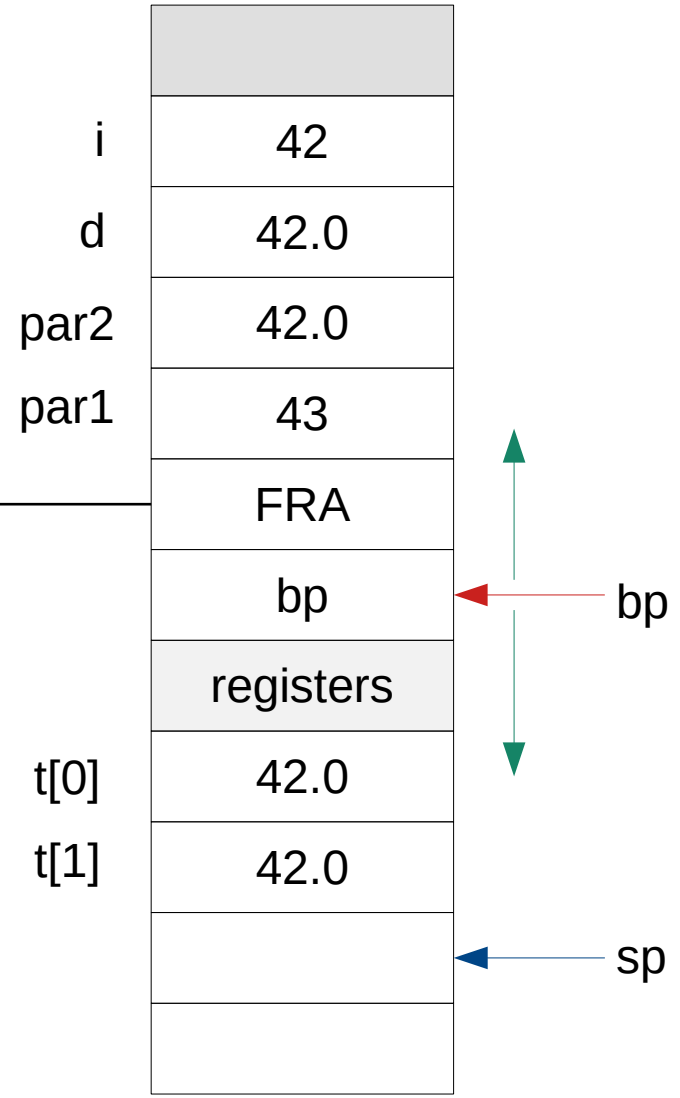
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



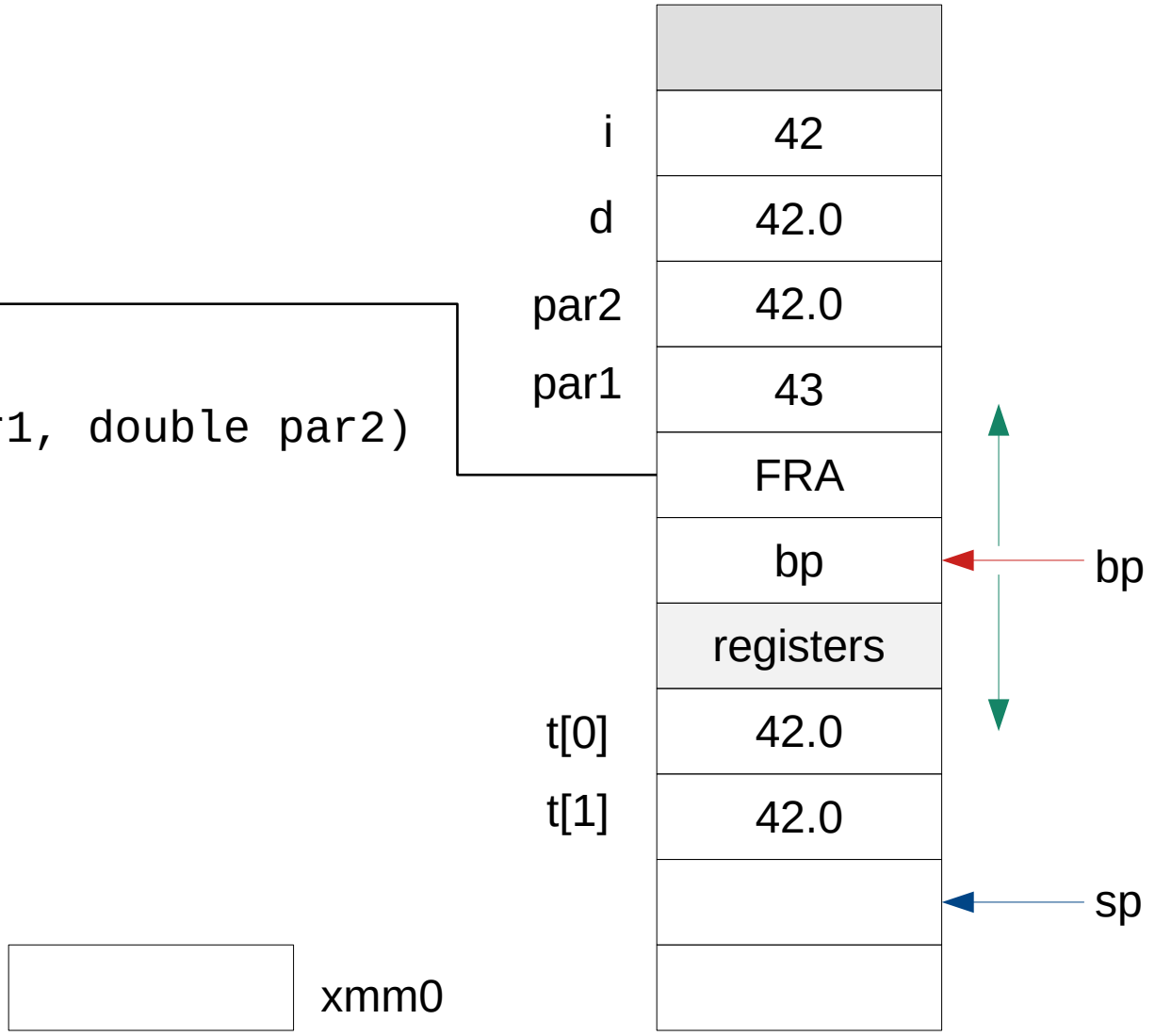
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



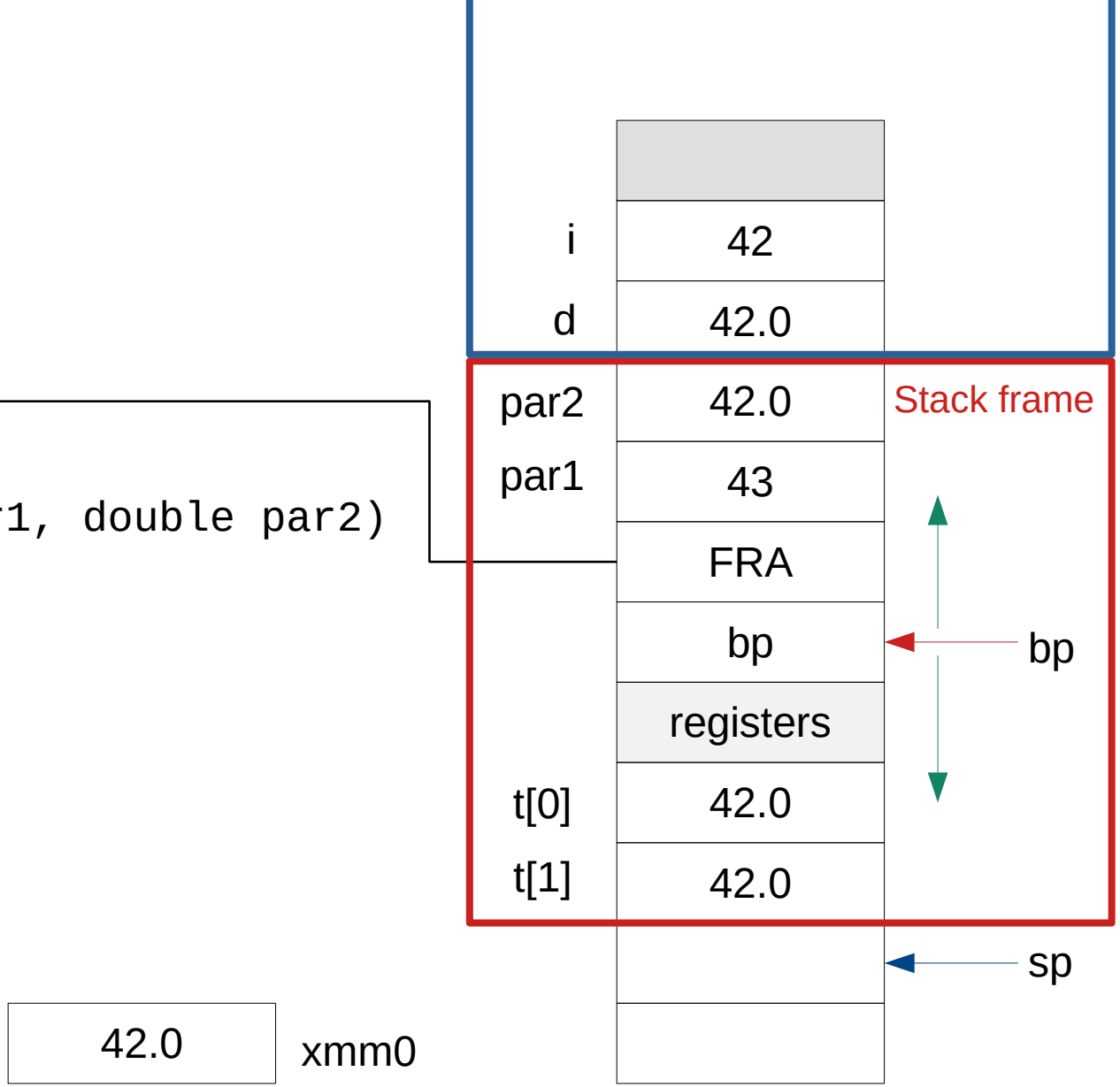
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



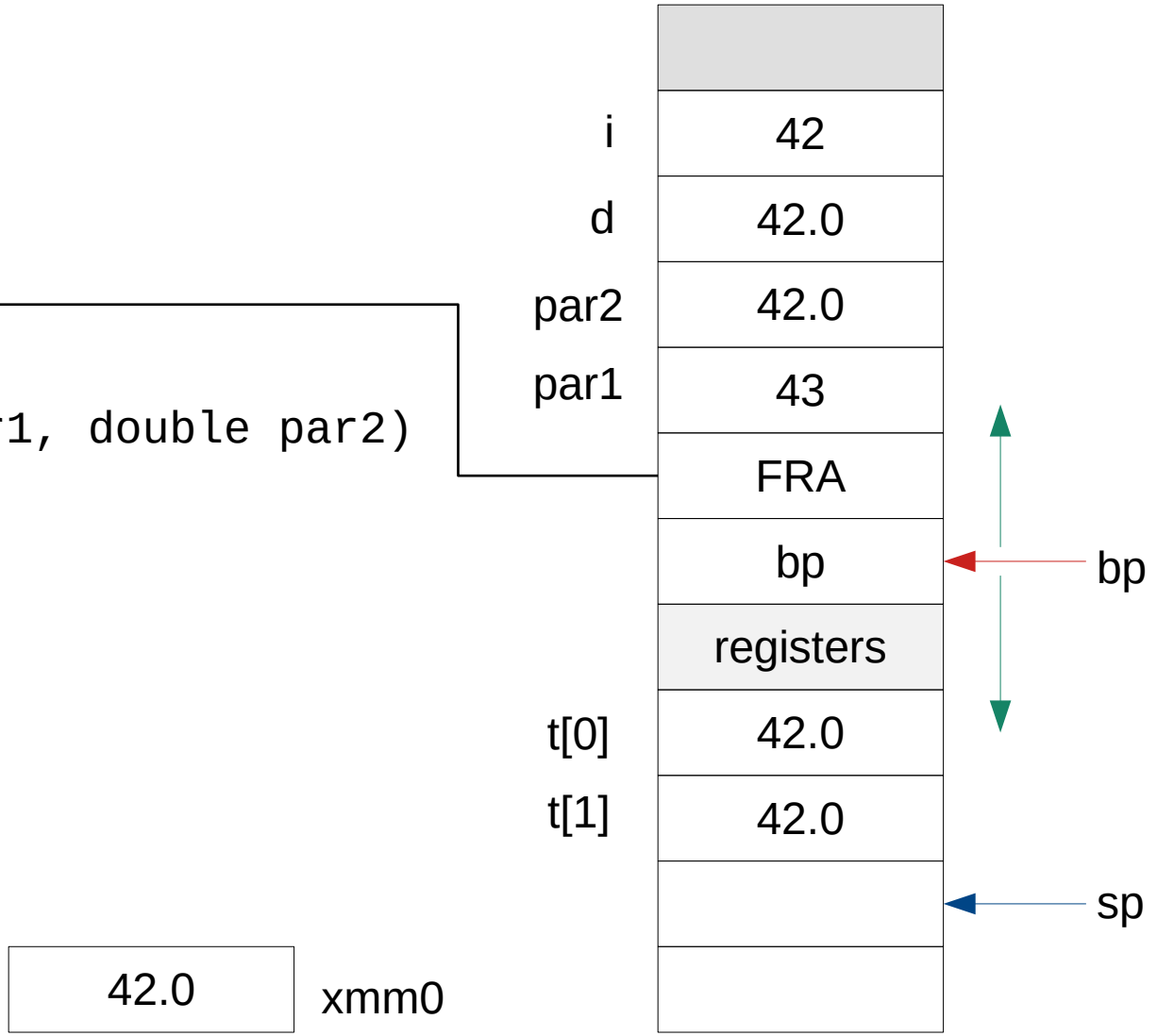
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



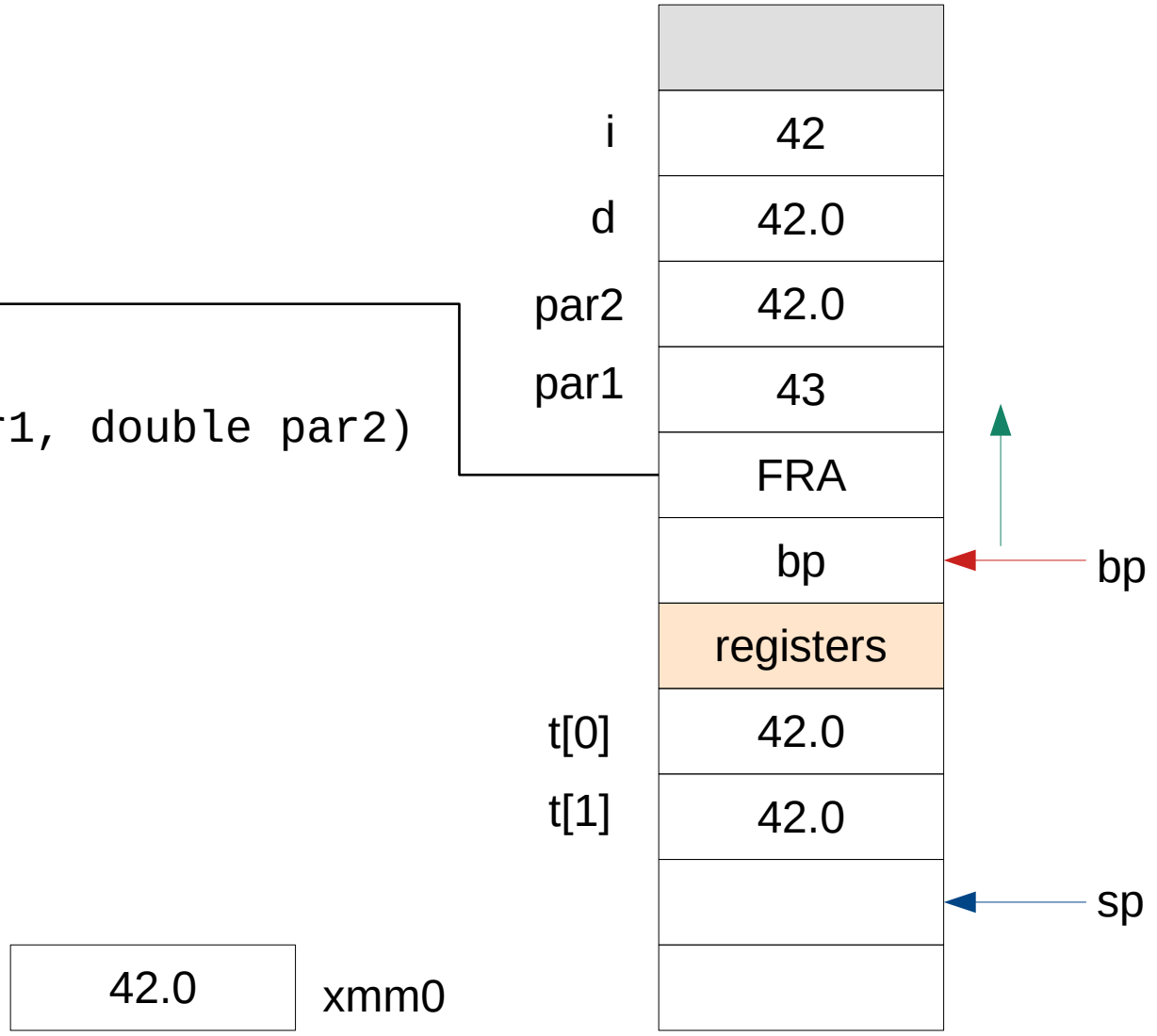
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



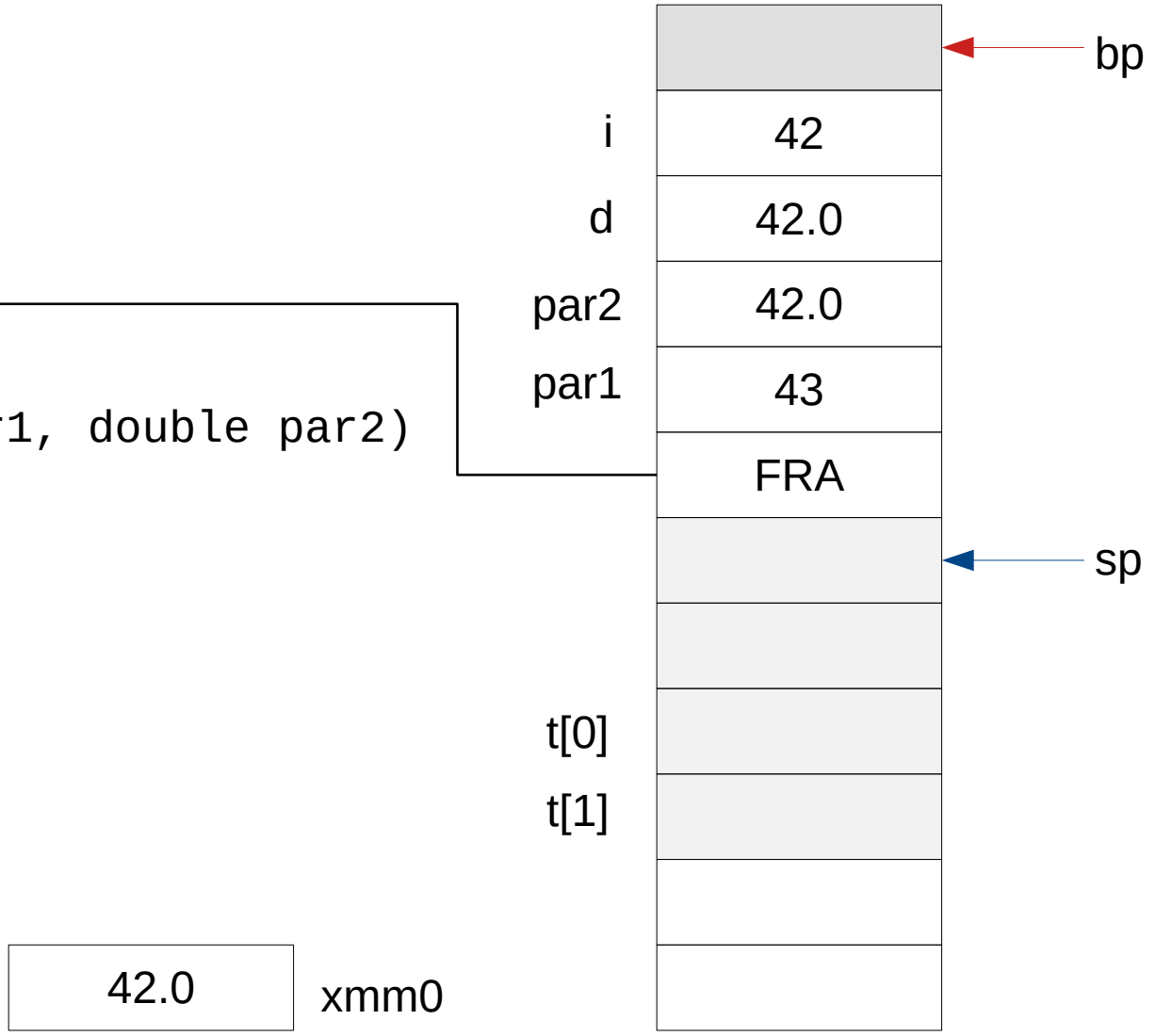
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



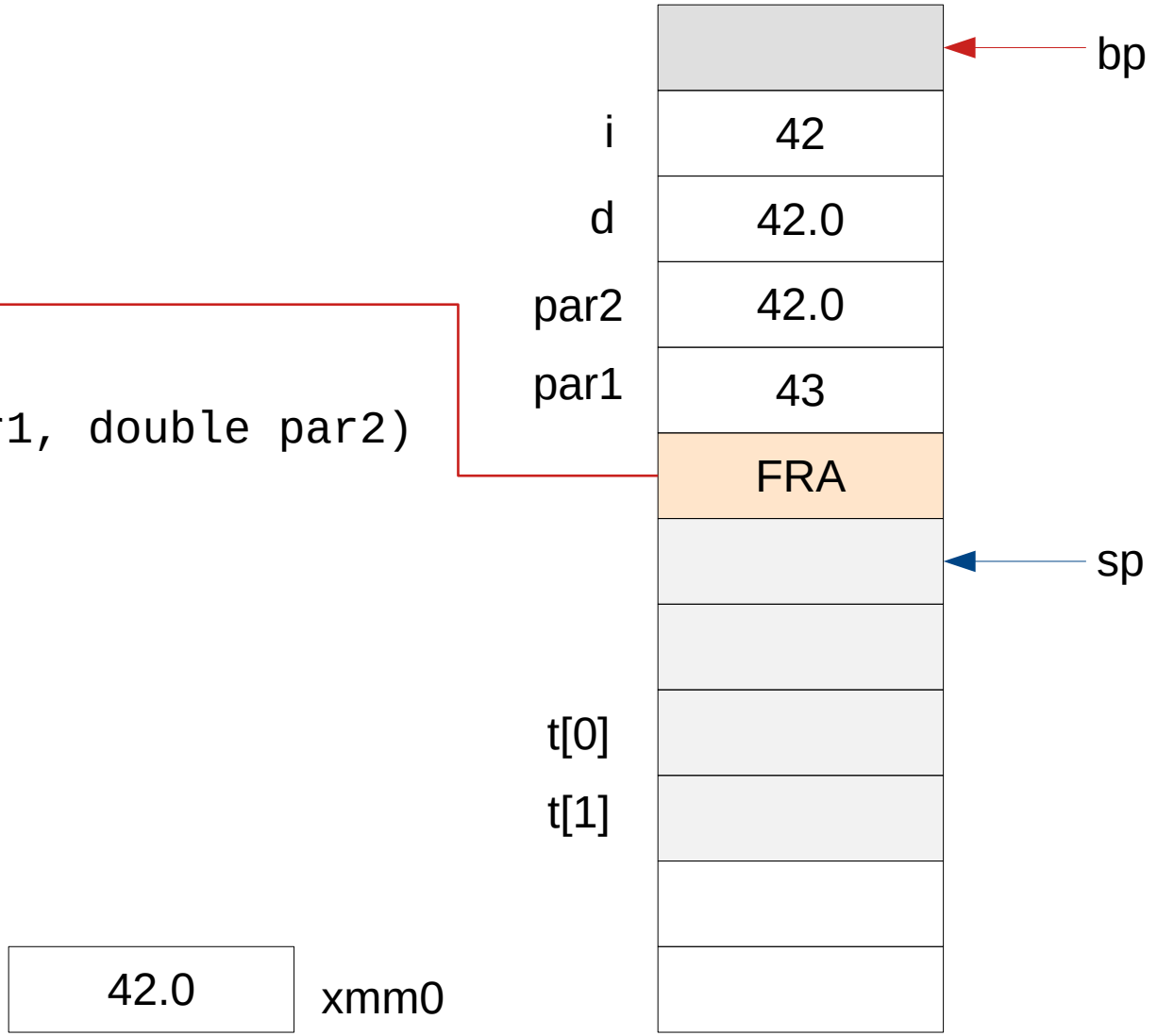
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



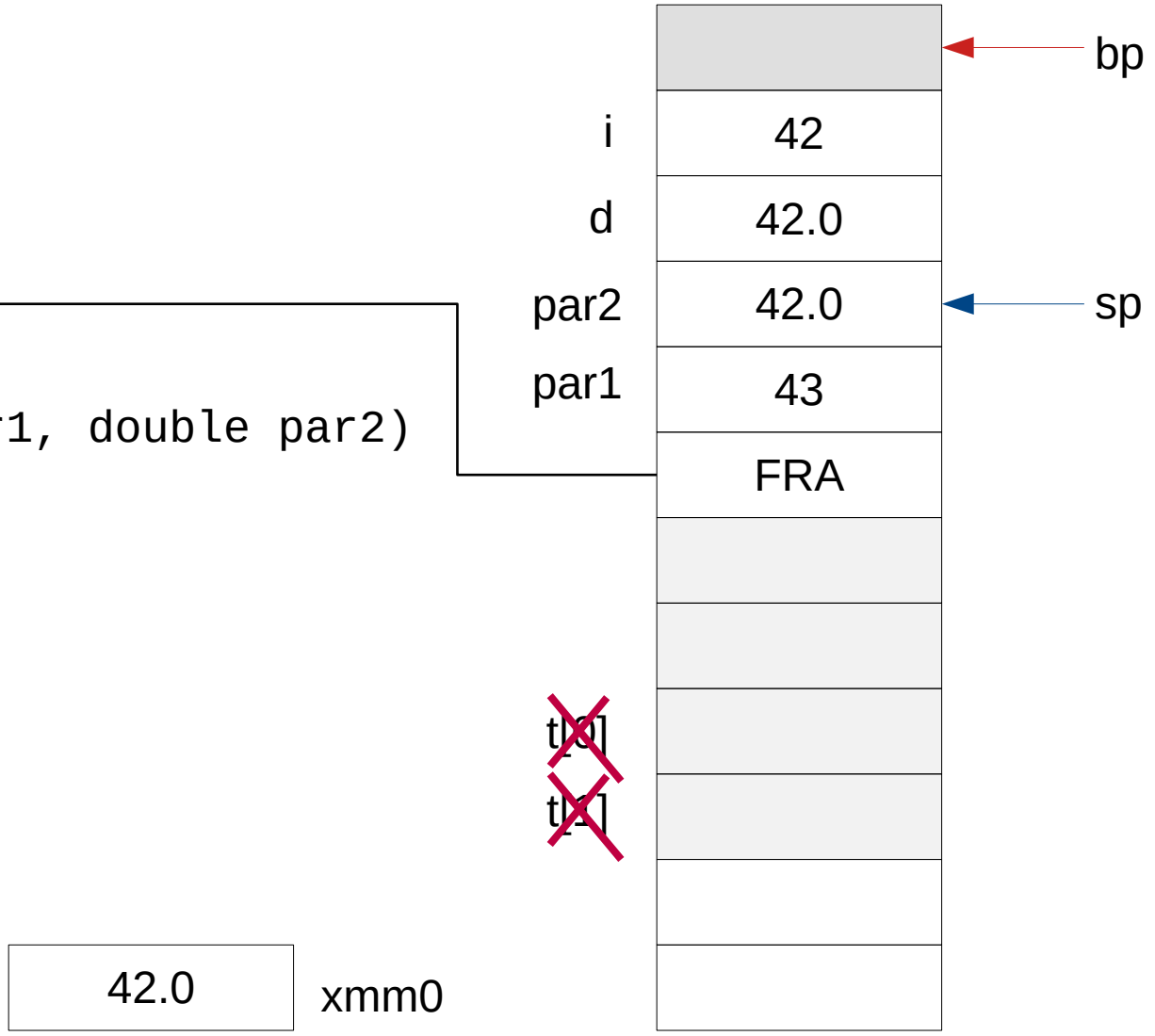
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



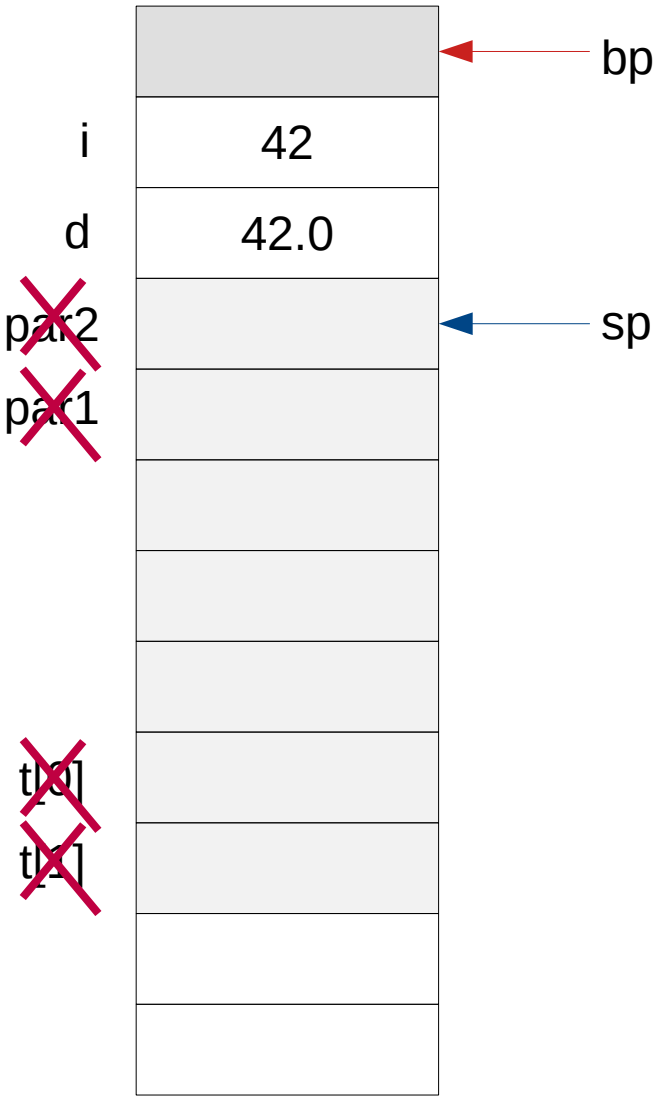
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```

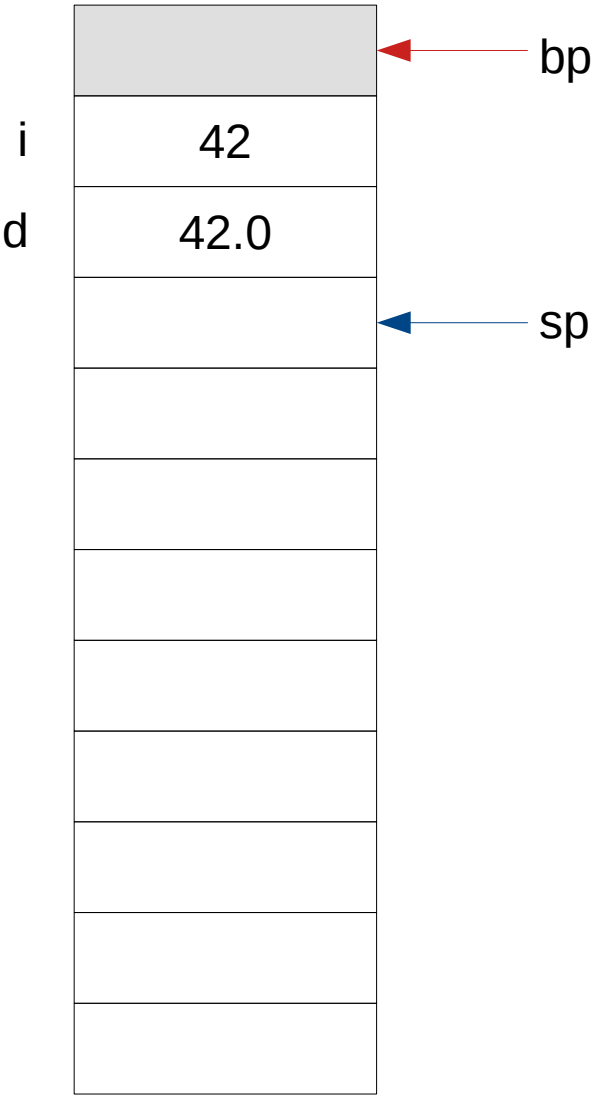


```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```

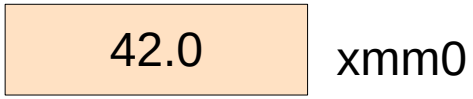


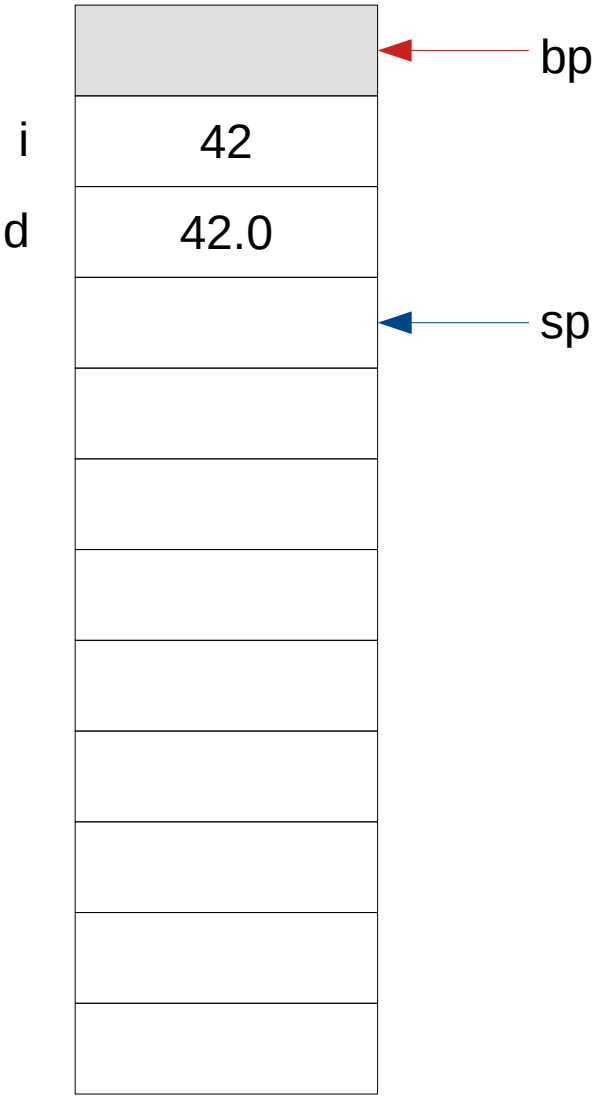
```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```

xmm0

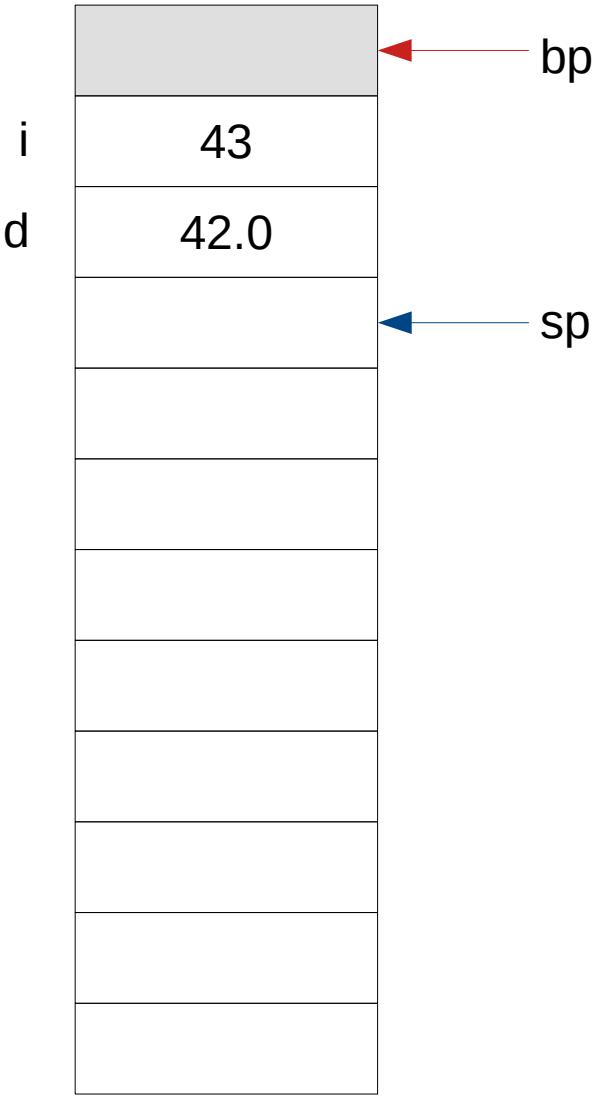


```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```





```
void f()
{
   int i;
   double d;
   i = 42;
   d = i;
   d = g( i, d);
   i++;
}
double g( int par1, double par2)
{
   double t[2];
   t[0] = par1;
   t[1] = par2;
   par1++;
   return t[0];
}
```



Temporaries

- Created when evaluating an expression
- Guaranteed to live until the full expression is evaluated

```
void f( string &s1, string &s2, string &s3)
{
  const char *cs = (s1+s2).c_str();
  cout << cs;     // Bad!!

   if ( strlen(cs = (s2+s3).c_str()) < 8 && cs[0] == 'a' ) // Ok
        cout << cs;     // Bad!!
}
void f( string &s1, string &s2, string &s3)
{
    cout << s1 + s2;            // lifetime extension:
    const string &s = s2 + s3; // binding to name keeps temporary
                                // alive until name goes out of scope
    if ( s.length() < 8 && s[0] == 'a' )
        cout << s;  // Ok
}
// s2+s3 destroyed here: when the "s" reference goes out of scope
```

Lifetime extension

- When a (const) reference is set to a temporary, the temporary will live until the reference goes out of scope

```
struct mystring : std::string {
  mystring(const std::string& s) : std::string(s) {};
  ~mystring() { std::cerr<<"lifetime end: "<< c_str() << '\n'; }
};
mystring operator+( const mystring& a,const mystring& b){
  std::string aa(a), bb(b);
  std::string res(aa+bb);
  return mystring(res);
}
int main() {
  {
    mystring first("first");
    mystring second("second");
    mystring third("third");
    mystring fourth("fourth");
    mystring fifth("fifth");

    const mystring& ref1 = first + second;
    static const mystring& ref2 = first + third;
    static const mystring& ref3 = std::max(fourth,fifth);
    std::cerr << "end of block" << '\n';
  }
  std::cerr << "end of main" << '\n';
}
```

Lifetime extension

- When a (const) reference is set to a temporary, the temporary will live until the reference goes out of scope

```
struct mystring : std::string {
  mystring(const std::string& s) : std::string(s) {};
  ~mystring() { std::cerr<<"lifetime end: "<< c_str() << '\n'; }
};
mystring operator+( const mystring& a,const mystring& b){
  std::string aa(a), bb(b);
  std::string res(aa+bb);
  return mystring(res);
}
int main() {
  {
    mystring first("first");
    mystring second("second");
    mystring third("third");
    mystring fourth("fourth");
    mystring fifth("fifth");

    const mystring& ref1 = first + second;                  // ok
    static const mystring& ref2 = first + third;            // ok
    static const mystring& ref3 = std::max(fourth,fifth); // wrong!!!
    std::cerr << "end of block" << '\n';
  }
  std::cerr << "end of main" << '\n';
}
```

```
end of block
lifetime end: firstsecond
lifetime end: fifth
lifetime end: fourth
lifetime end: third
lifetime end: second
lifetime end: first
end of main
lifetime end: firstthird
```

Lifetime extension

- When a (const) reference is set to a temporary, the temporary will live until the reference goes out of scope

```
template <class T>
constexpr const T& max(const T& a, const T& b);
```

```
int main() {
  {
    mystring first("first");
    mystring second("second");
    mystring third("third");
    mystring fourth("fourth");
    mystring fifth("fifth");

    const mystring& ref1 = first + second;                // ok
    static const mystring& ref2 = first + third;          // ok
    static const mystring& ref3 = std::max(fourth,fifth); // wrong!!!
    std::cerr << "end of block" << '\n';
  }
  std::cerr << "end of main" << '\n';
}
```

```
end of block
lifetime end: firstsecond
lifetime end: fifth
lifetime end: fourth
lifetime end: third
lifetime end: second
lifetime end: first
end of main
lifetime end: firstthird
```

Lifetime extension

- When a (const) reference is set to a temporary, or member of a temporary the temporary will live until the reference goes out of scope

```
struct mystring : std::string {
  mystring(const std::string& s) : std::string(s) {};
  ~mystring() { std::cerr<<"lifetime end: "<< c_str() << '\n'; }
  int ifield;
};
mystring operator+( const mystring& a,const mystring& b){
  std::string aa(a), bb(b);
  std::string res(aa+bb);
  return mystring(res);
}
int main() {
  {
    mystring first("first");
    mystring fourth("fourth");
    mystring fifth("fifth");

    static const int &ref4 = (first + fourth).ifield;
    static const char *const &ref5 = (first + fifth).c_str();
    std::cerr << "end of block" << '\n';
  }
  std::cerr << "end of main" << '\n';
}
```

Lifetime extension

- When a (const) reference is set to a temporary, or member of a temporary the temporary will live until the reference goes out of scope

```
struct mystring : std::string {
  mystring(const std::string& s) : std::string(s) {};
  ~mystring() { std::cerr<<"lifetime end: "<< c_str() << '\n'; }
  int ifield;
};
mystring operator+( const mystring& a,const mystring& b){
  std::string aa(a), bb(b);
  std::string res(aa+bb);
  return mystring(res);
}
int main() {
  {
    mystring first("first");
    mystring fourth("fourth");
    mystring fifth("fifth");

    static const int &ref4 = (first + fourth).ifield;          // ok
    static const char *const &ref5 = (first + fifth).c_str(); // wrong!!!
    std::cerr << "end of block" << '\n';
  }
  std::cerr << "end of main" << '\n';
}
```

```
end of block
lifetime end: firstfifth
lifetime end: fifth
lifetime end: fourth
lifetime end: first
end of main
lifetime end: firstfourth
```

Dynamic lifetime

- Begins on the call of **new** or **new[ ]** expression
  - Different syntax for a single object or for arrays
  - Usually calls **operator new()** internally
  - There are several overloads of **operator new()**
- Ends on the call of **delete** or **delete[ ]** expression
- Placed in the “heap”
- No automatic garbage collection in C++
  - Objects local to a block (and non-static)
  - Memory leak is a common C++ error
  - Use RAII (Resource Acquisition Is Initialization) technique
  - Smart pointers (?)

Dynamic lifetime

```
void reverse()
{
    std::cout <<  "How many elements? ";
    int n;
    std::cin  >> n;

    n =  n > 0 ? n : 10;
    int *ptr = new int[n]; // life starts here, or std::bad_alloc exception

    for ( int i = 0; i < n; ++i)
    {
        std::cin >> ptr[i];   // (*ptr)[i]
    }
    for ( int i = n-1; i >= 0; --i)
    {
        std::out << ptr[i] << '\n';
    }

    delete [] ptr;   // life ends here
}
catch ( std::bad_alloc )
{
    std::cerr << "No memory\n";
}
```

Dynamic lifetime

```
double *ptr1;
double *ptr2;
double *ptr3;
double *ptr4;


void res()
{
    ptr1 = new double;        // 1 double, uninitialized
    ptr2 = new double{3.14}; // 1 double, initialized

    ptr3 = new double[2];    // 2 doubles, uninitialized
    ptr4 = new double[2]{2.71,3.14}; // 2 doubles, initialized
}

void fre()
{
    delete    d1;   // ok
    delete [] d2;   // run-time error: unallocate non-array as array
    delete [] d3;   // ok
    delete    d4;   // run-time error: unallocate array as non-array
}
```

Typical errors with scope and life

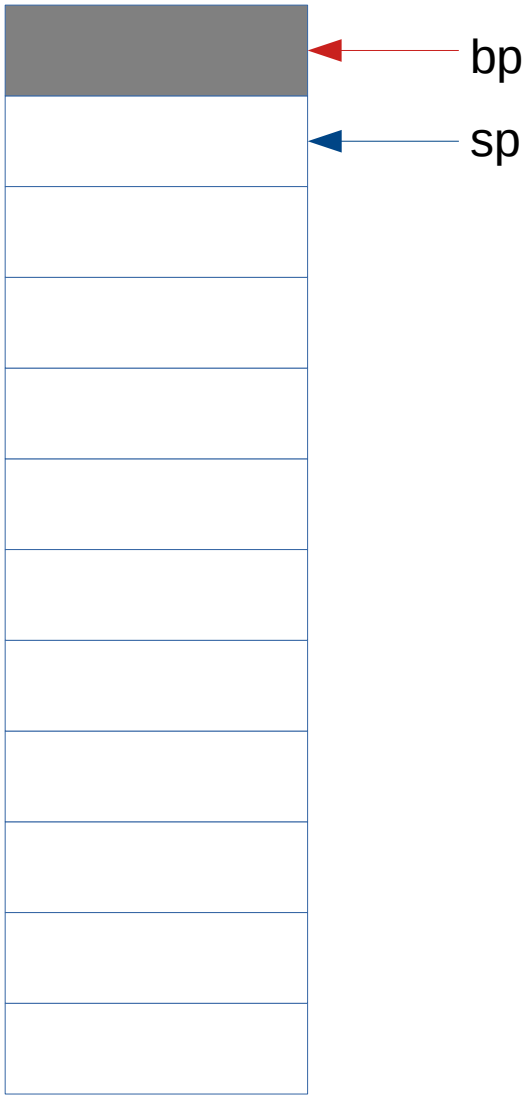
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}


char *answer( char *question)
{
   char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```

```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}


char *answer( char *question)
{
   char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```

.rodata

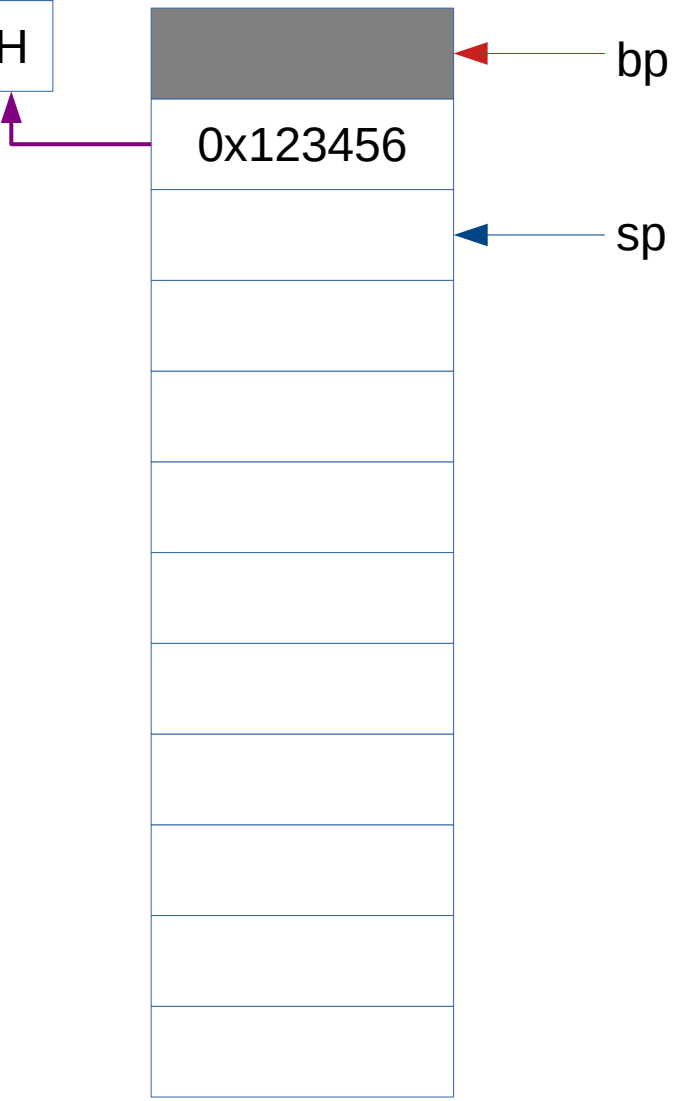
\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}


char *answer( char *question)
{
   char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



.rodata

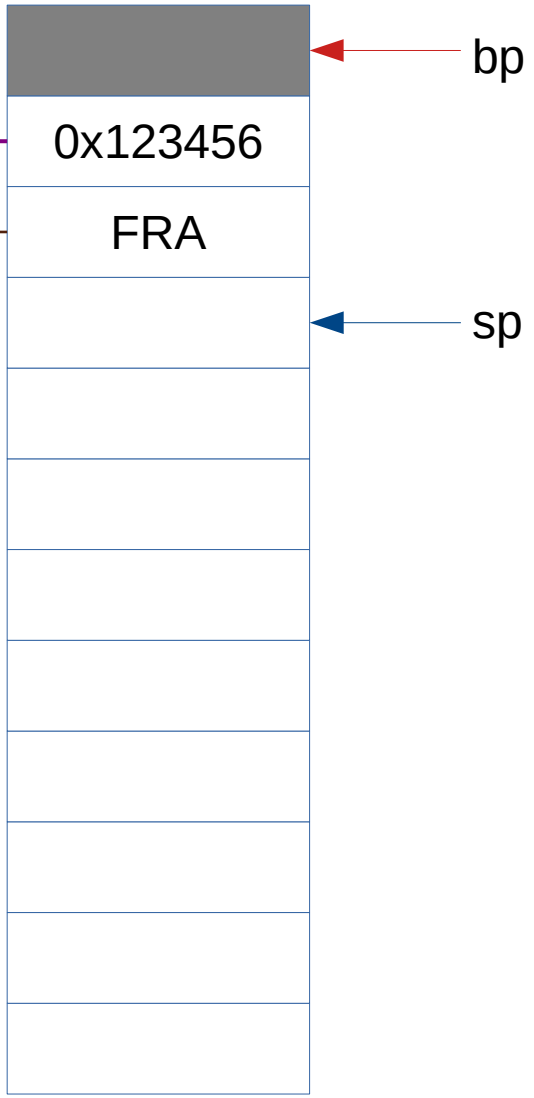
\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H




```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}


char *answer( char *question)
{
   char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```

.rodata

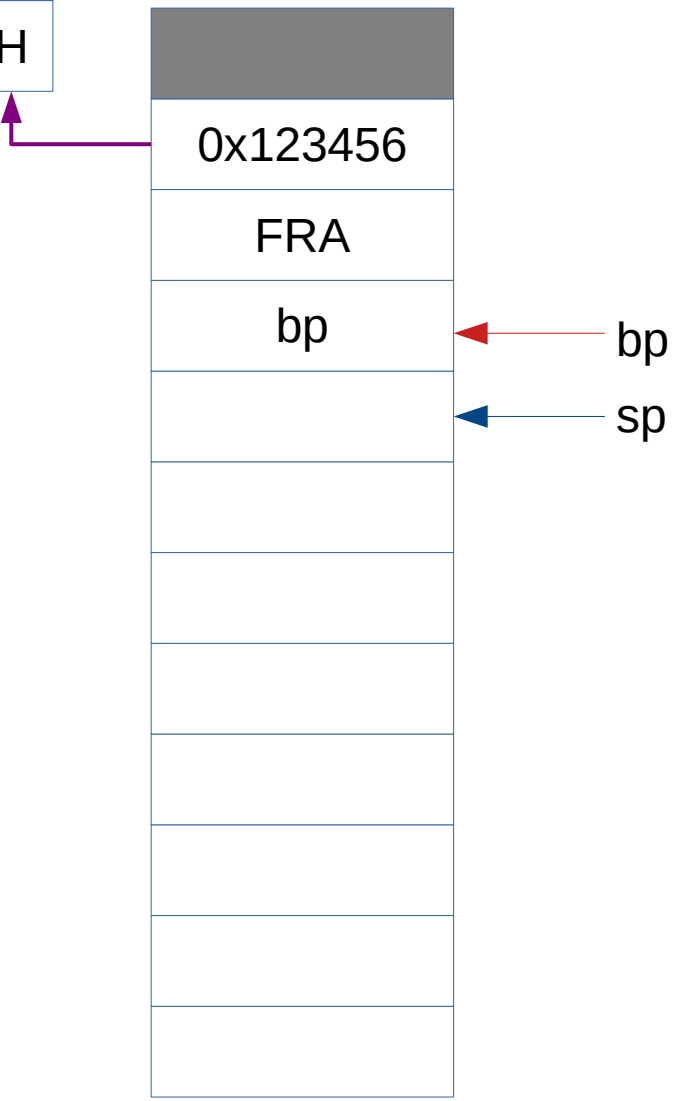
\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H




```
void f()
{
    std::cout << "answer = "
              <<answer("How are you? ")
              << '\n';
}


char *answer( char *question)
{
    char buffer[20];
    std::cout << question;
    std::cin >> buffer;
    return buffer;
}
```

.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

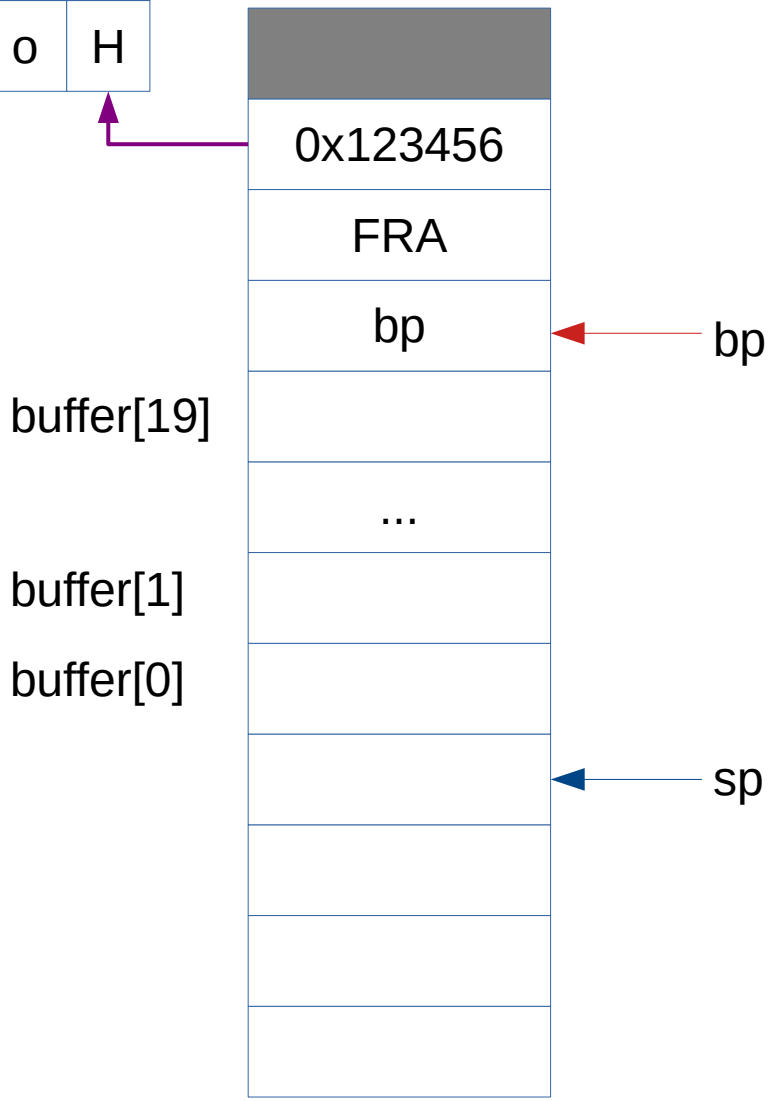



```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}


char *answer( char *question)
{
   char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```

.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

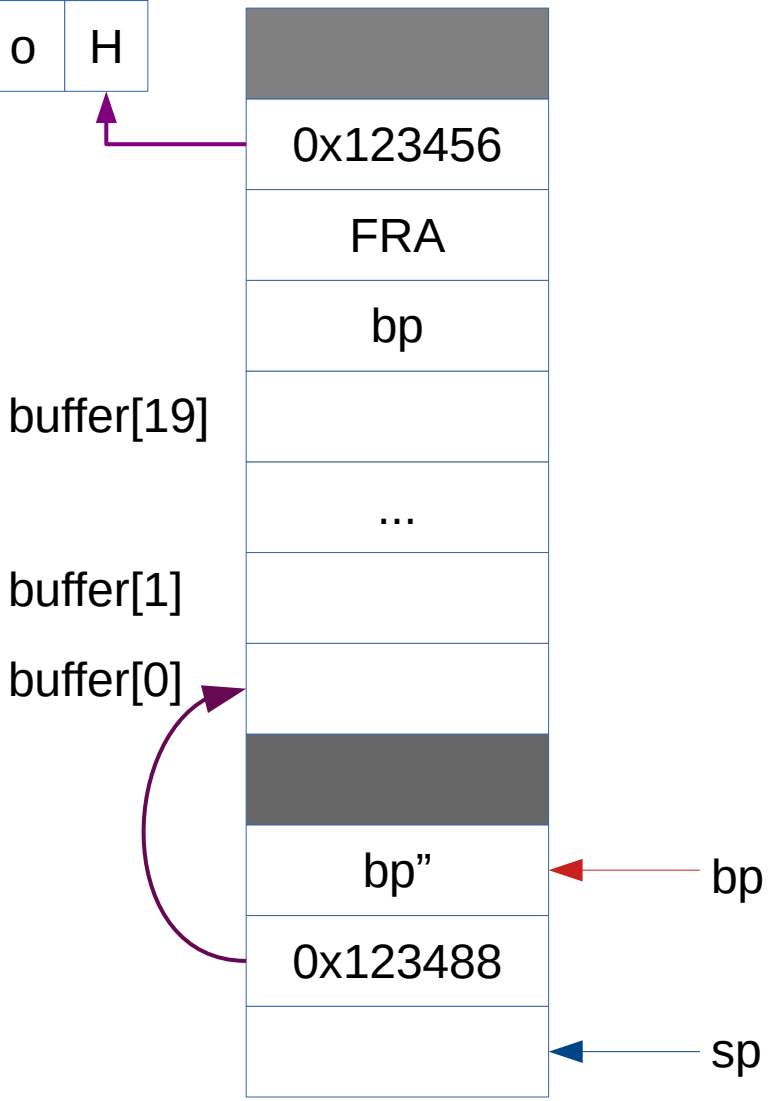
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}

char *answer( char *question)
{
   char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

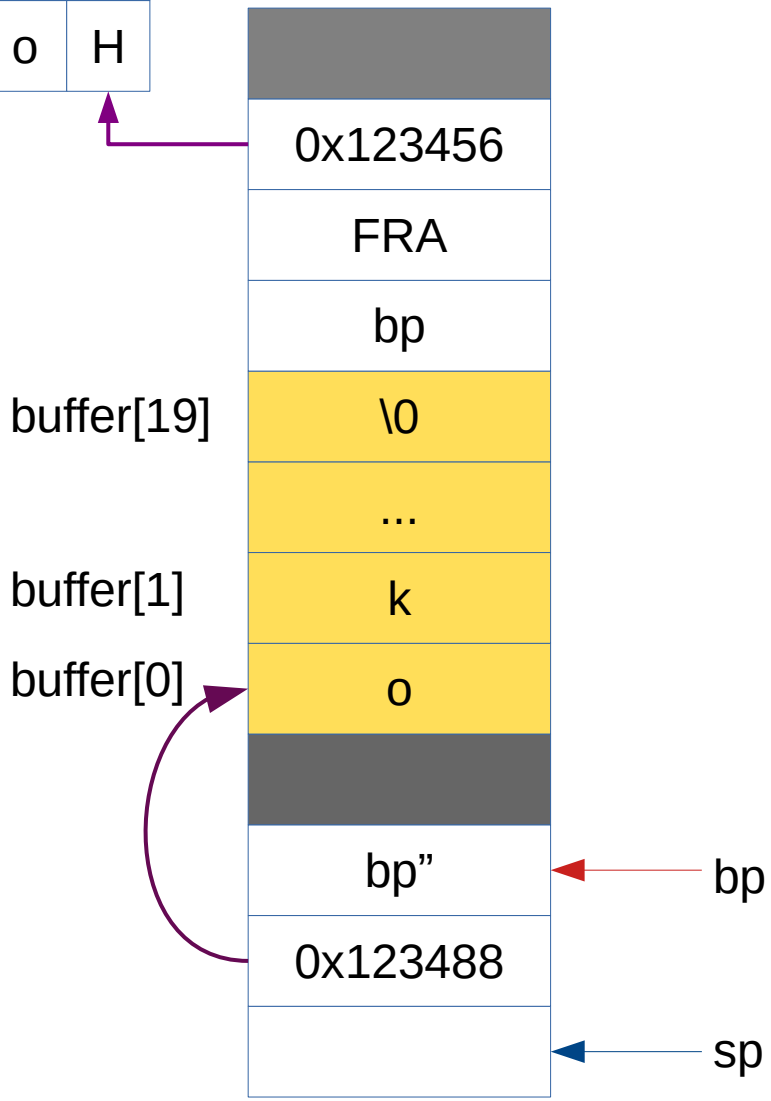
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}


char *answer( char *question)
{
   char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

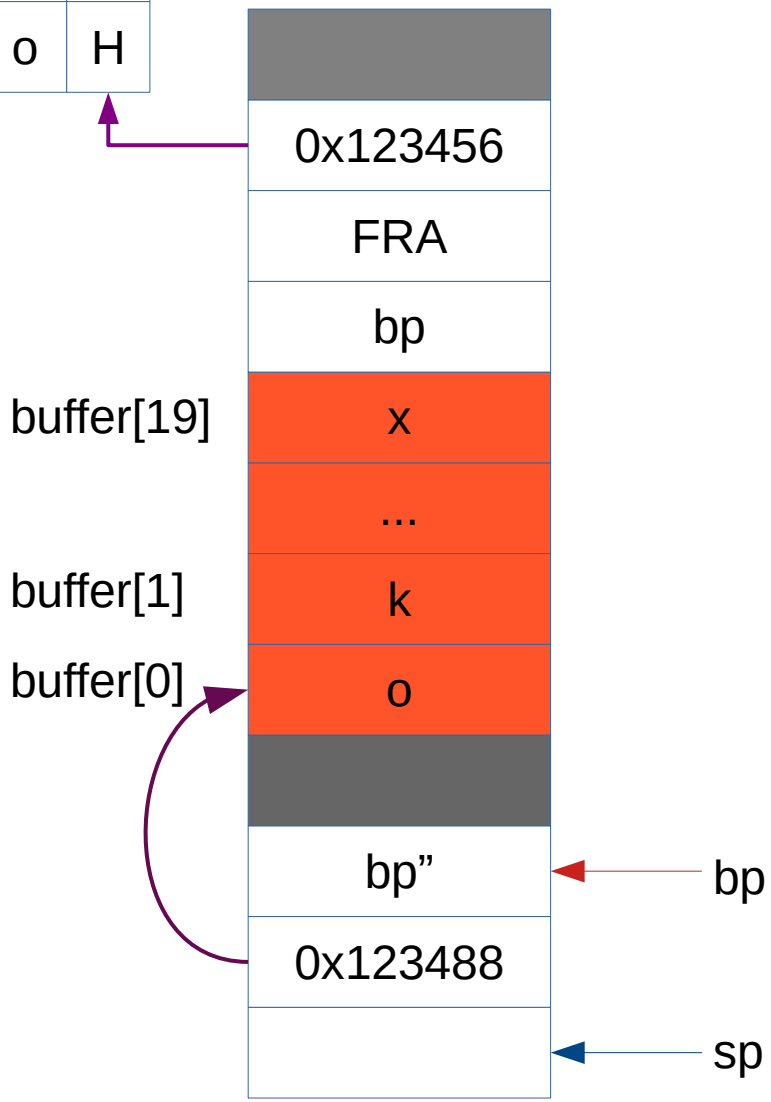
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}


char *answer( char *question)
{
   char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

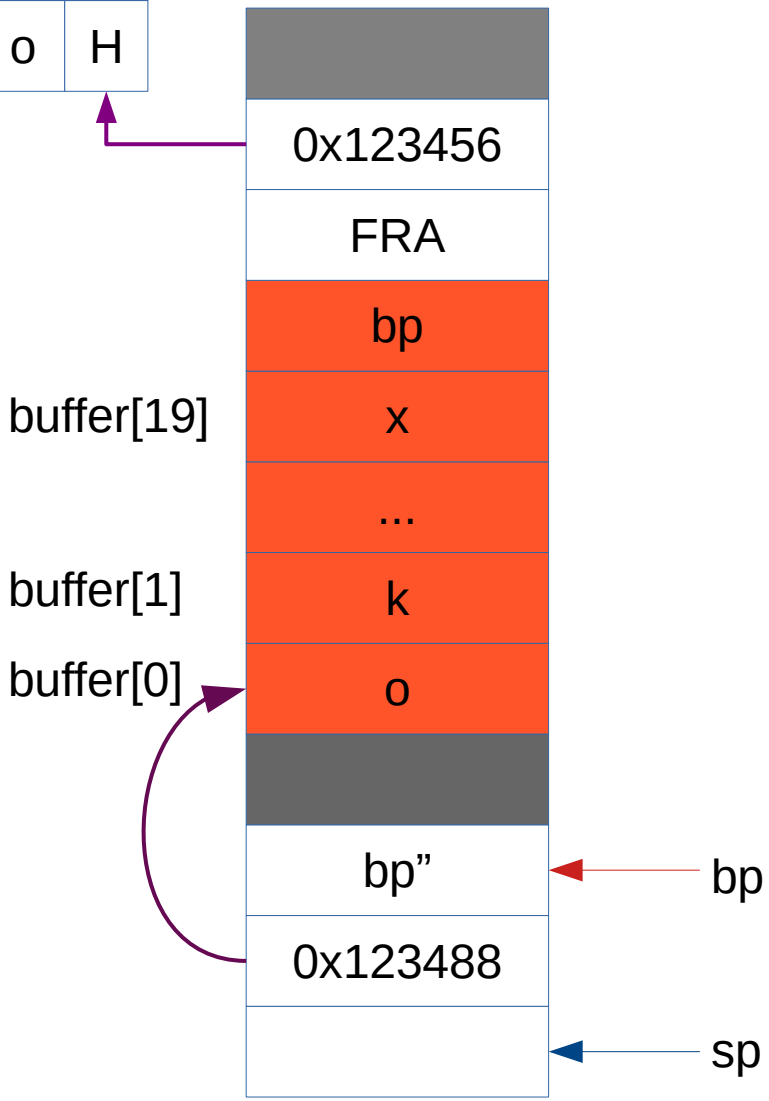
```
void f()
{
    std::cout << "answer = "
              << answer("How are you? ")
              << '\n';
}


char *answer( char *question)
{
    char buffer[20];
    std::cout << question;
    std::cin >> buffer;
    return buffer;
}
```



.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

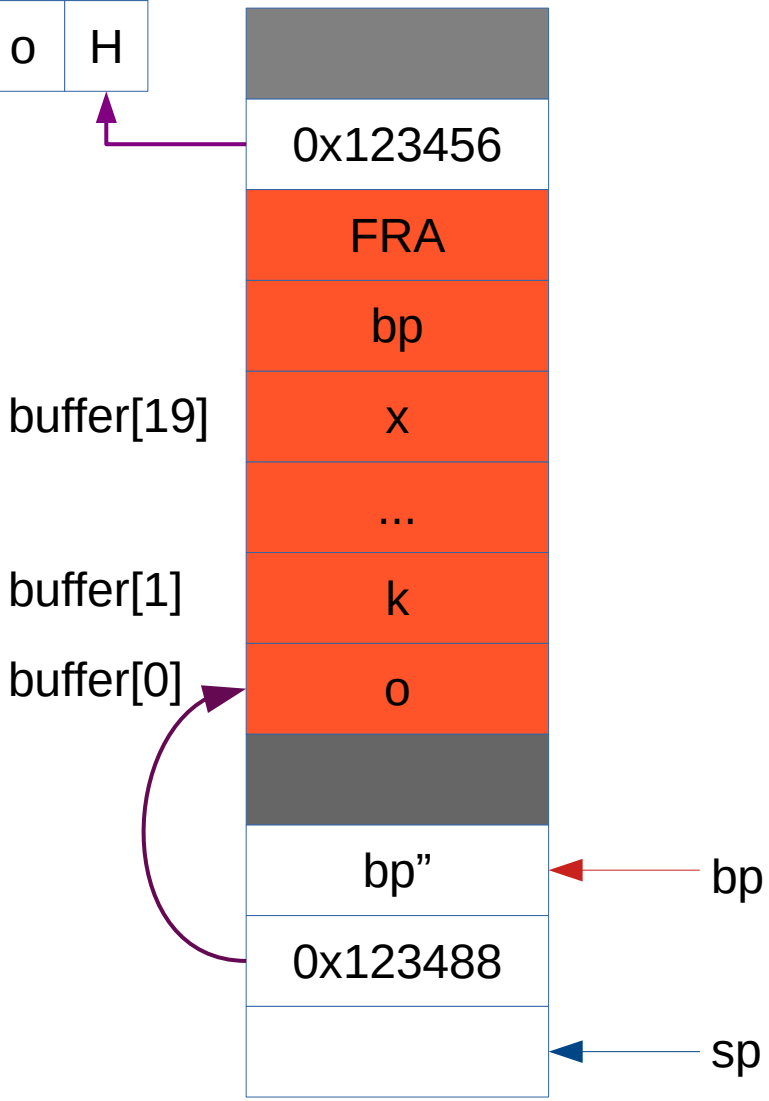
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}


char *answer( char *question)
{
   char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

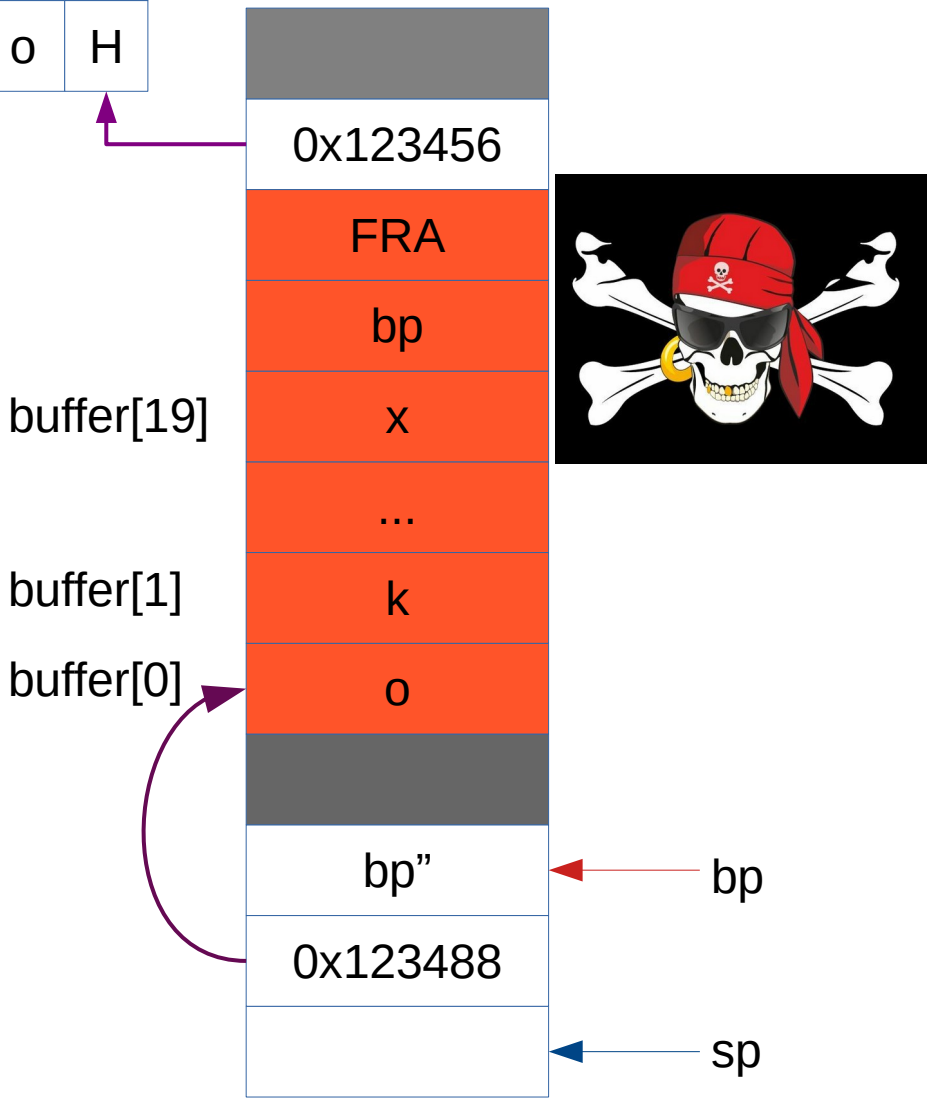
```
void f()
{
    std::cout << "answer = "
              << answer("How are you? ")
              << '\n';
}


char *answer( char *question)
{
    char buffer[20];
    std::cout << question;
    std::cin >> buffer;
    return buffer;
}
```



.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f()
{
    std::cout << "answer = "
              << answer("How are you? ")
              << '\n';
}


char *answer( char *question)
{
    char buffer[20];
    std::cout << question;
    std::cin >> buffer; // before C++20
    return buffer;
}
```

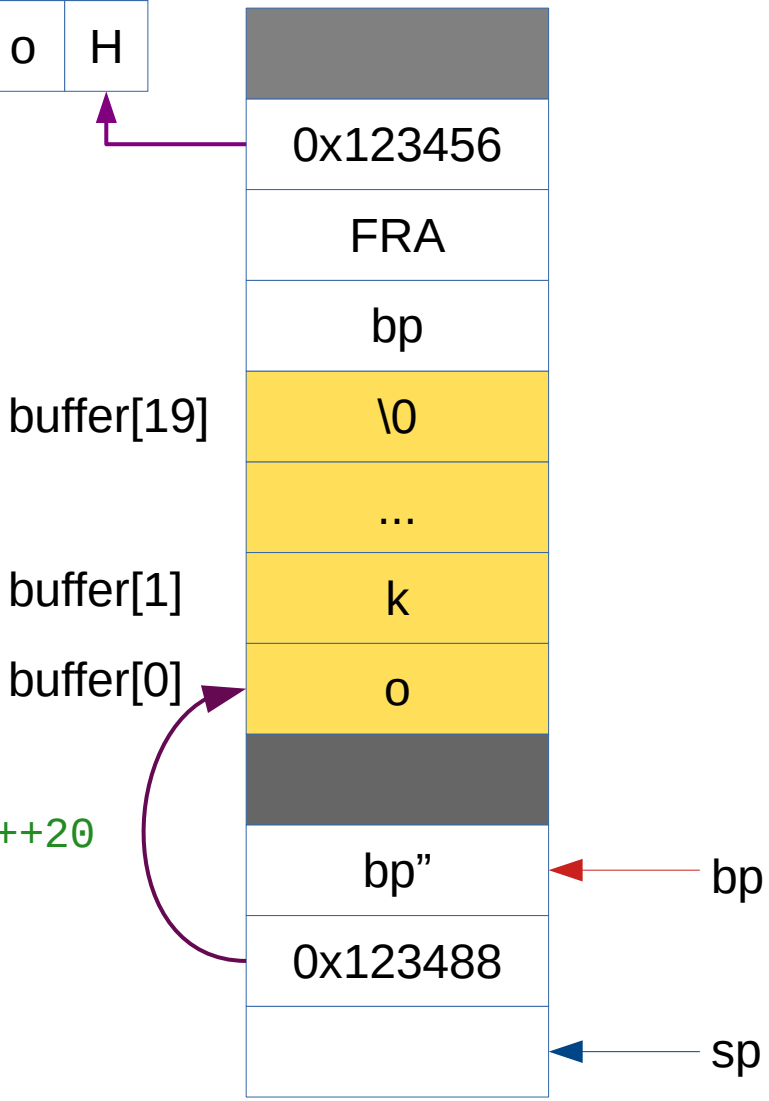


BUFFER OVERFLOW!

.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

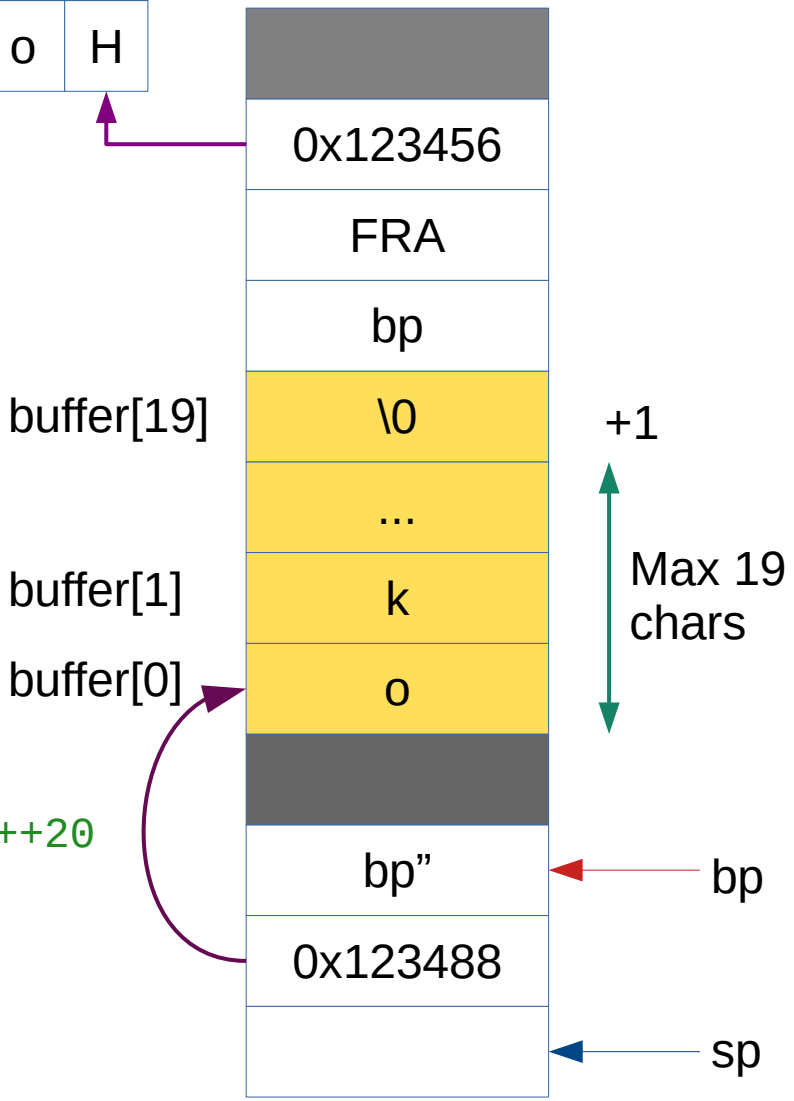
```
void f()
{
    std::cout << "answer = "
              << answer("How are you? ")
              << '\n';
}


char *answer( char *question)
{
    char buffer[20];
    std::cout << question;
    std::cin >> buffer;  // safe since C++20
    return buffer;
}
```



.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

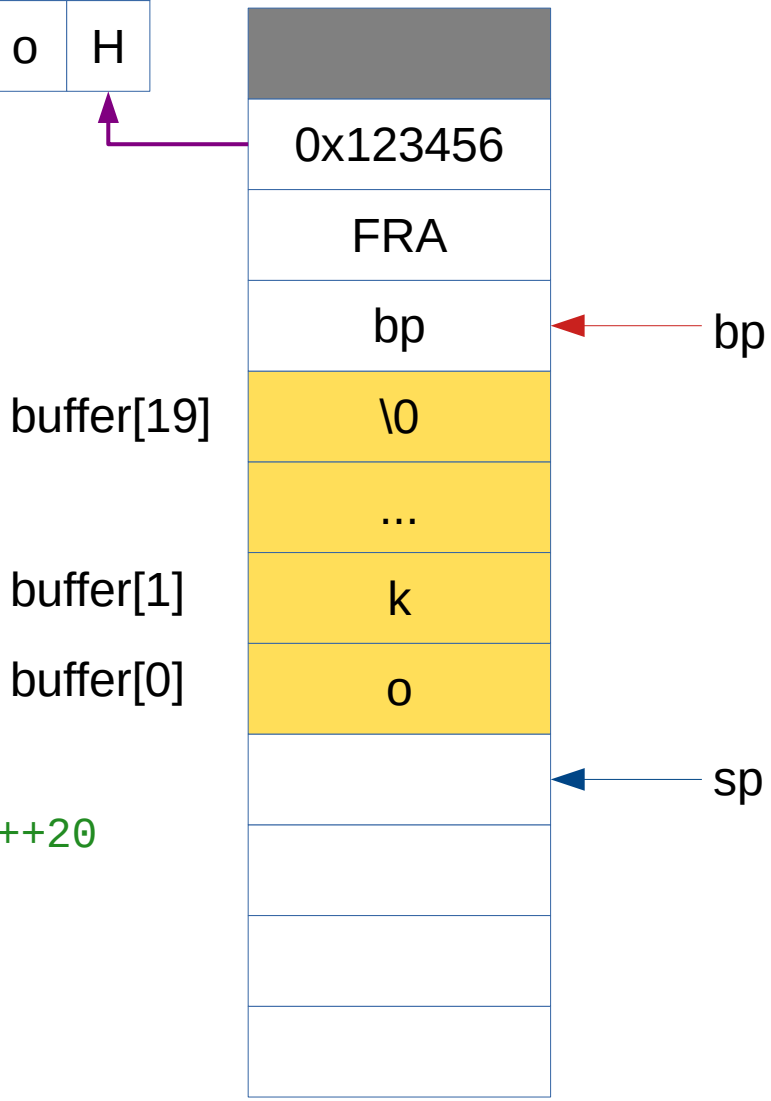
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}


char *answer( char *question)
{
   char buffer[20];
   std::cout << question;
   std::cin >> buffer;  // safe since C++20
   return buffer;
}
```



.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

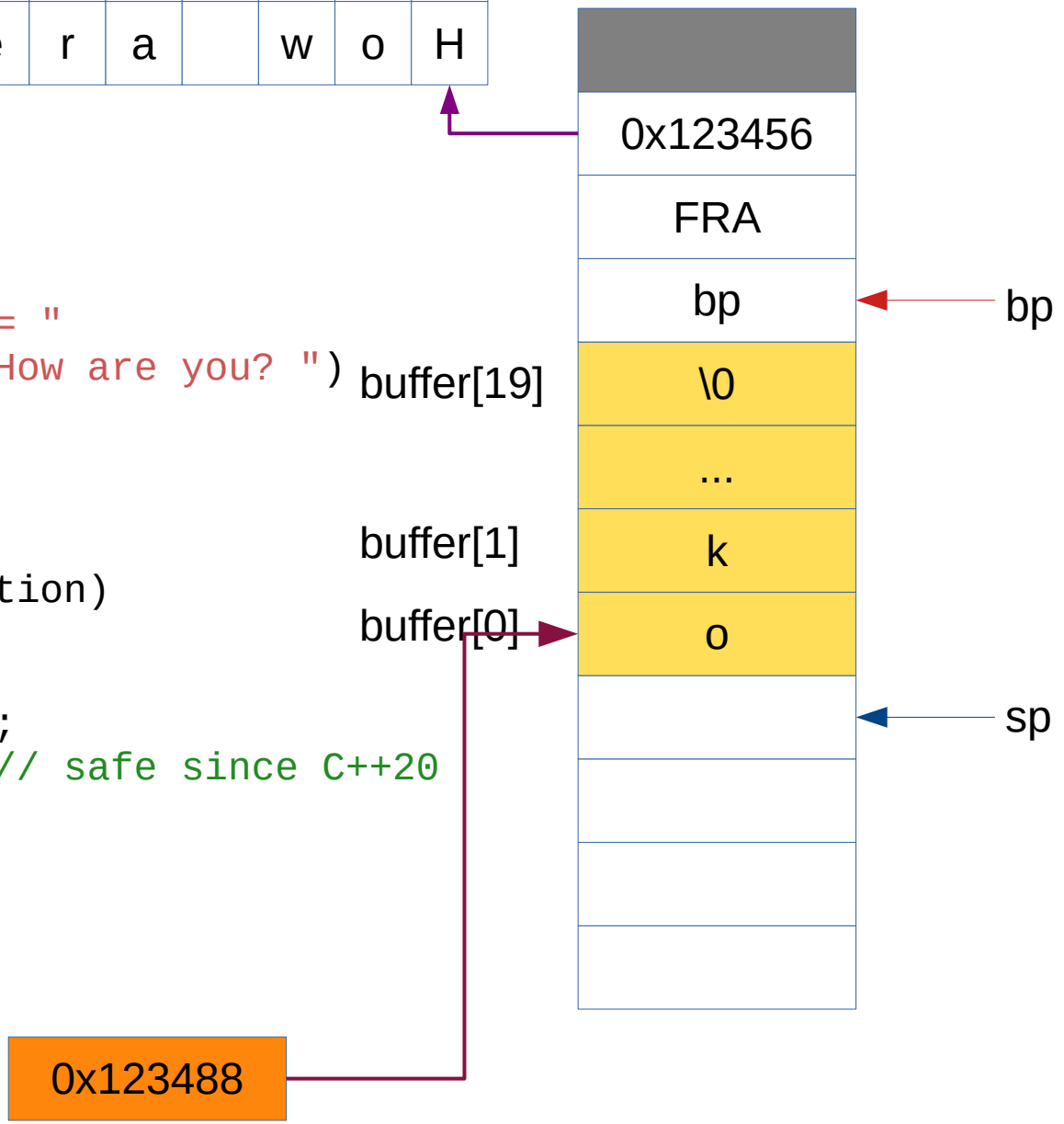
```
void f()
{
    std::cout << "answer = "
              << answer("How are you? ")
              << '\n';
}


char *answer( char *question)
{
    char buffer[20];
    std::cout << question;
    std::cin >> buffer;  // safe since C++20
    return buffer;
}
```

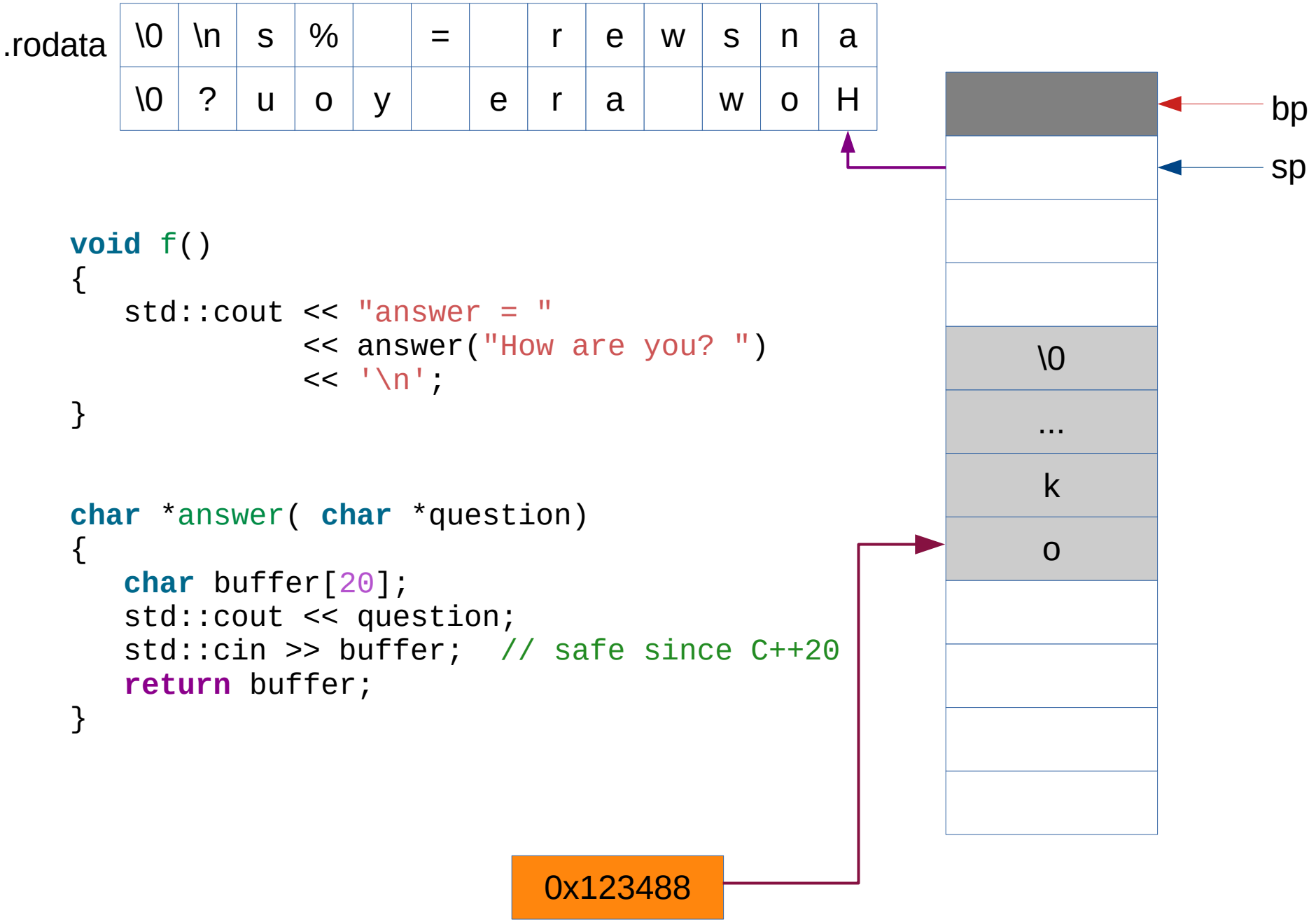


.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

```
void f()
{
    std::cout << "answer = "
              << answer("How are you? ")
              << '\n';
}


char *answer( char *question)
{
    char buffer[20];
    std::cout << question;
    std::cin >> buffer;  // safe since C++20
    return buffer;
}
```

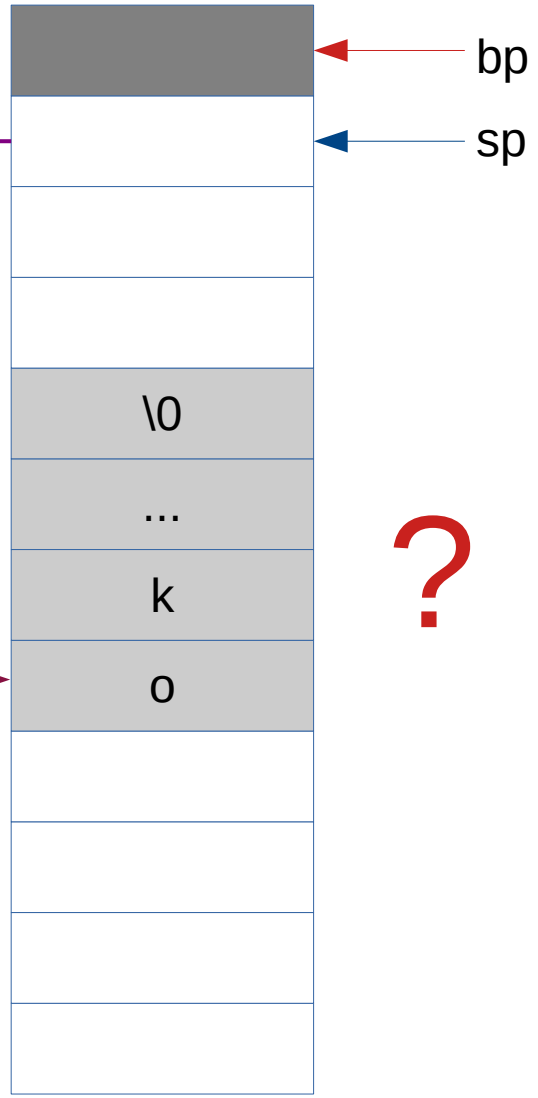





```
void f()
{
    std::cout << "answer = "
              << answer("How are you? ")
              << '\n';
}


char *answer( char *question)
{
    char buffer[20];
    std::cout << question;
    std::cin >> buffer;  // safe since C++20
    return buffer;
}
```

.rodata

\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H



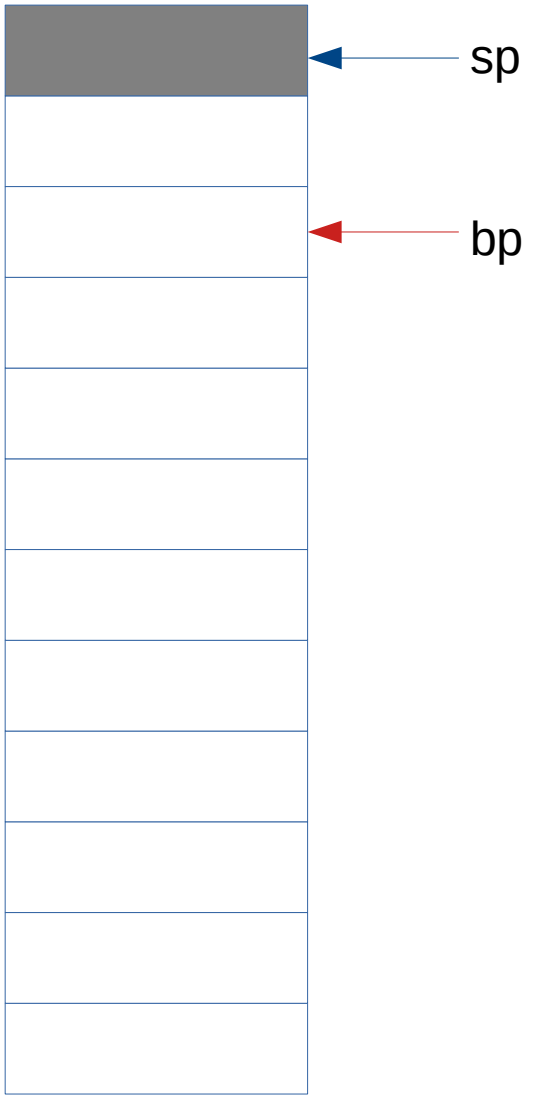

```
void f()
{
    std::cout << "answer = "
              << answer("How are you? ")
              << '\n';
}


char *answer( char *question)
{
    char buffer[20];
    std::cout << question;
    std::cin >> buffer;  // safe since C++20
    return buffer;
}
```

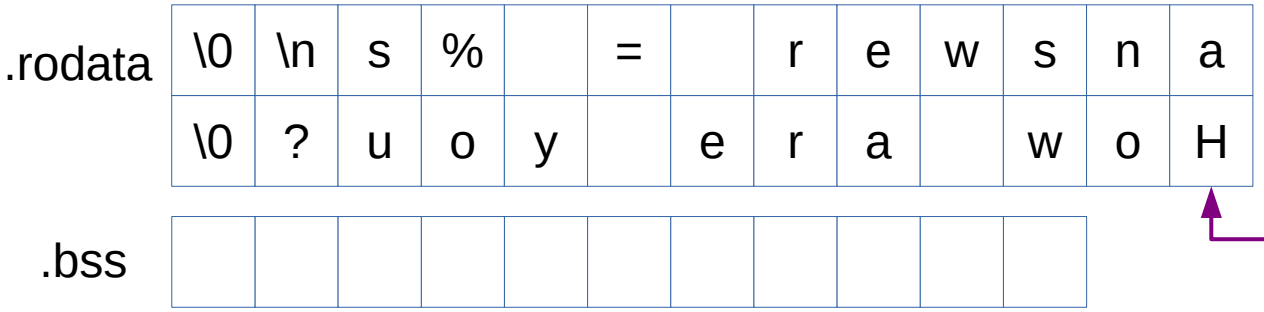
.rodata

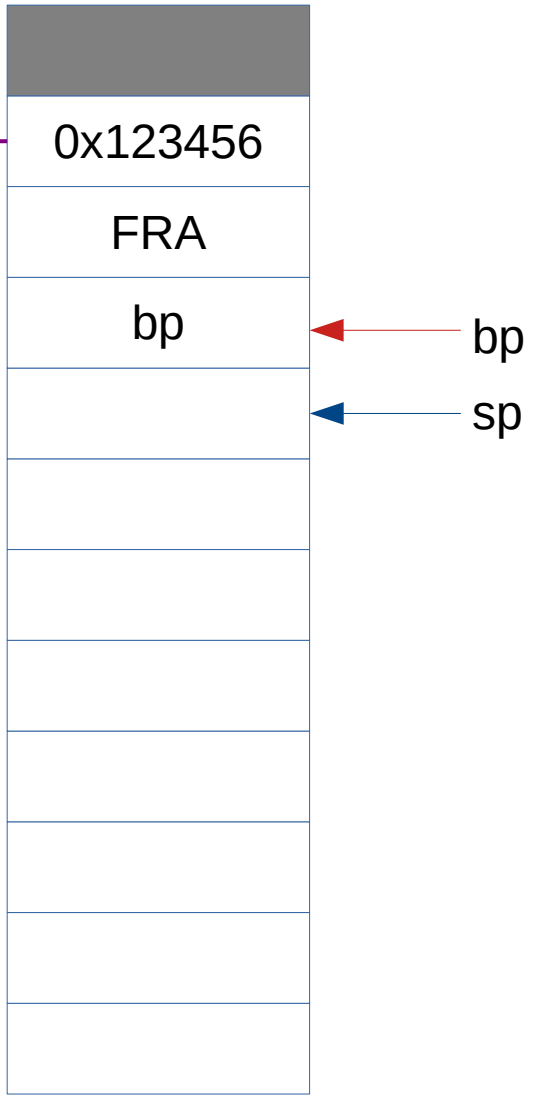
\0	\n	s	%		=		r	e	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

.bss

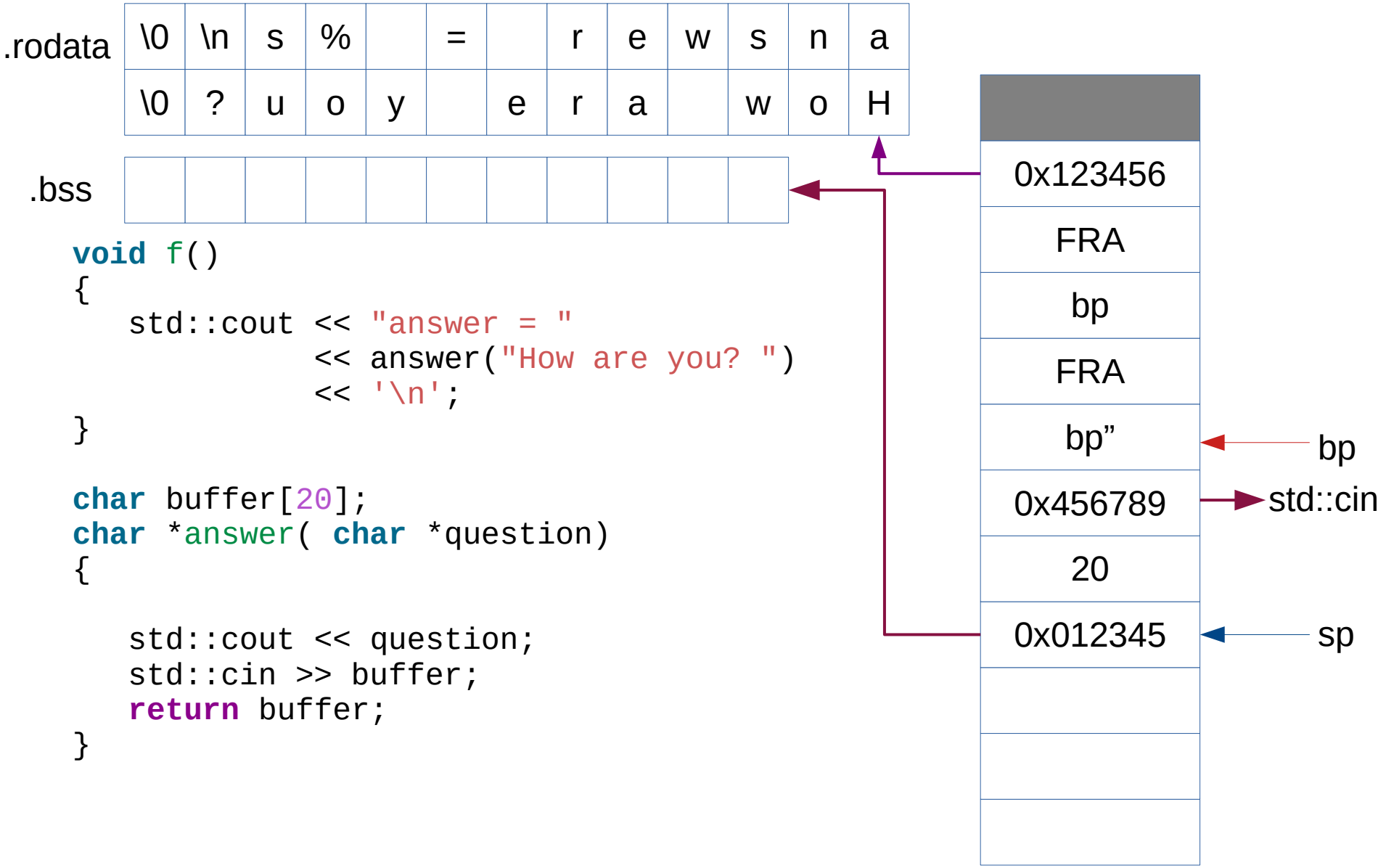

```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}

char buffer[20];
char *answer( char *question)
{
   char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



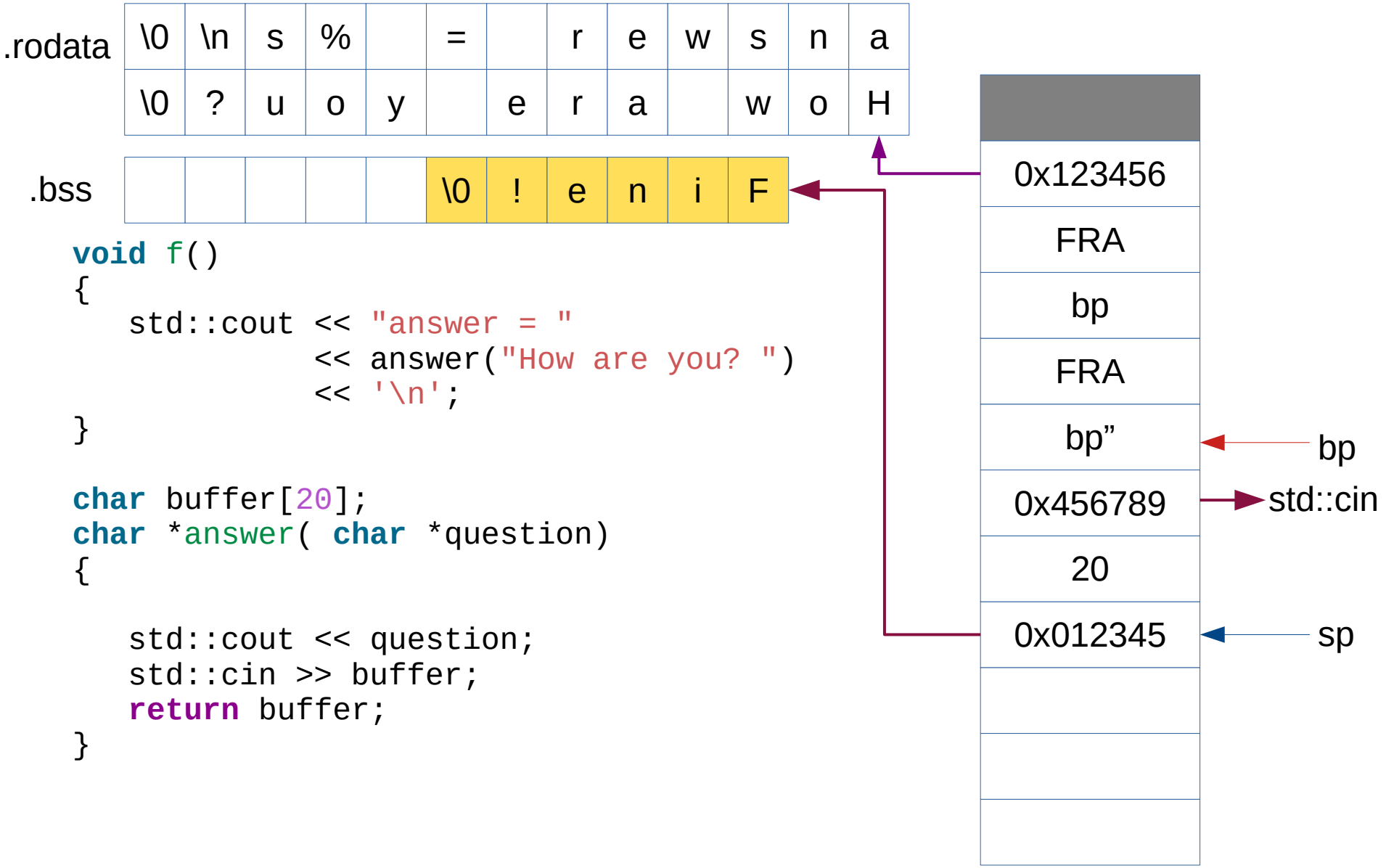




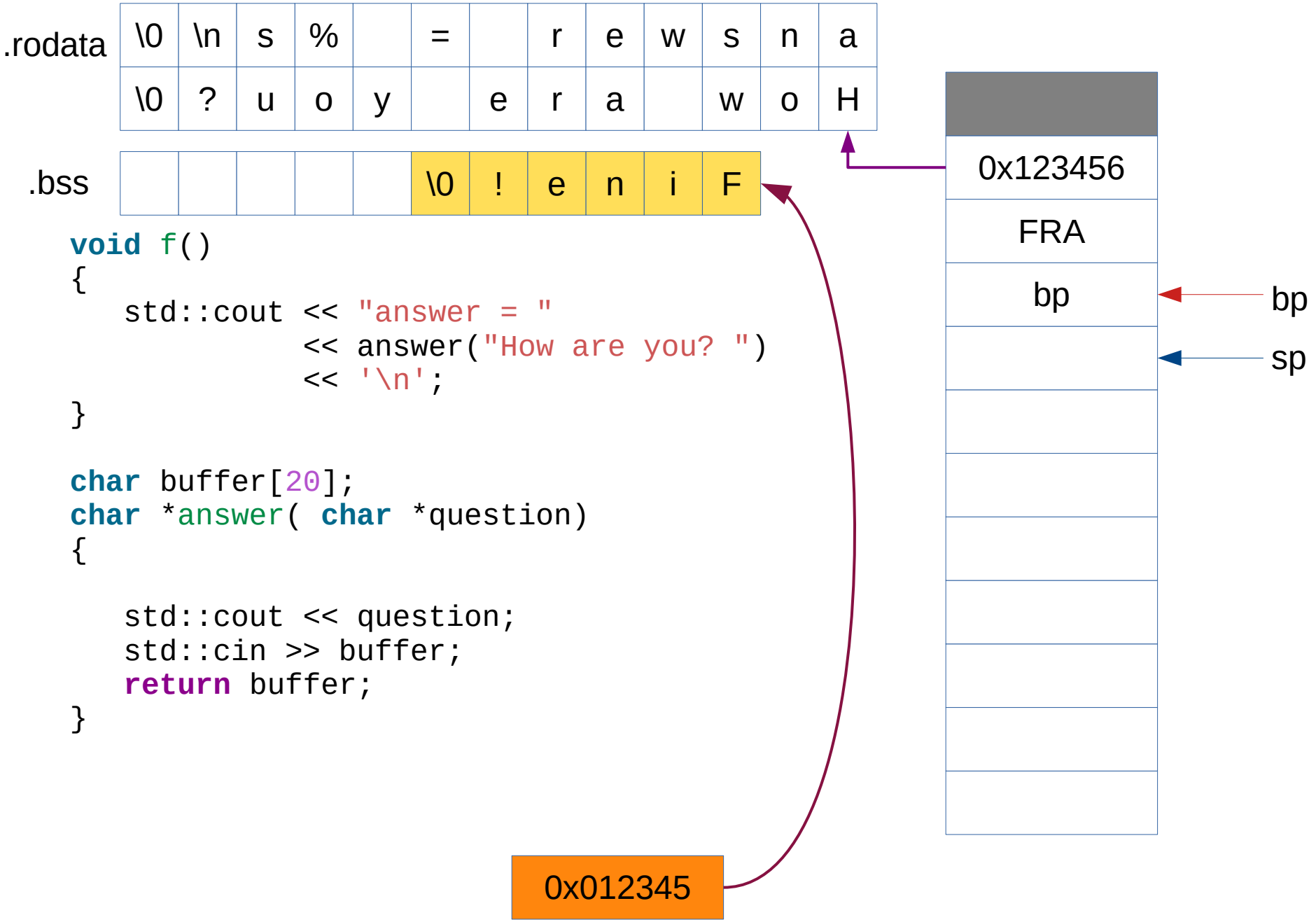

```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}

char buffer[20];
char *answer( char *question)
{

   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



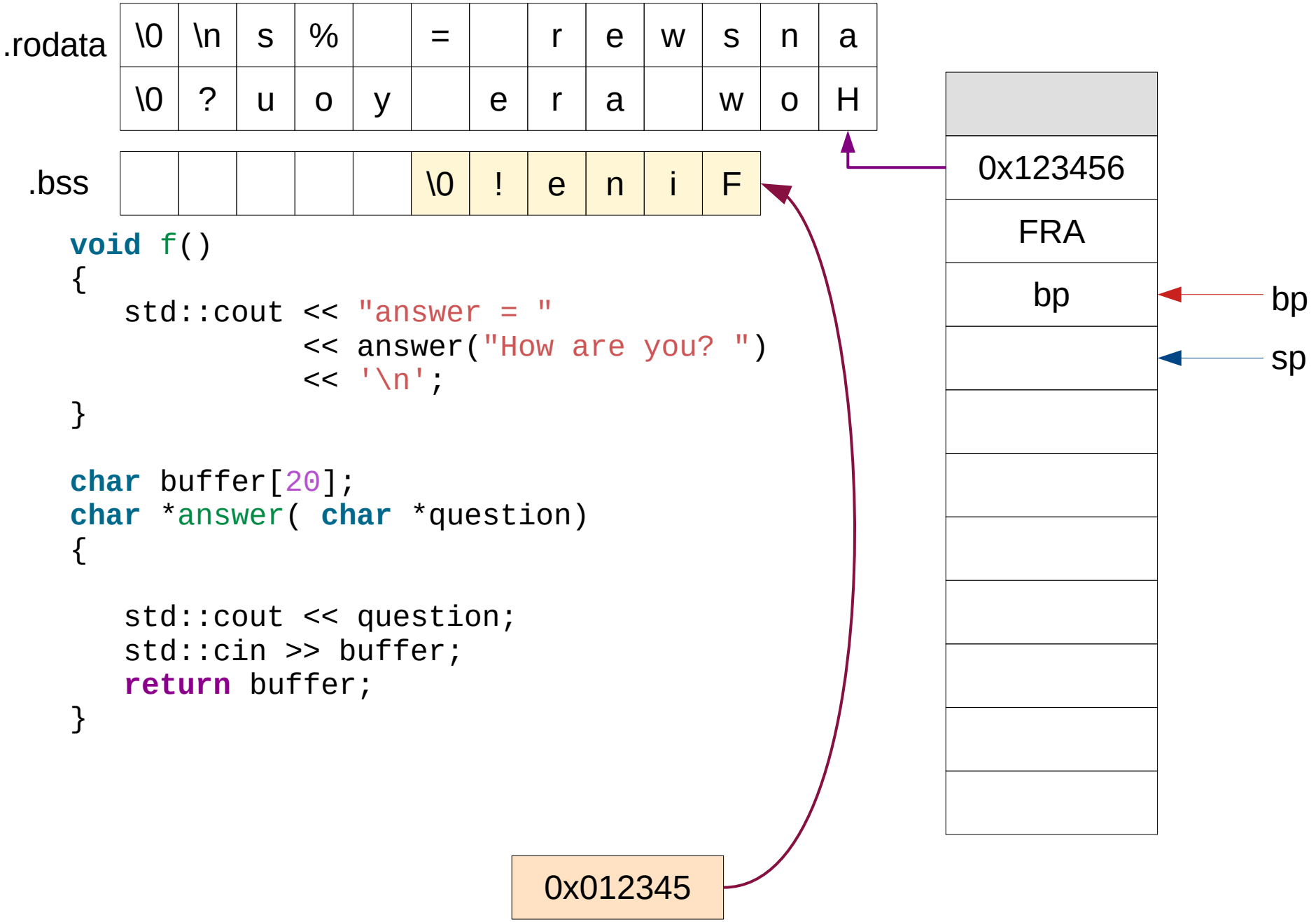

```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}

char buffer[20];
char *answer( char *question)
{

   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```




```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}

char buffer[20];
char *answer( char *question)
{

   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```




```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}

char buffer[20];
char *answer( char *question)
{

   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



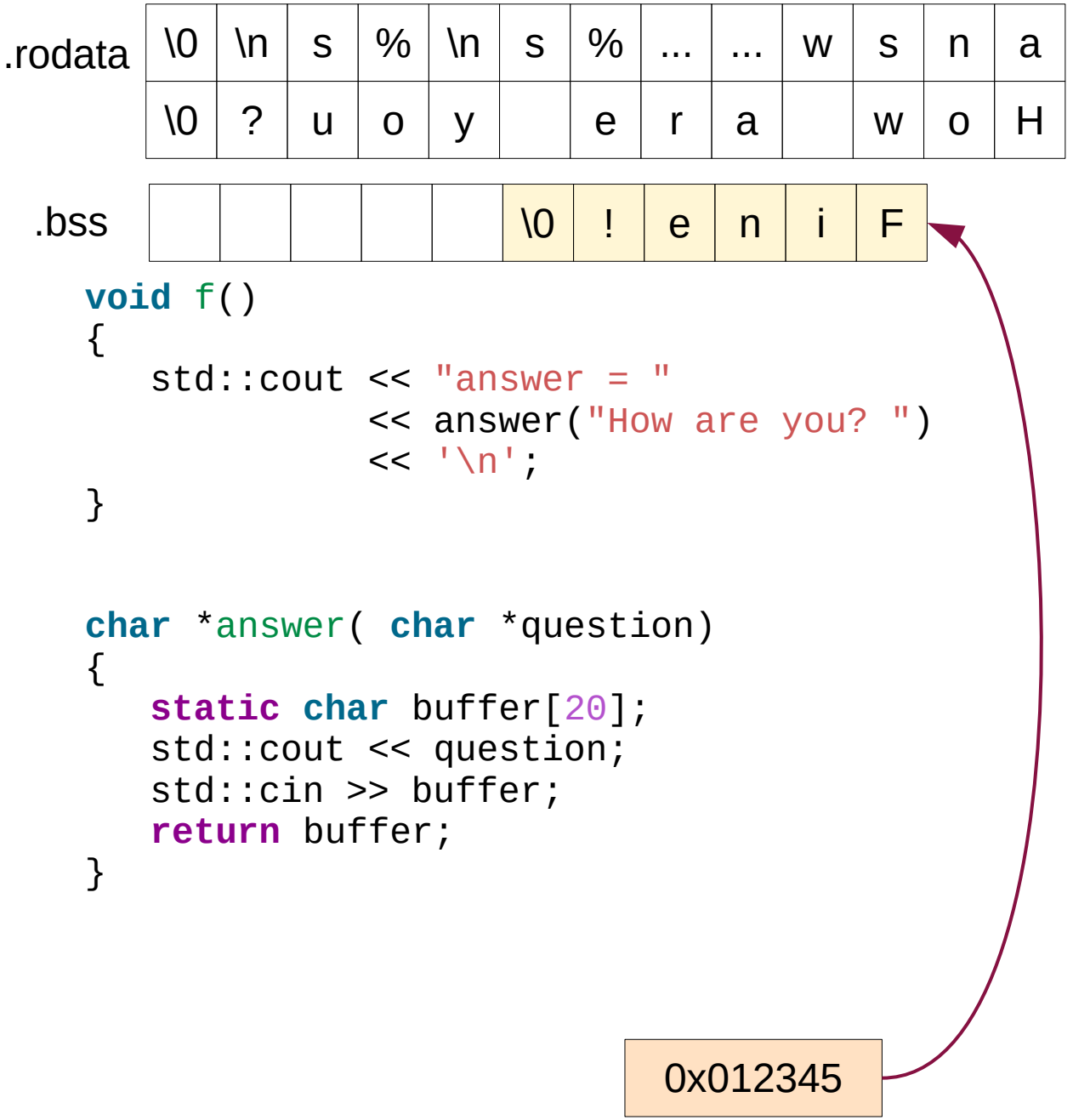

```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}

char buffer[20];
char *answer( char *question)
{

   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```

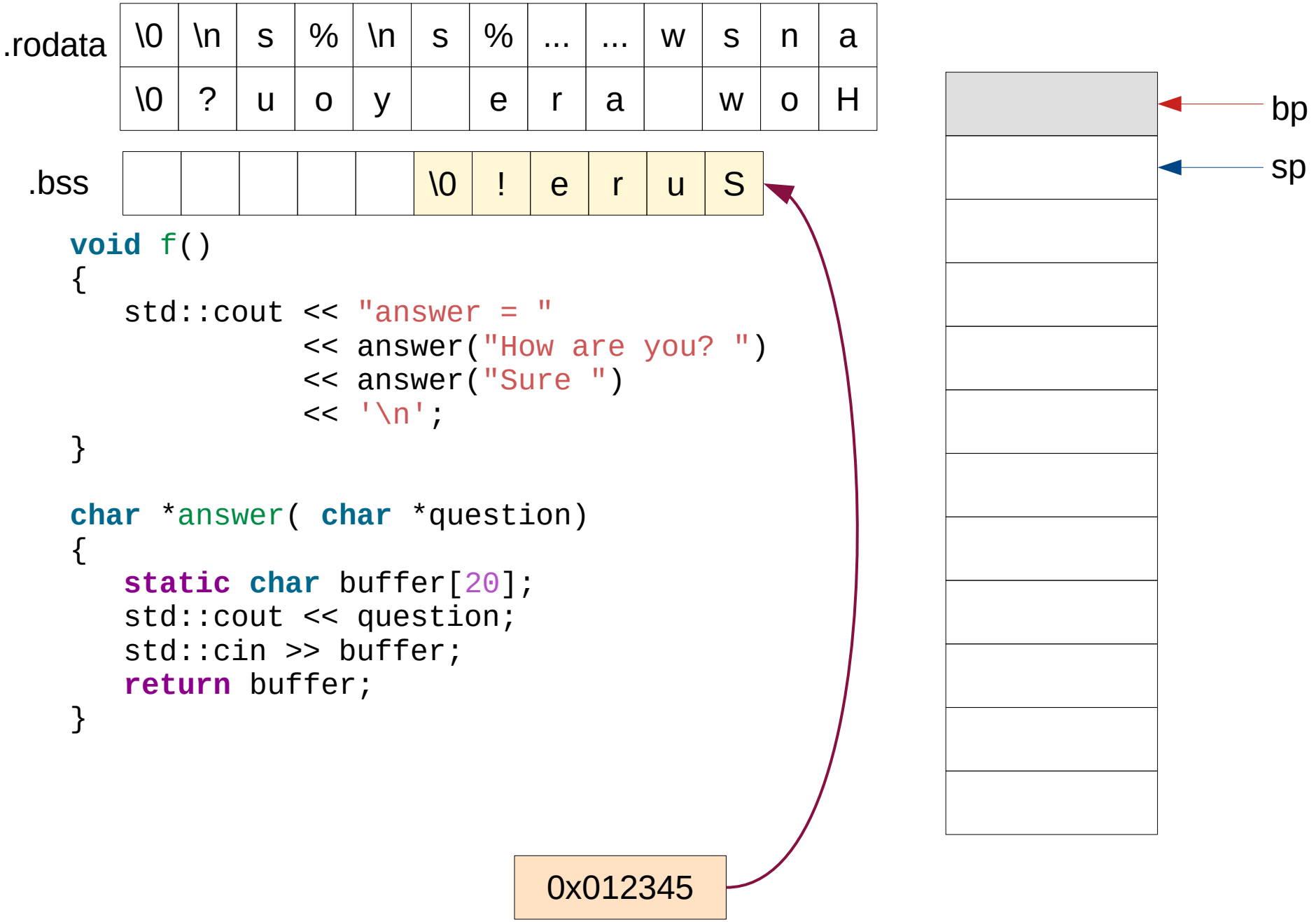
.rodata

\0	\n	s	%	\n	s	%	...	...	w	s	n	a
\0	?	u	o	y		e	r	a		w	o	H

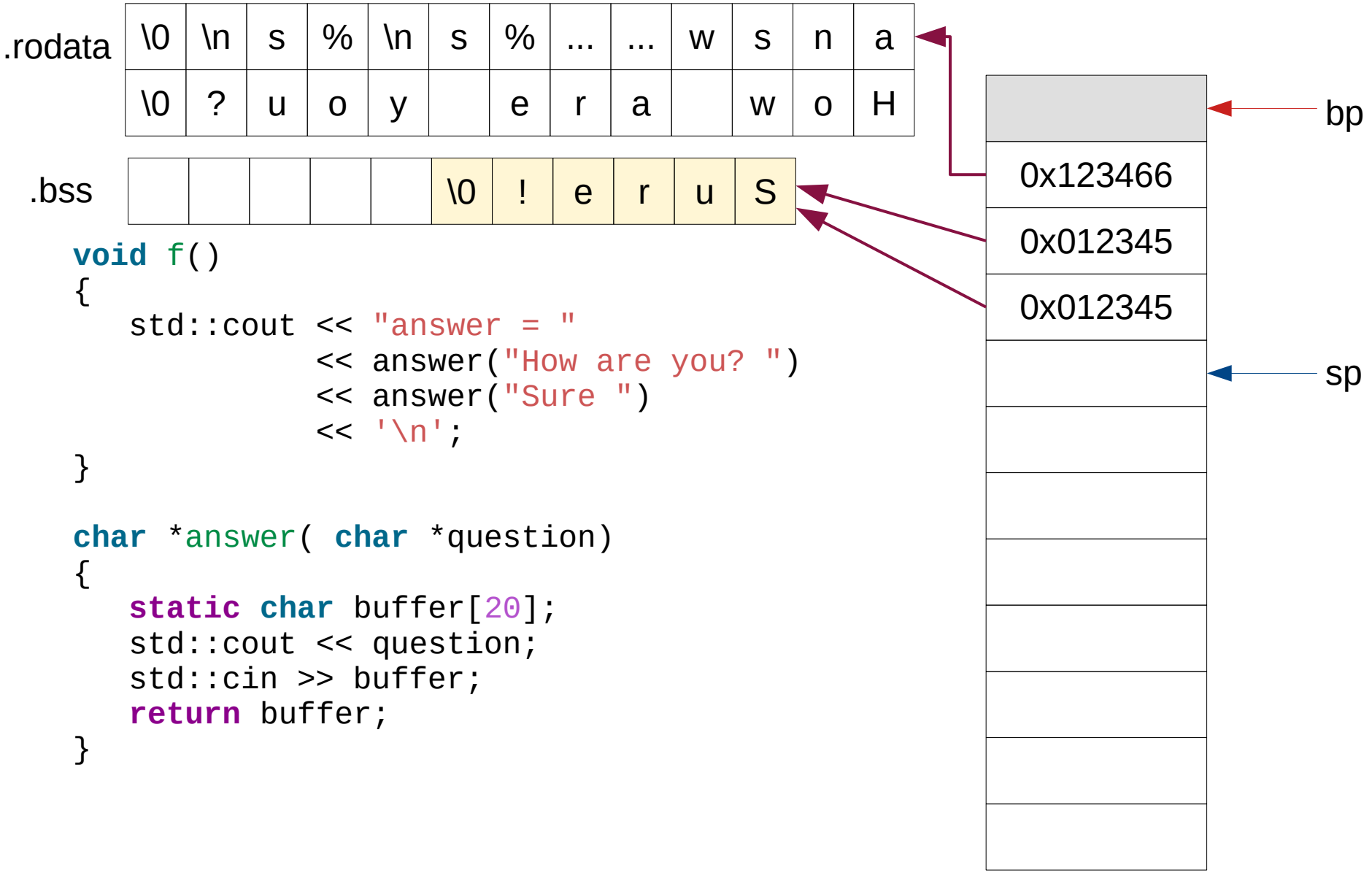
.bss


```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << '\n';
}

char buffer[20];
char *answer( char *question)
{
   static char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```


.rodata
\0 \n s % \n s % ... ... w s n a
\0 ? u o y e r a w o H
.bss
\0 ! e n i F
void f()
{
std::cout << "answer = "
<< answer("How are you? ")
<< '\n';
}
char *answer(char *question)
{
static char buffer[20];
std::cout << question;
std::cin >> buffer;
return buffer;
}
0x012345
bp
sp



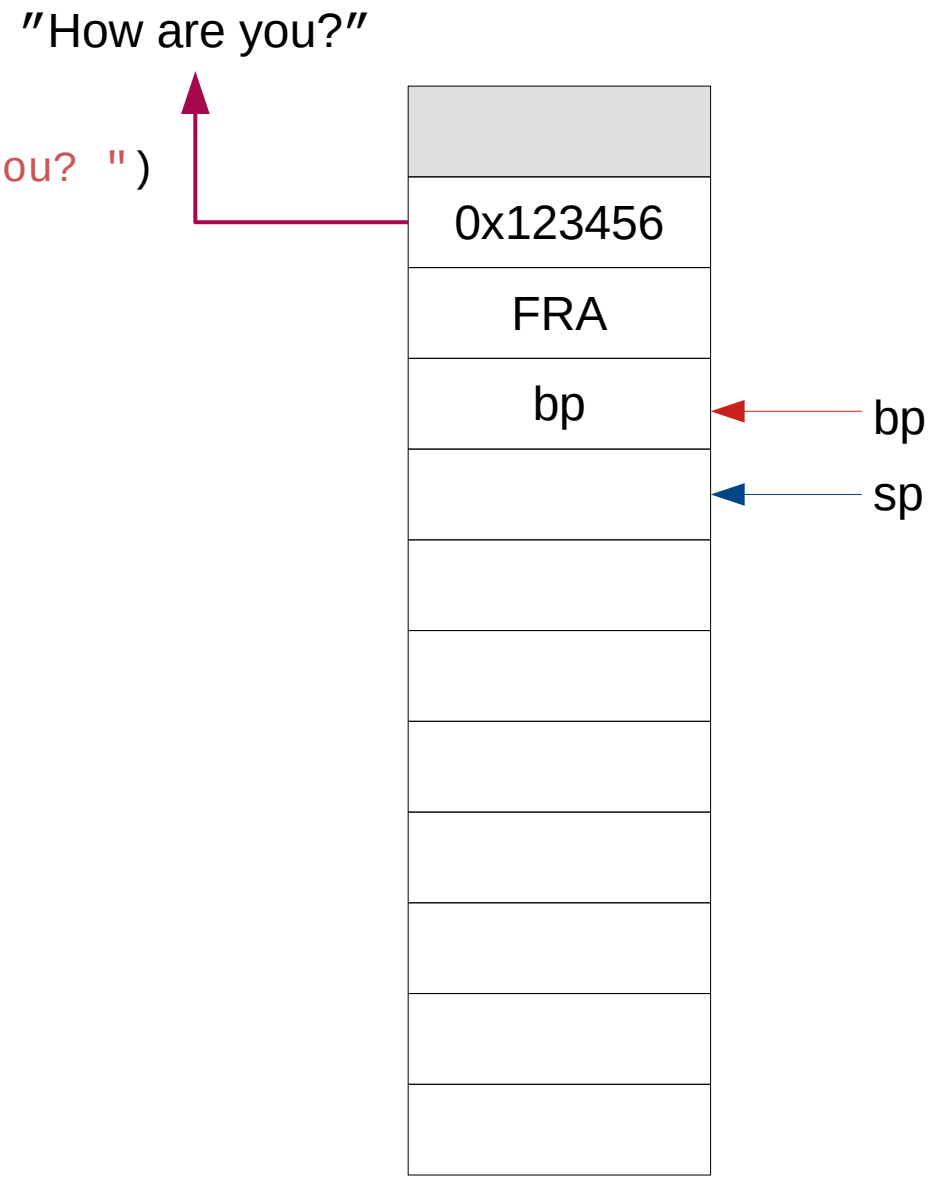

```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   static char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



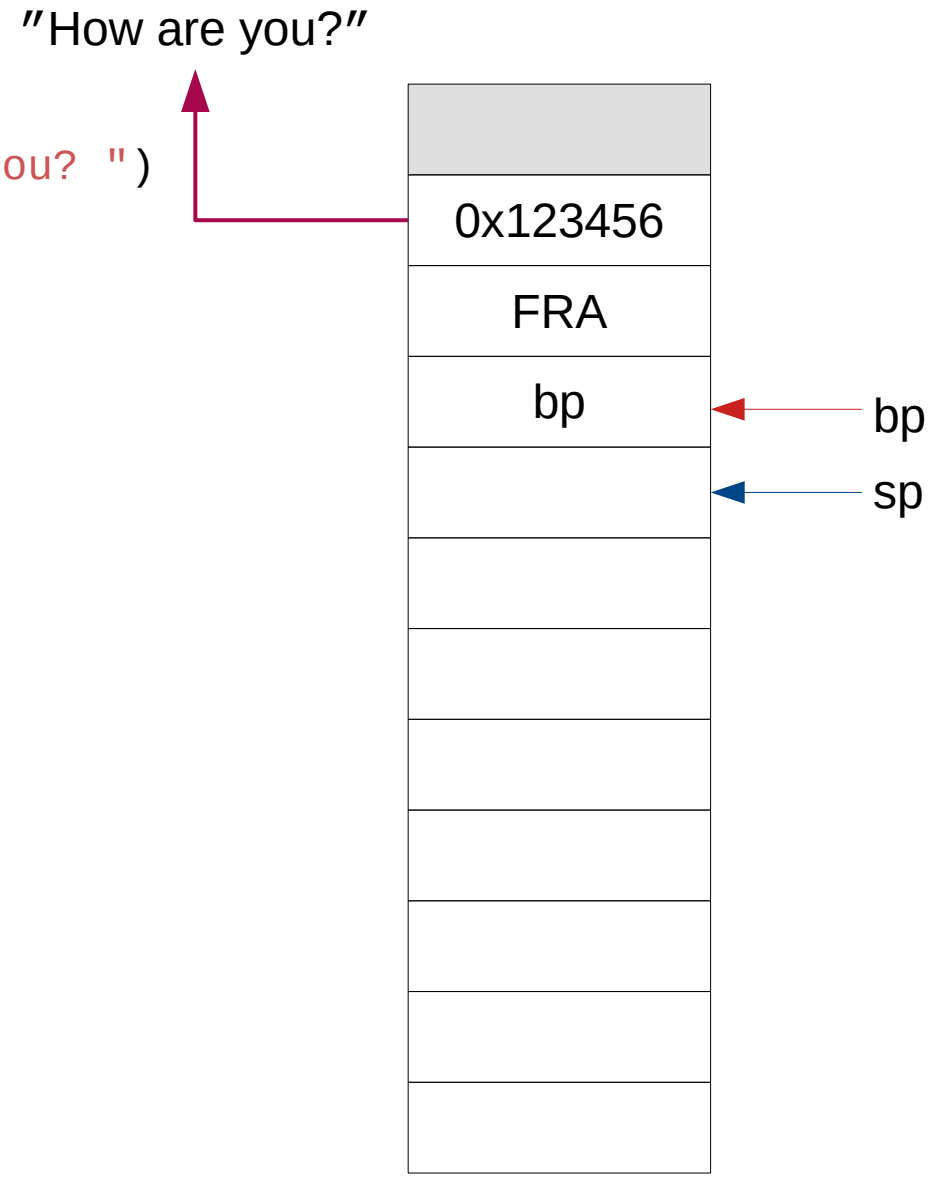

```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   static char buffer[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```

```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```

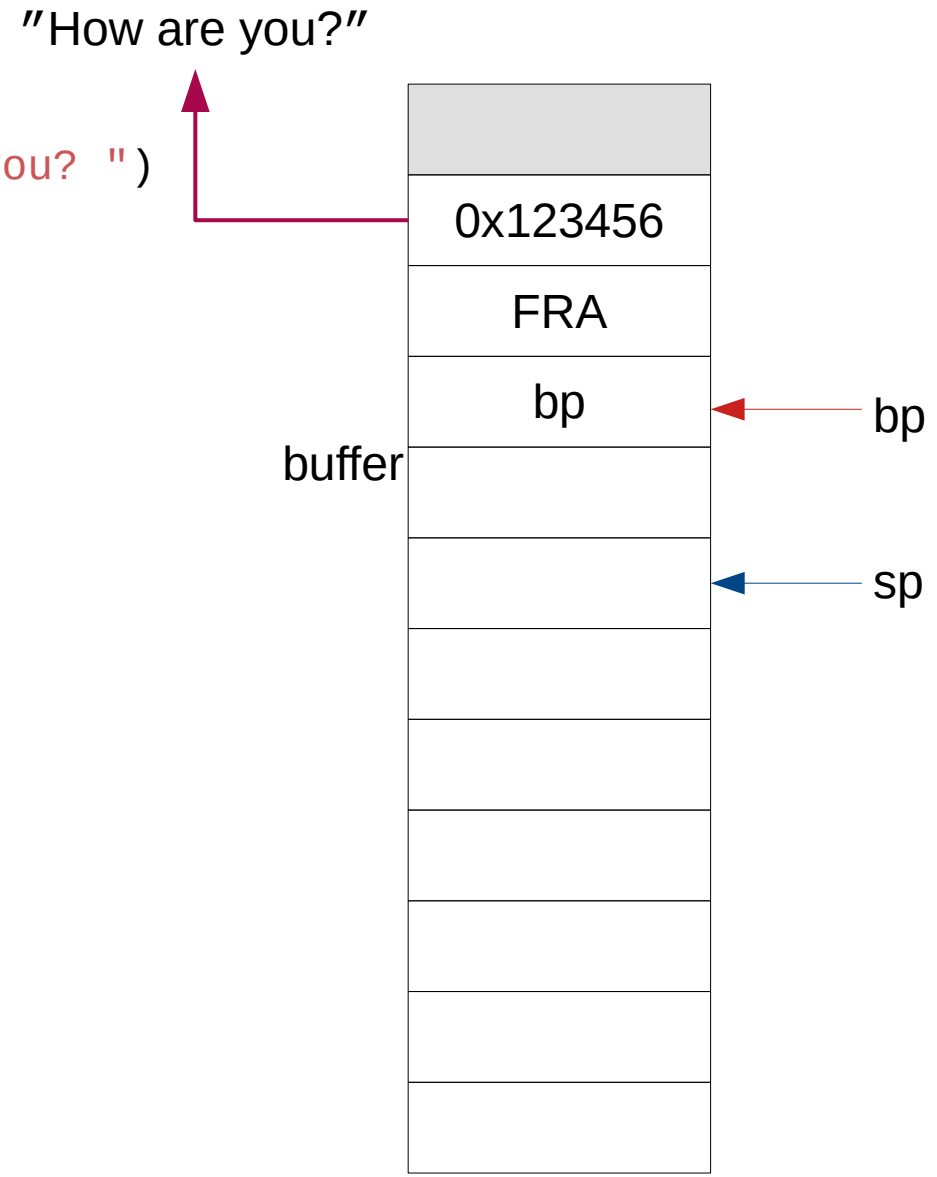
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



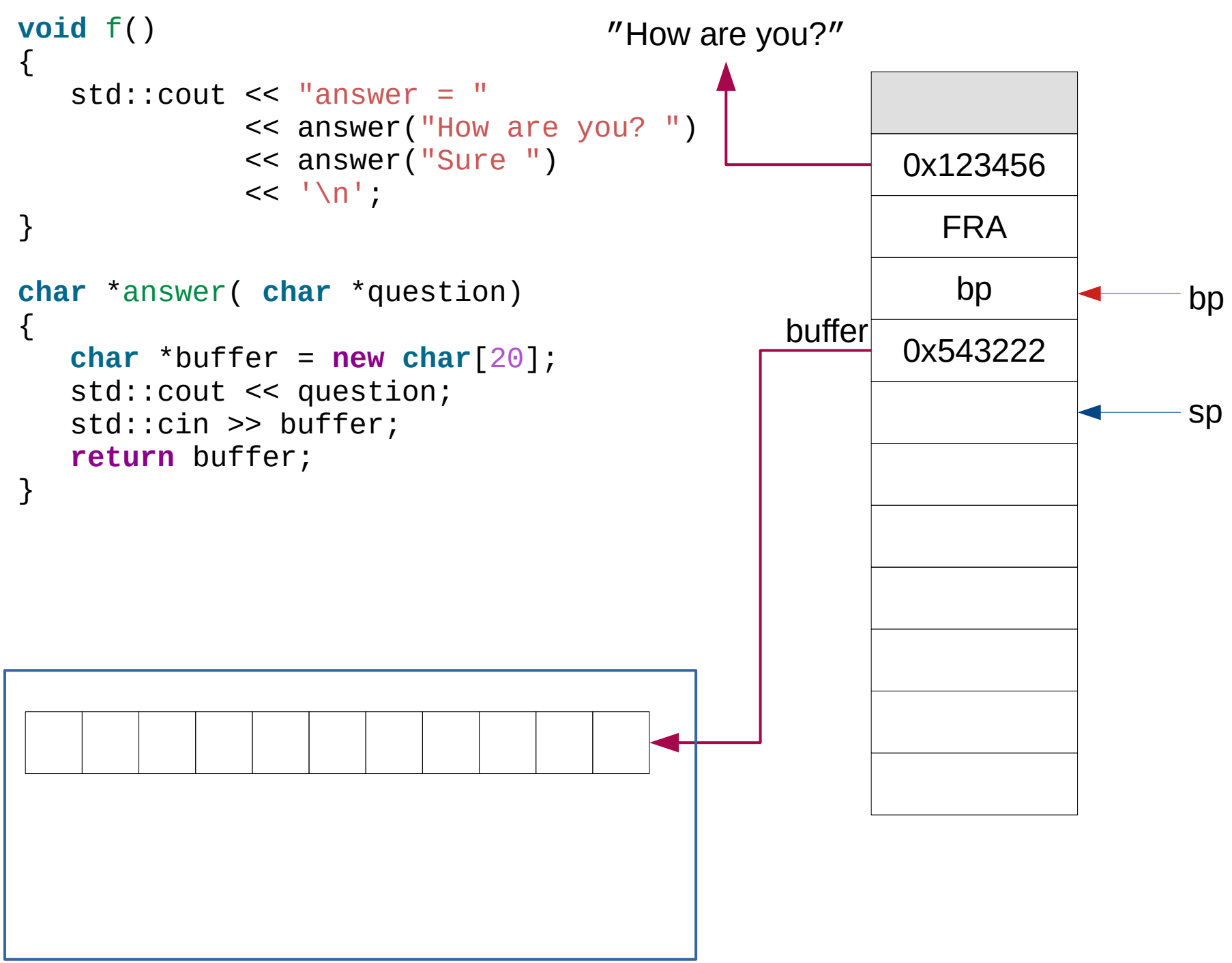
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



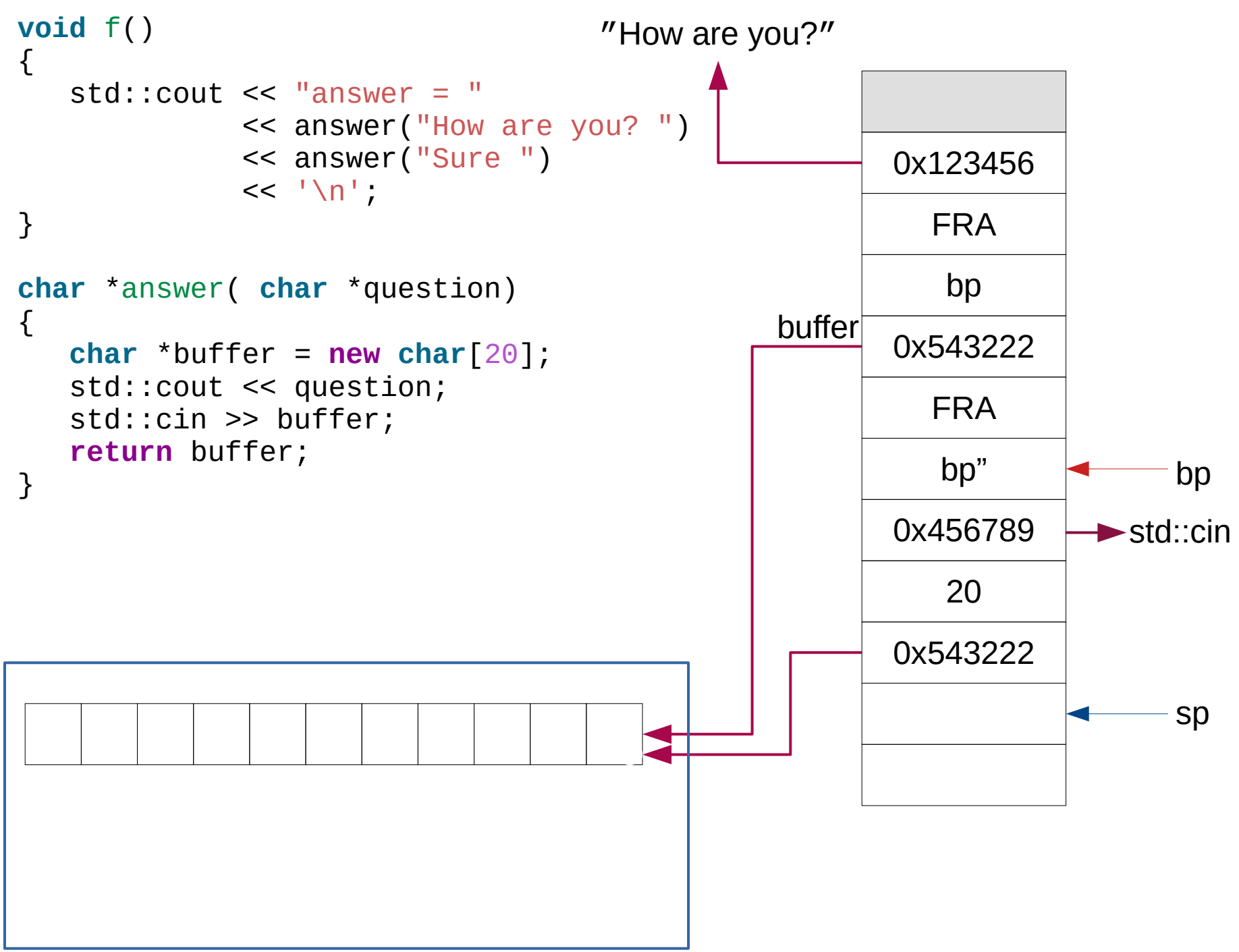
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



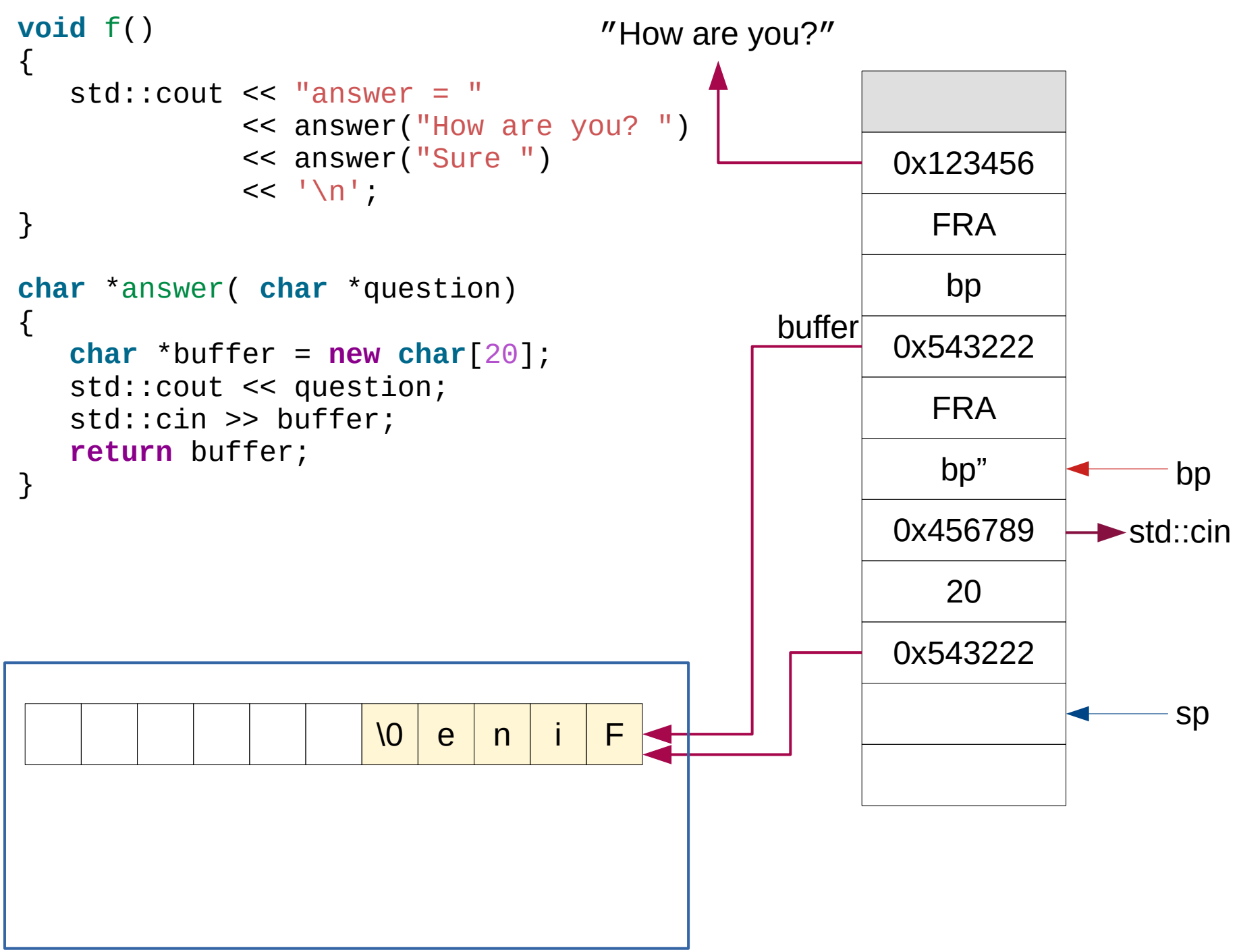
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



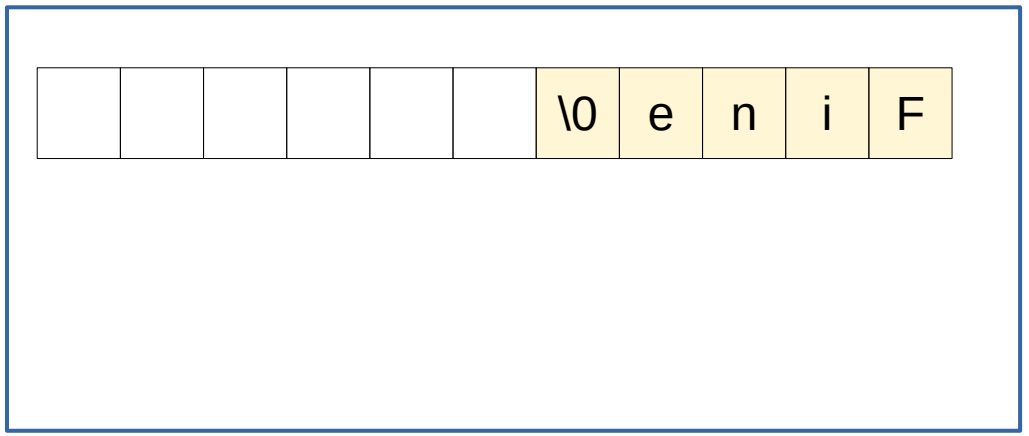
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```

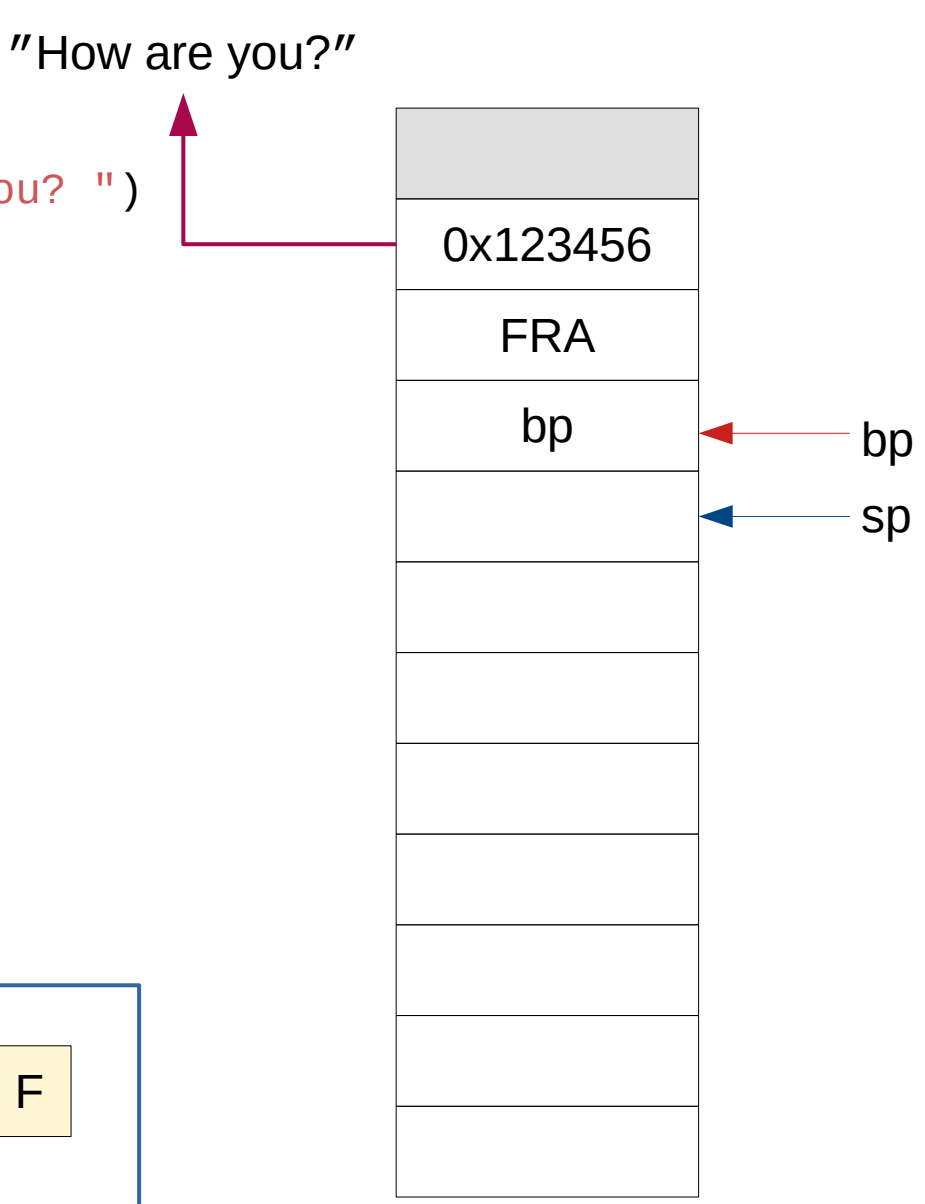


```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```

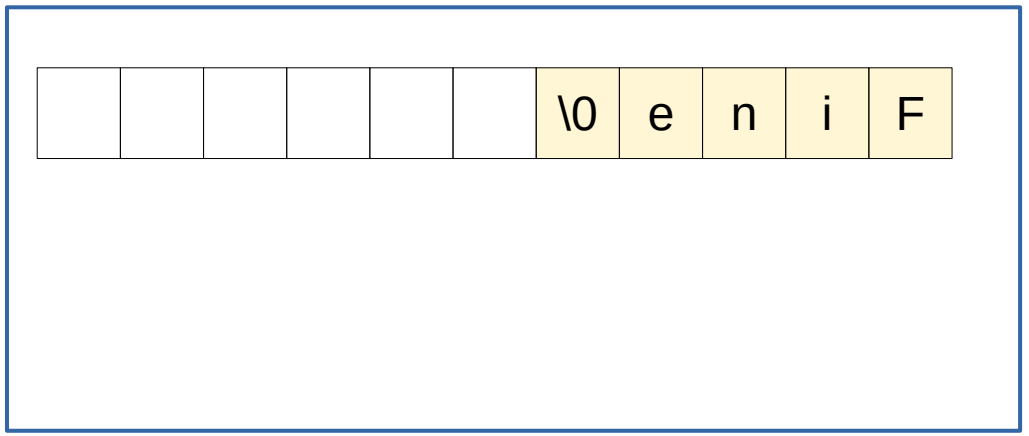


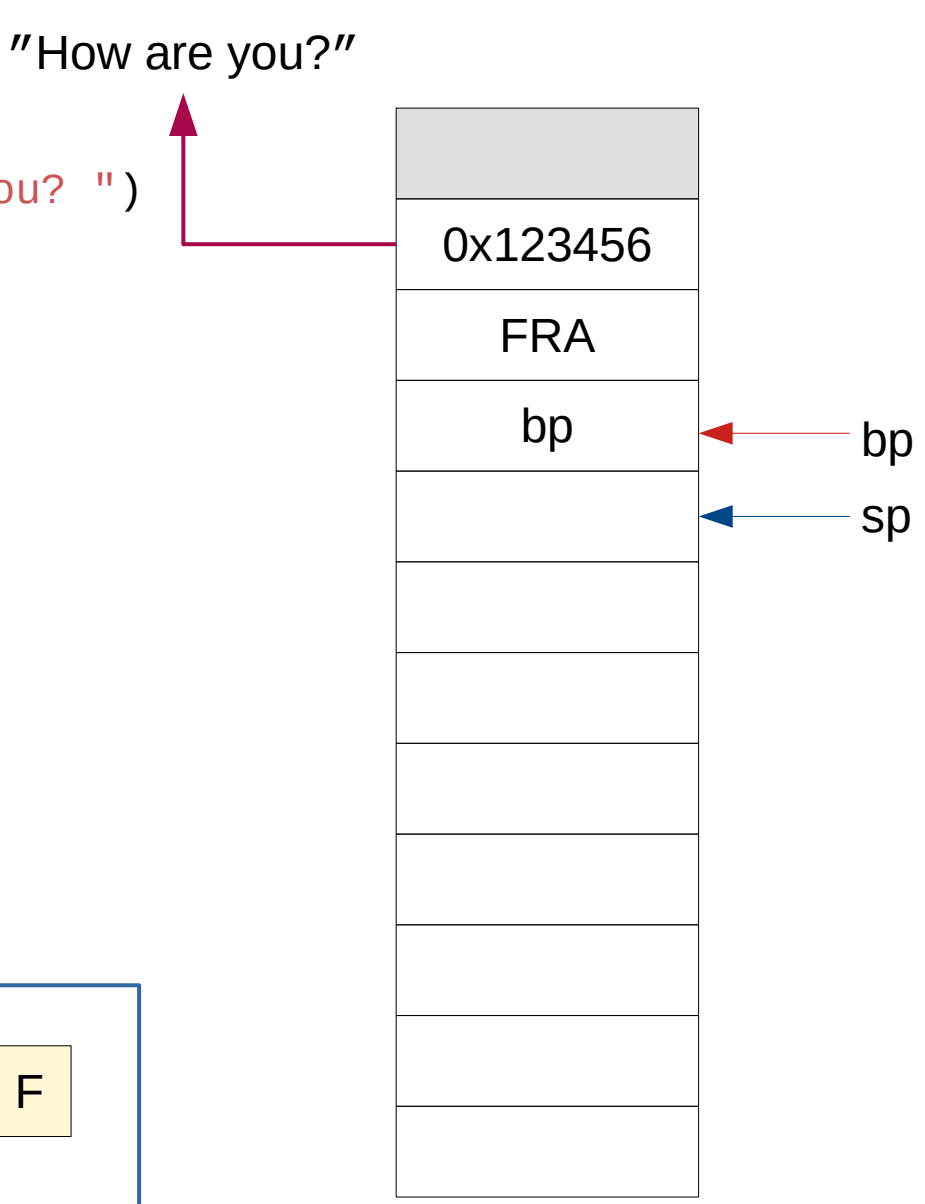
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



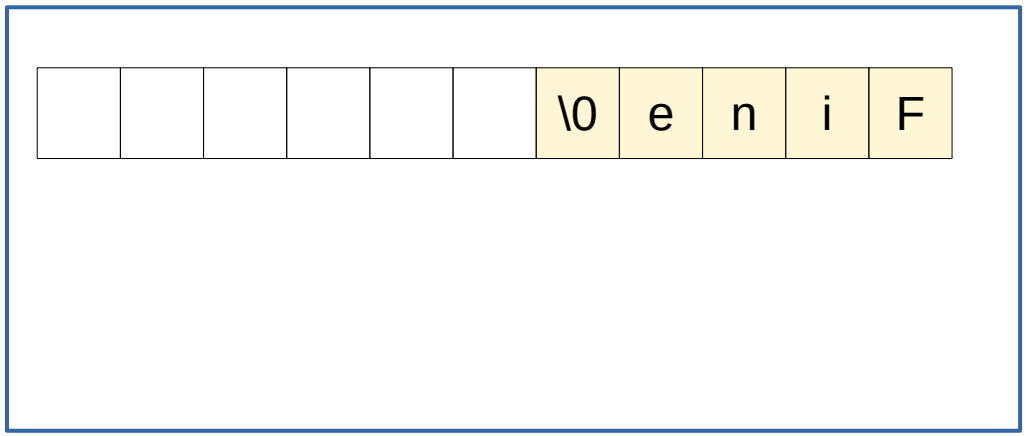


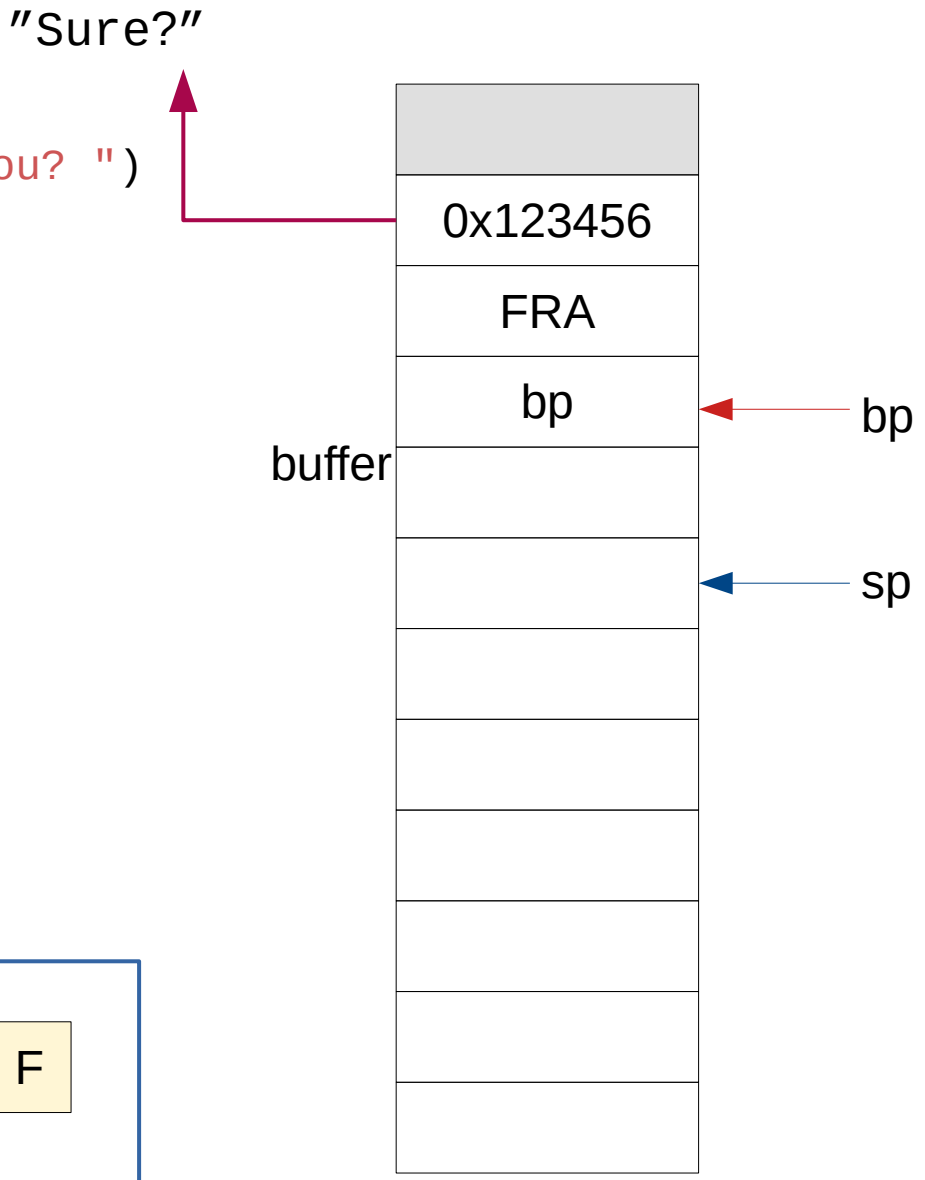
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



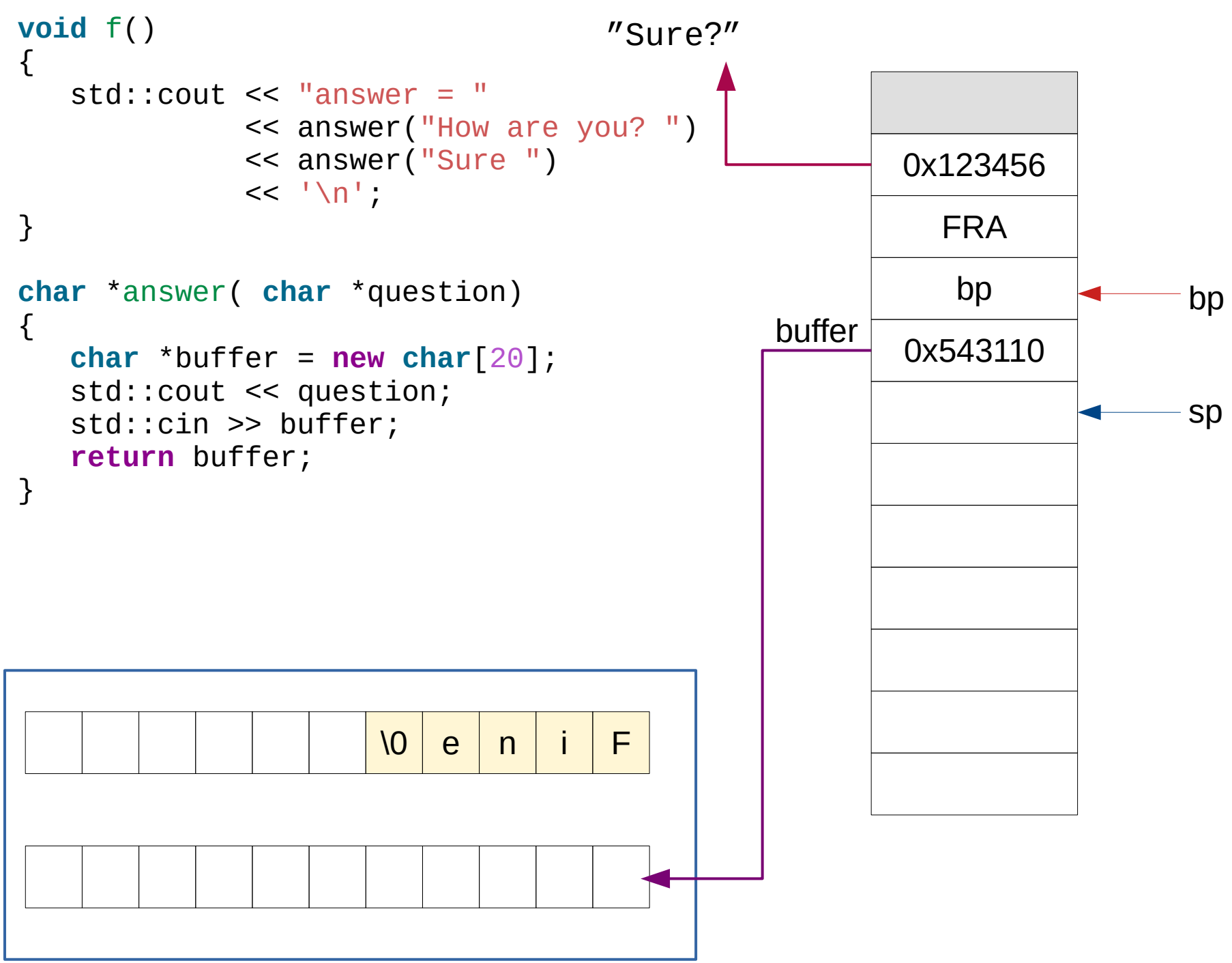


```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```

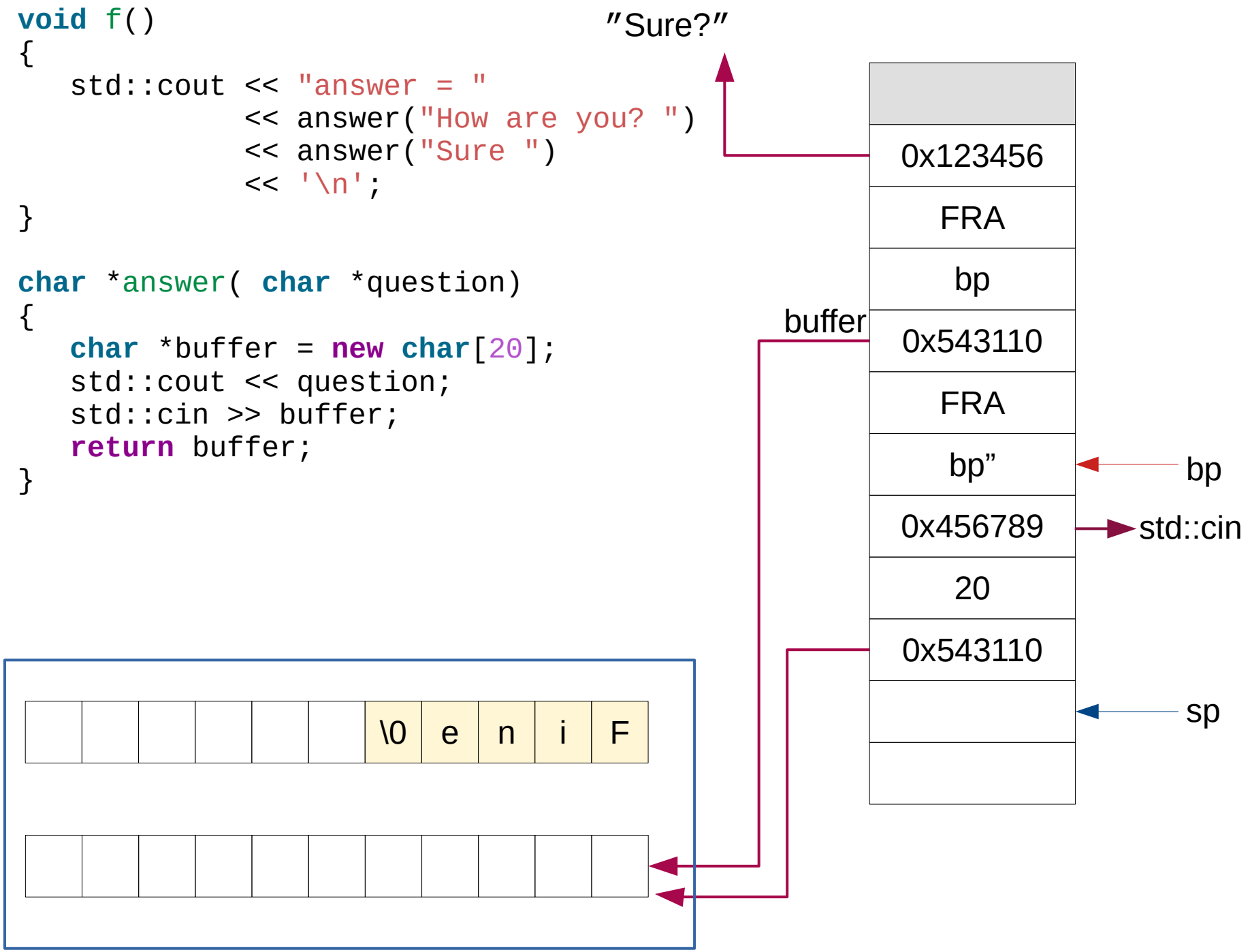




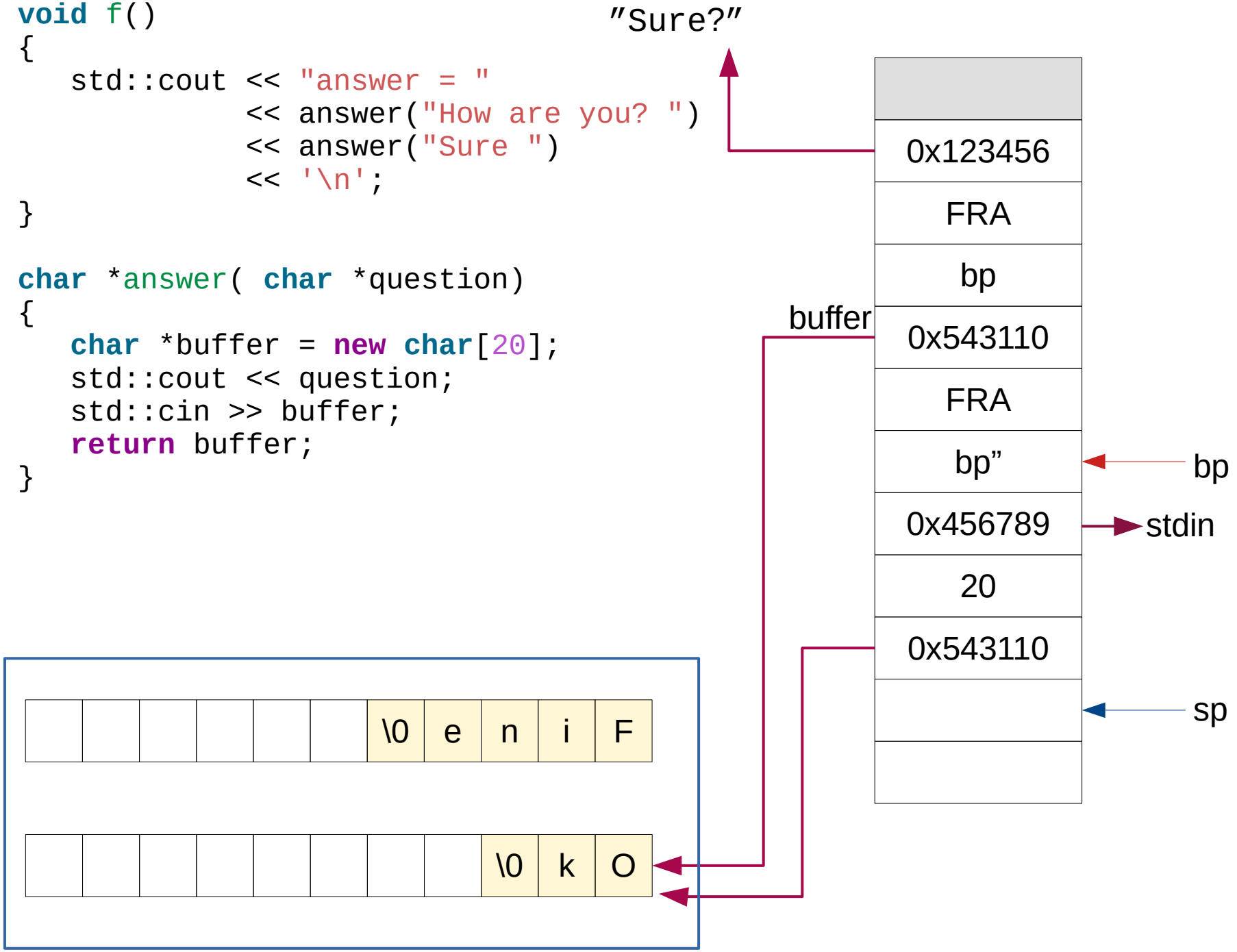
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



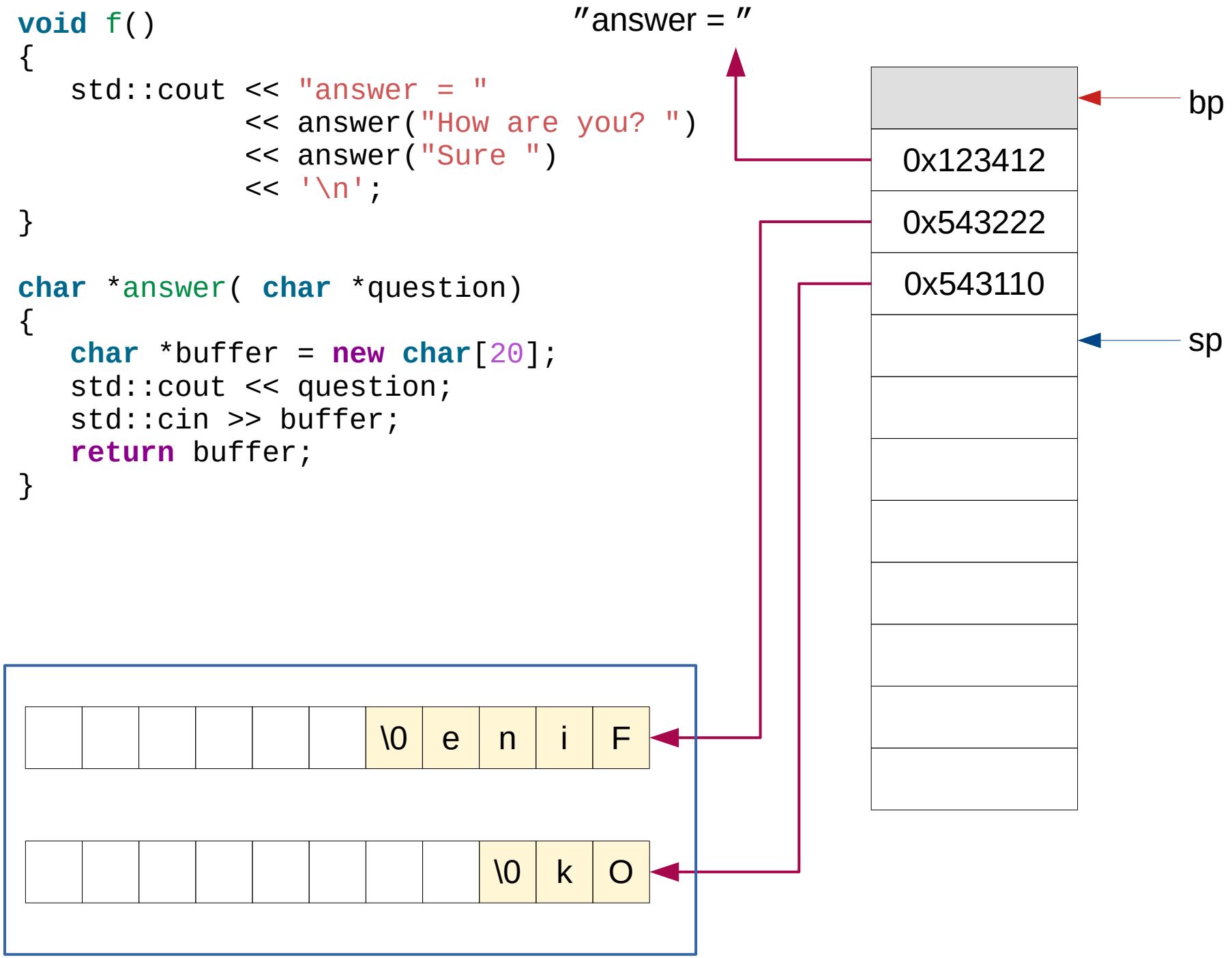
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



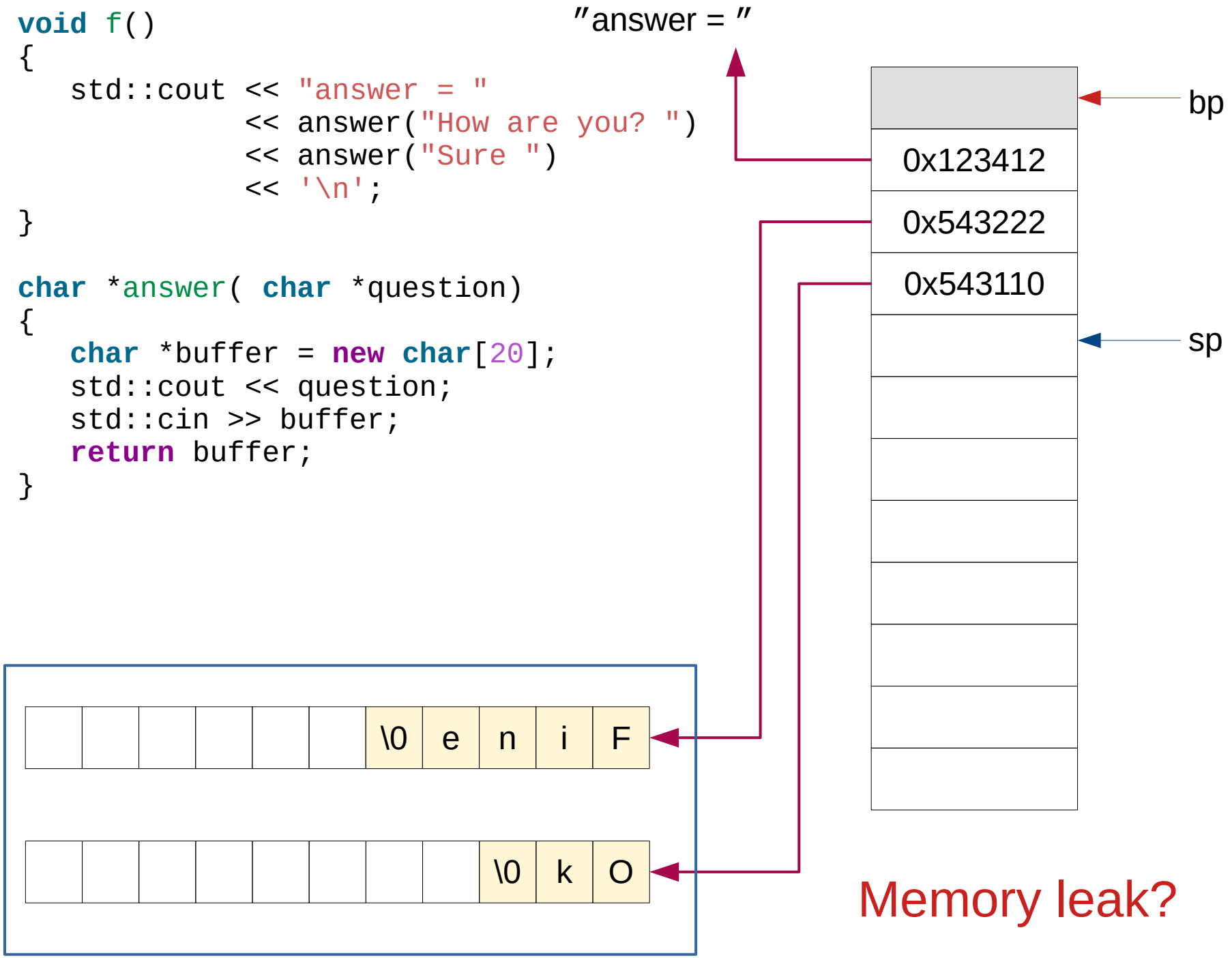
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



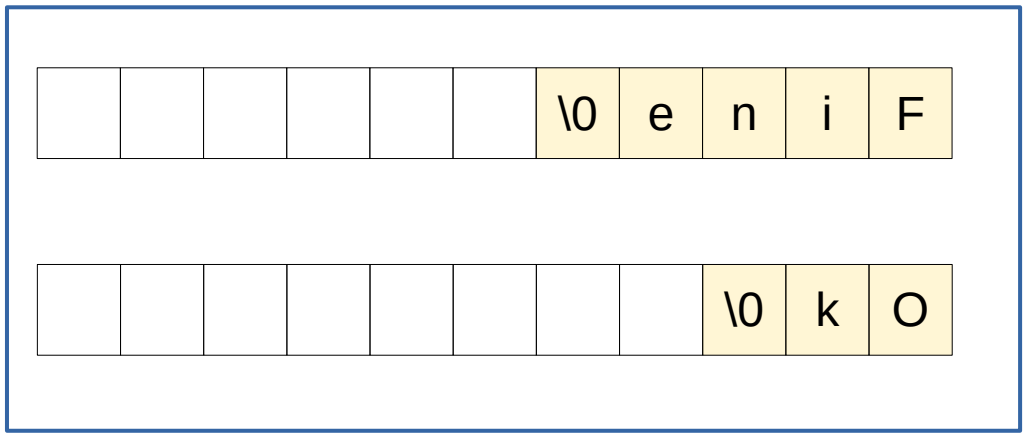
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



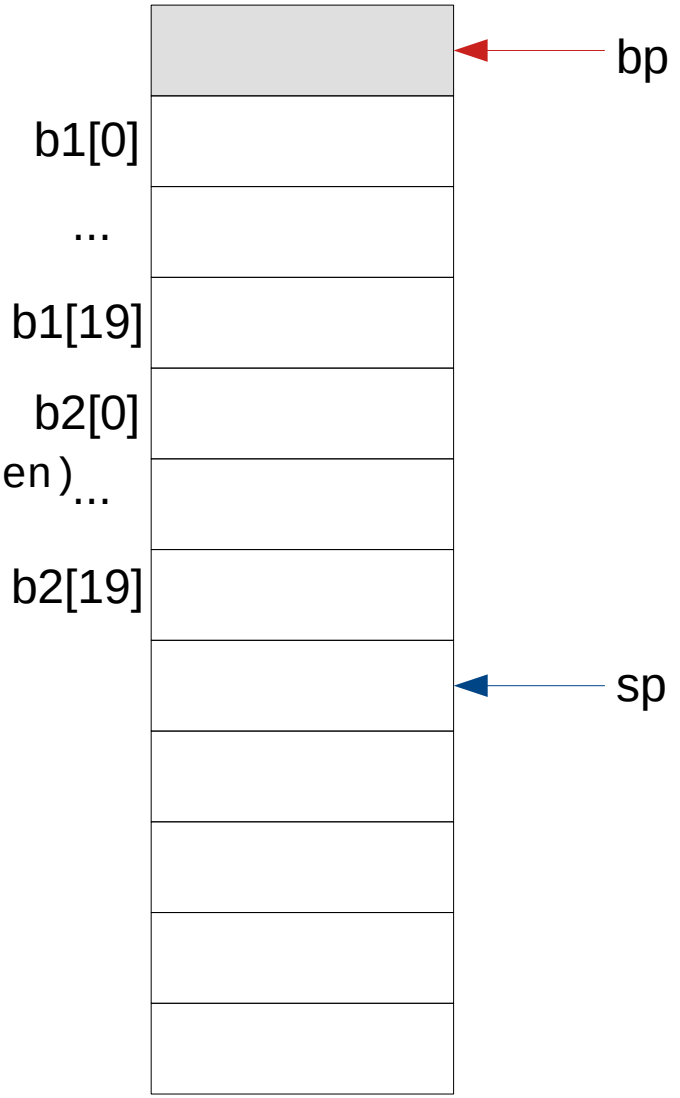
```
void f()
{
   std::cout << "answer = "
             << answer("How are you? ")
             << answer("Sure ")
             << '\n';
}

char *answer( char *question)
{
   char *buffer = new char[20];
   std::cout << question;
   std::cin >> buffer;
   return buffer;
}
```



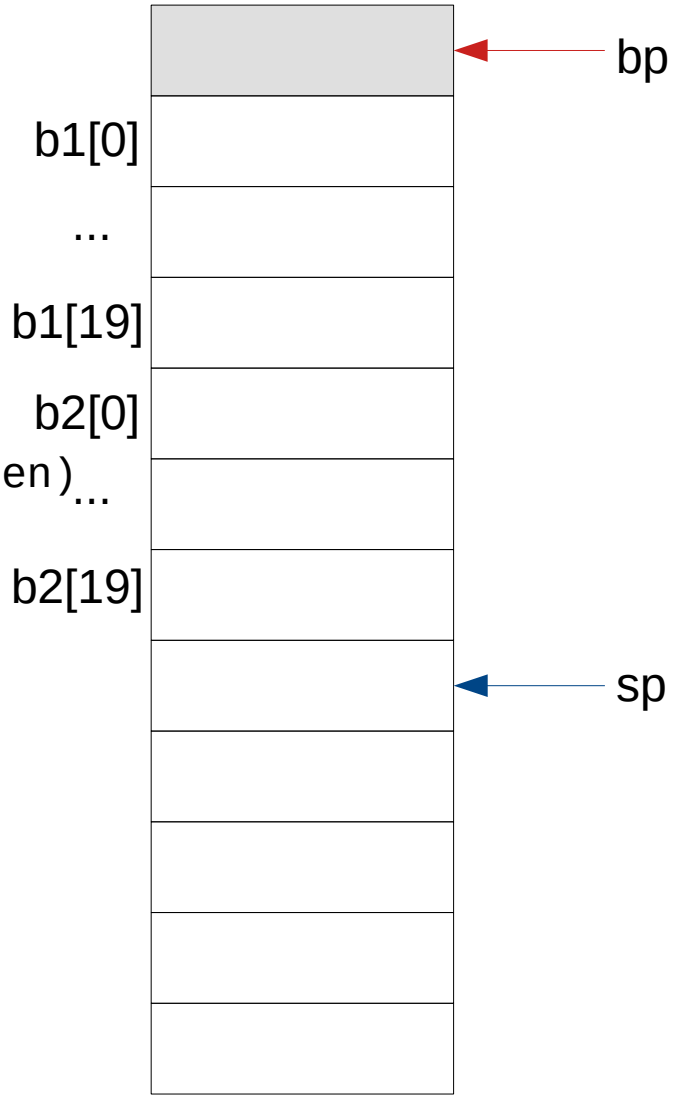
Memory leak?

```
void f()
{
    const int len = 80;
    char buffer1[len], buffer2[len];

    std::cout << "answer = "
     << answer("How are you? ",buffer1,len)
     << answer("Sure? ",buffer2,len)
     << '\n';
}

char *answer(char *question, char *b, int len)
{
   std::cout << question;
   std::cin >> std::setw(len-1) >> b;
   return b;
```

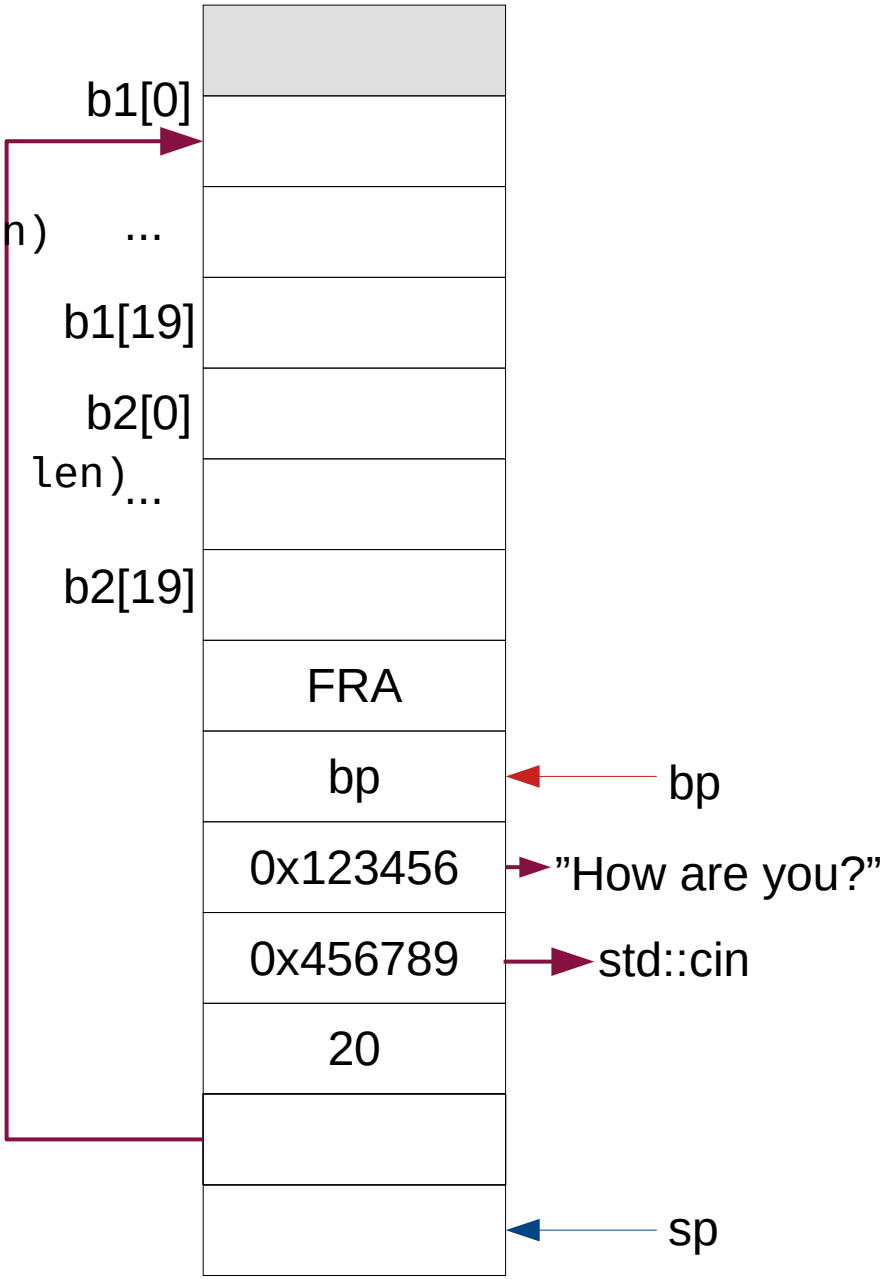
```
void f()
{
    const int len = 80;
    char buffer1[len], buffer2[len];

    std::cout << "answer = "
     << answer("How are you? ",buffer1,len)
     << answer("Sure? ",buffer2,len)
     << '\n';
}

char *answer(char *question, char *b, int len)
{
   std::cout << question;
   std::cin >> std::setw(len-1) >> b;
   return b;
}
```



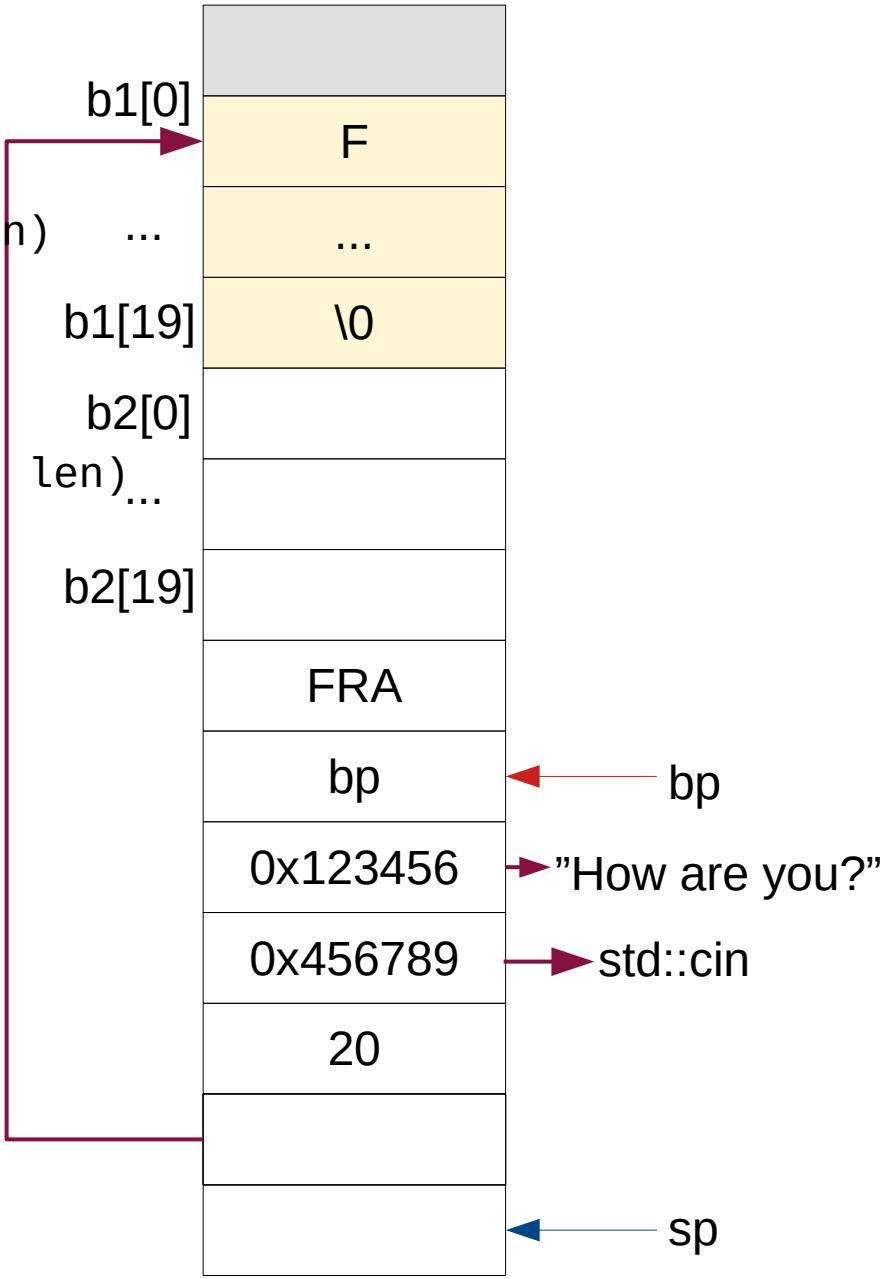
```
void f()
{
    const int len = 80;
    char buffer1[len], buffer2[len];

    std::cout << "answer = "
     << answer("How are you? ",buffer1,len)
     << answer("Sure? ",buffer2,len)
     << '\n';
}

char *answer(char *question, char *b, int len)
{
   std::cout << question;
   std::cin >> std::setw(len-1) >> b;
   return b;
}
```



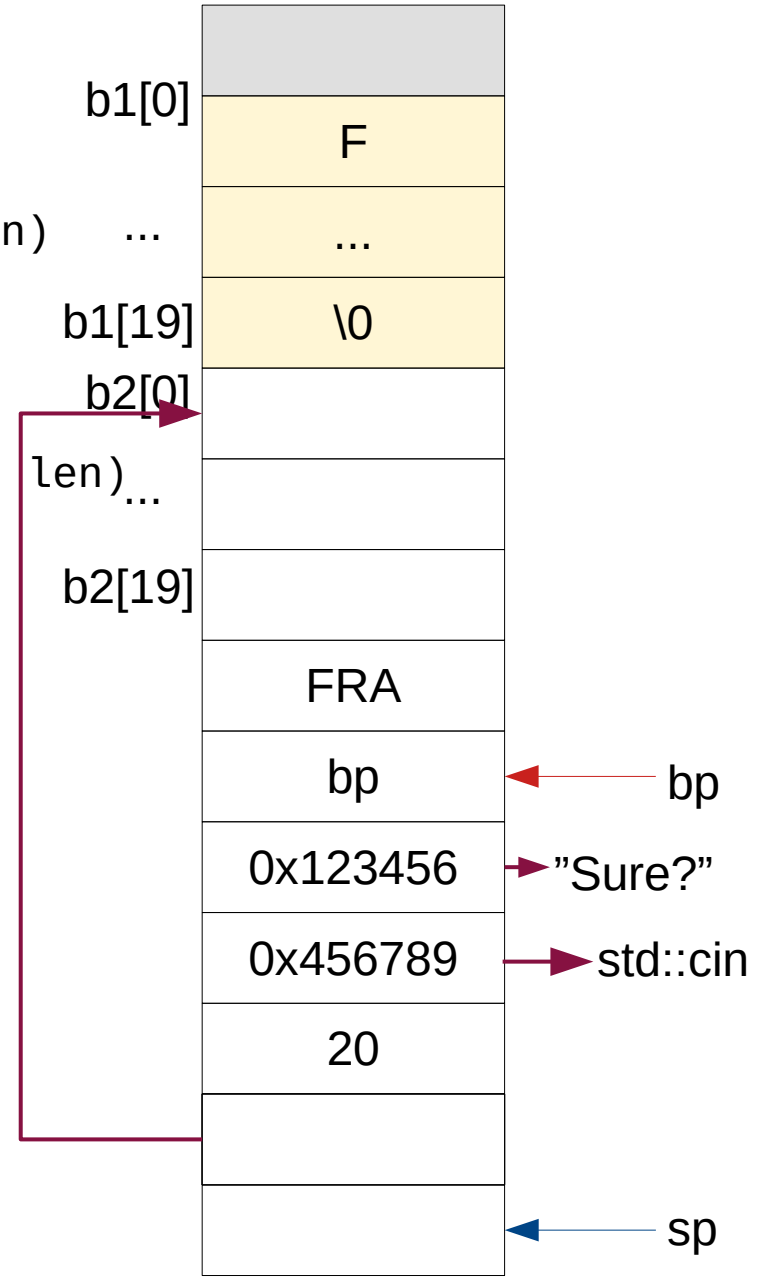
```
void f()
{
    const int len = 80;
    char buffer1[len], buffer2[len];

    std::cout << "answer = "
     << answer("How are you? ",buffer1,len)
     << answer("Sure? ",buffer2,len)
     << '\n';
}

char *answer(char *question, char *b, int len)
{
   std::cout << question;
   std::cin >> std::setw(len-1) >> b;
   return b;
}
```



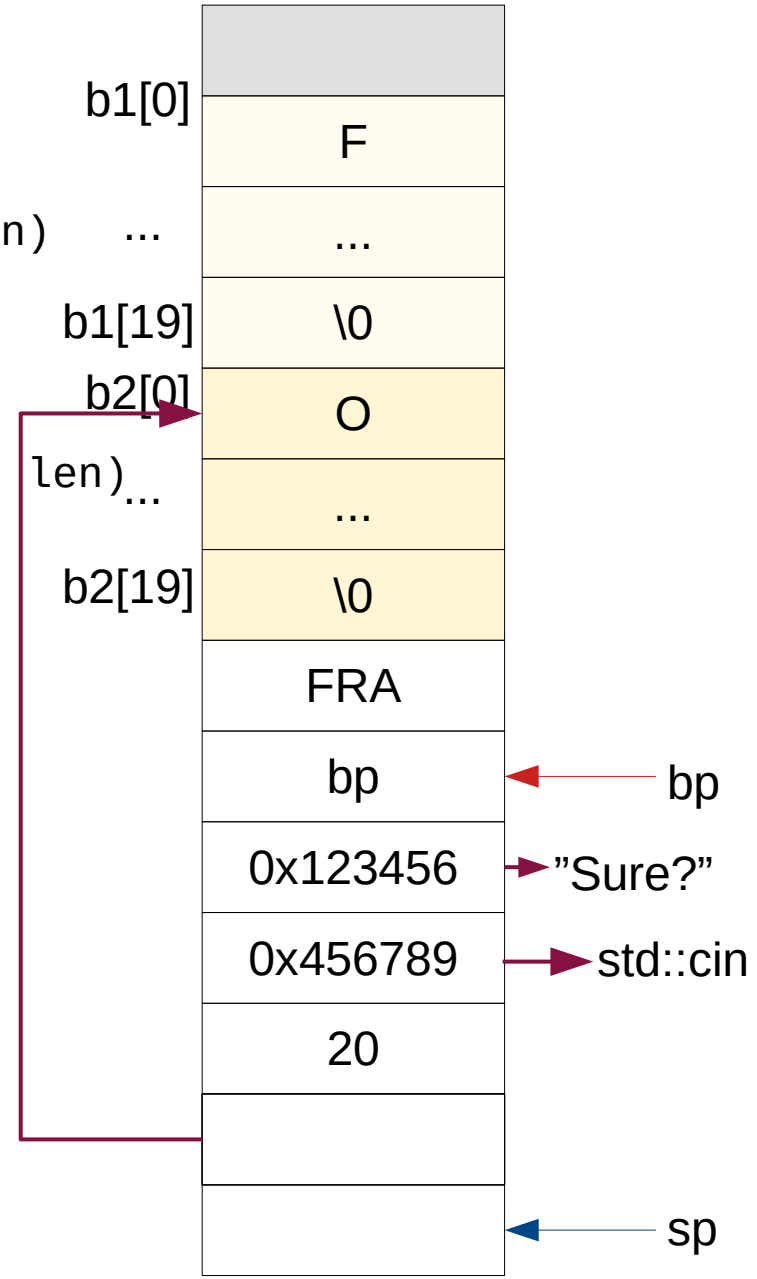
```
void f()
{
    const int len = 80;
    char buffer1[len], buffer2[len];

    std::cout << "answer = "
     << answer("How are you? ",buffer1,len)
     << answer("Sure? ",buffer2,len)
     << '\n';
}

char *answer(char *question, char *b, int len)
{
   std::cout << question;
   std::cin >> std::setw(len-1) >> b;
   return b;
}
```



```
void f()
{
    const int len = 80;
    char buffer1[len], buffer2[len];

    std::cout << "answer = "
     << answer("How are you? ",buffer1,len)
     << answer("Sure? ",buffer2,len)
     << '\n';
}

char *answer(char *question, char *b, int len)
{
   std::cout << question;
   std::cin >> std::setw(len-1) >> b;
   return b;
}
```



```
void f()
{
    const int len = 80;
    char buffer1[len], buffer2[len];

    std::cout << "answer = "
     << answer("How are you? ",buffer1,len)
     << answer("Sure? ",buffer2,len)
     << '\n';
}

char *answer(char *question, char *b, int len)
{
   std::cout << question;
   std::cin >> std::setw(len-1) >> b;
   return b;
}
```



C++ -like solution

```
#include <string>
#include <iostream>
#include <iomanip>

void f()
{
    std::cout << "answer = "
              << answer("How are you? ")
              << answer("Sure? ")
              << '\n';
}

std::string answer(std::string question)
{
    std::cout << question;
    std::string answ;
    std::cin >> answ;
    return answ;
}
```

C++ -like solution

```
#include <string>
#include <iostream>
#include <iomanip>

void f()
{
    std::cout << "answer = "
              << answer("How are you? ")  // the evaluation order
              << answer("Sure? ")         // is still not defined
              << '\n';
}

std::string answer(std::string question)
{
    std::cout << question;
    std::string answ;
    std::cin >> answ;     // this can be still a target for DoS attack
    return answ;
}
```

C++ -like solution

```
#include <string>
#include <iostream>
#include <iomanip>

void f()
{
    std::cout << "answer = ";
    std::cout << answer("How are you? ");  // the evaluation order
    std::cout << answer("Sure? ");         // is now defined
              << '\n';
}

std::string answer(std::string question)
{
    std::cout << question;
    std::string answ;
    std::cin >> setw(80) >> answ;    // reads max 80 characters
    return answ;
}
```