

Data Structures

Function Calls in C
Dynamic Memory
Abstract Data Types

Function Calls in C

C passes arguments to functions by value (*call-by-value*).

For that reason, the called function cannot alter the value of a variable in the calling program.

2 ITÜ, BLG221E Data Structures, G. Eryigit, S. Kabaday © 2011

Function Calls in C

- If the function has to make a change to a single value, this value can be passed back to the calling program with a *return* statement.

```
int add(int a,int b){
    .....
    return sum;
}
int result=add(5,7);
```

What if the function has to change more than one value???

3 ITÜ, BLG221E Data Structures, G. Eryigit, S. Kabaday © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Erroneous program: function only swaps copies of a and b sent to it.

```
void swap(int x,int y){
    int temp;
    temp=x;
    x=y;
    y=temp;
}
int main(int argc, char* argv[])
{
    int a=5,b=6;
    swap(a,b);
    return 0;
}
```

4 ITÜ, BLG221E Data Structures, G. Eryigit, S. Kabaday © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Correct Call: swap(&a,&b)

```
int main(int argc, char* argv[])
{
    int a=5,b=6;
    swap(&a,&b);
    return 0;
}
void swap(int *px,int *py){
    int temp;
    temp=*px;
    *px=*py;
    *py=temp;
}
```

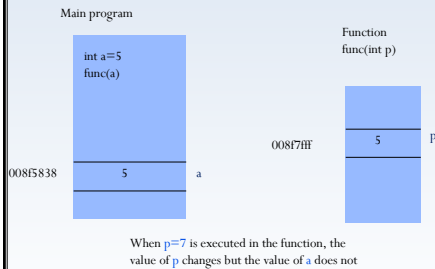
5 ITÜ, BLG221E Data Structures, G. Eryigit, S. Kabaday © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Call-by-Value vs. Call-by-Reference

- When an argument is passed *call-by-value*, a *copy* of the argument's value is made and passed to the called function.
- With *call-by-reference*, the addresses of the parameters are passed to the function. Thus, the function is given direct access to these parameters.

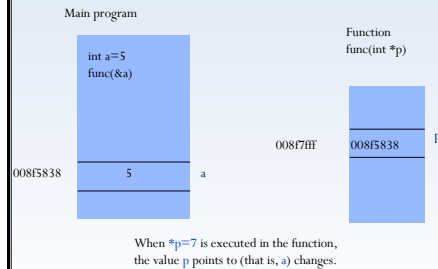
6 ITÜ, BLG221E Data Structures, G. Eryigit, S. Kabaday © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Call-by-Value



7 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabaday © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Call-by-Reference



8 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabaday © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Arrays As Arguments

- When an array name is passed to a function, what is passed is the address of the first element of the array.
- Within the called function, this argument is a local variable, and so an array name parameter is a pointer, that is, a variable containing an address.
- We can use this information as a pointer.**

9 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabaday © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Example

```
int strlen(char *s){
    int n = 0;
    for (n = 0; *s != '\0'; s++)
        n++;
    return n;
}

int main(int argc, char* argv[])
{
    char array[10] = "array";
    int k;
    char *ptr = array;
    k = strlen("hello world");
    k = strlen(array);
    k = strlen(ptr);
    return 0;
}
```

example0.c

10 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabaday © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Example

All strings end with the null ('`\0`') character. This value corresponds to "false".

```
int strlen(char *s){
    int n = 0;
    while (*s){
        s++;
        n++;
    }
    return n;
}

int main(int argc, char* argv[])
{
    char array[10] = "array";
    int k;
    char *ptr = array;
    k = strlen("hello world");
    k = strlen(array);
    k = strlen(ptr);
    return 0;
}
```

example0.c

11 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabaday © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Arrays as Arguments

- When an array name is passed to a function, the function can use this information as either an array or a pointer.
- It is also possible to pass part of an array to a function, by passing a pointer to the beginning of the subarray.

```
f(&a[2]);
f(a+2);
```

Both pass to the function `f` the address of the subarray beginning with the 3. element of array `a`

12 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabaday © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Example

```

int strlen(char *s){
    int n = 0;
    for (n = 0; *s != '\0'; s++)
        n++; for (n=0; s[n] != '\0'; n++);
    return n;
}

int main(int argc, char* argv[])
{
    char array[10] = "array";
    int k;
    char *ptr = array;
    k = strlen("hello world");
    k = strlen(&array[2]);
    k = strlen(ptr);
    return 0;
}

```

13 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadaş © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Pointer to a Pointer as an Argument

- If we want to change the value of an argument, we pass the address of the argument to the function.
- Similarly, if we want to change the value of a pointer, we pass the address of the pointer to the function (that is, we must pass a pointer to a pointer.)

14 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadaş © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

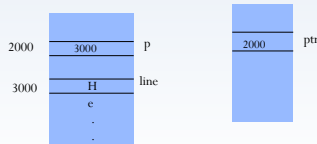
Pointer to a Pointer as an Argument

```

void f(char **ptr){
    (*ptr)++;
}

int main(int argc, char* argv[])
{
    char line[20] = "Hello world";
    char *p = line;
    f(&p);
    return 0;
}

```



15 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadaş © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Dynamic Memory

16

Dynamic Memory Allocation

- At execution time, a program can obtain and use new memory space. Memory space obtained at execution time can be returned to the system when it is no longer needed.
- This ability is known as **dynamic memory allocation**.
- The limit on the size of the memory space that can be allocated is determined by how much the operating system allows. (You will learn about memory management in operating systems in courses to come.)
- In C, functions `malloc` and `free` are essential to manipulating dynamic memory space.
- In C++, the essential functions are the `new` and `delete` operators.
- In this course, we will use `new` and `delete`.

17 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadaş © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Pointers and Dynamic Memory Allocation

- It is necessary to use pointers to dynamically request memory space from the system.
- Since the program is compiled and running at that moment, it is not possible to give a symbolic name to the memory space to be allocated.


```

int integer1;
int *dnumber;

```
- `integer1` is a symbolic name and the name of an integer. This name can be used where the variable is defined (in scope).
- `dnumber` is a pointer. It cannot be used yet because it does not point to any memory location. Pointers can only be used if they are pointing to a meaningful memory slot.
- To dynamically allocate memory, there has to be a pointer that can be used.

18 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadaş © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

new Operator

```
dnnumber = new int;
```

- This expression allocates memory space from the system for data of type `int` and assigns the address of this space to the `dnnumber` pointer.
- The `new` operator both allocates memory space that is large enough to accommodate a variable of the right type and returns a pointer of a suitable type.
- Compared to the usage of `malloc`, this obviously provides a great convenience. The need for the `sizeof` call and the conversion of the returned "generic" pointer type is eliminated.
- If memory cannot be allocated for whatever reason, `new` returns `NULL`.
- Now, the `dnnumber` pointer is pointing to a real location. This dynamic memory space can be accessed using this pointer.

```
*dnnumber = 5;
```

19 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadayı © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

delete Operator

- Dynamically allocated memory space should be returned to the system when it is no longer needed.
- Memory is returned using the expression

```
delete dnnumber;
```

Note!

- If new memory has not been allocated for a pointer that had the memory it was previously pointing to returned to the system, using `delete` again on this pointer may lead to unexpected errors.
- `delete` can **only** be applied to memory space that has been **allocated with new**.
- `delete` cannot be applied to pointers pointing to nondynamic variables. These memory slots cannot be returned.

20 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadayı © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Dynamic Creation of Arrays

- Dynamic memory may be allocated for the whole array at once.

```
float *darray;
darray = new float[100];
```

- Here, an array large enough to hold 100 real variables has been dynamically created. The start address of the array has been assigned to the `darray` pointer.
- After the above assignment, `darray` can be used to access array elements.
- To return the array to the system:

```
delete [] darray;
```

- Note:** When deallocating the array, we should not forget to use `[]`.

21 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadayı © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Discussion

- What happens if dynamically allocated memory slots are not returned?
- Try it out: What is the largest memory space that can be allocated at a time?

22 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadayı © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Abstract Data Types

23

Abstract Data Types - ADTs

- Integers, real numbers, and Booleans are built-in data types.
- Similarly, we could define more complex abstract data types such as the list, set, stack, and queue.
- Certain operations that can be performed are defined on every built-in data type.
 - Integer addition
 - Real number multiplication
 - Boolean "or" operation
- Likewise, abstract data types are incomplete without the operations that can be performed on them.

Example: On the set structure,

- We could define operations such as union, intersection, complement, and set size.

24 ITÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadayı © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Basic idea:

- The implementation of these operations is written once in the program.
- Any other part of the program that needs to perform an operation on the ADT can do so by calling the appropriate function.
- If for some reason, implementation details need to be changed, it should be easy to do so by merely changing the functions that perform the ADT operations, and should not affect the programs using this data type. This property is known as **transparency**.
- Which operations (functions) will be defined for an ADT is a design choice, and it is determined by the programmer depending on the need.
- How an ADT will be implemented (that is, how it will be designed and programmed) may vary from programmer to programmer. These design details and differences do not change the outside appearance of the ADT (as viewed by the programs that use it).

25 ITÜ, BİLGİSAYAR MÜHÜRÜ, G. ERYİĞİT, S. KABALAYI © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

C++ and struct

- In C++, the structure provides a natural capsule for defining an abstract data type.
- Thus, the data type and the functions that define the operations to be performed on this type are located in the same capsule and logically related.

C	C++
<pre>typedef struct DataType{ int array[10]; int elementnumber; } NewArrayType; int CountElmt(NewArrayType *d) { return d->elementnumber; }</pre>	<pre>struct NewArrayType{ int array[10]; int elementnumber; int CountElmt(); }; int NewArrayType::CountElmt() { return elementnumber; }</pre>

26 ITÜ, BİLGİSAYAR MÜHÜRÜ, G. ERYİĞİT, S. KABALAYI © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

struct

- Structures cannot contain another structure of the same type. They can only contain a reference (pointer) of the same type.

```
struct NewArrayType{
    int array[10];
    int elementnumber;
    NewArrayType *y;
    int CountElmt();
};
```

- Defining a structure does not reserve any space in memory. It only creates a new data type. However, when a structure variable (variable of the type the structure defines) is declared, memory is allocated for this variable.

27 ITÜ, BİLGİSAYAR MÜHÜRÜ, G. ERYİĞİT, S. KABALAYI © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

struct

The following operations can be performed on structure variables:

- Assignment operation =
- Address operator &
- Member access operators . and ->
- sizeof operator

28 ITÜ, BİLGİSAYAR MÜHÜRÜ, G. ERYİĞİT, S. KABALAYI © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

struct

- The members of a structure do not have to be located consecutively in memory.
- Passing structures to functions:
The whole structure or its members may be passed to the function. Structures are passed to functions by value.
- To pass a structure to a function by reference, we have to pass its reference.
- Note:** An array that is a member of the structure also gets passed to the function by value.
- When the assignment operator (=) is used between structures, all members are copied.

29 ITÜ, BİLGİSAYAR MÜHÜRÜ, G. ERYİĞİT, S. KABALAYI © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

struct (advanced topics)

- By defining nested structures, more general-purpose (easier to update based on the data type it holds) programs can be written.
- This situation is a choice completely dependent on the design.
- Example:

```
struct node{
    char firstname[20];
    char lastname[20];
    node *next;
};

void add(char *firstname, char *lastname){
    node *newnode = new node;
    strcpy(newnode->firstname, firstname);
    strcpy(newnode->lastname, lastname);
    ...;
    add("ahmet", "cetin");
}
```

30 ITÜ, BİLGİSAYAR MÜHÜRÜ, G. ERYİĞİT, S. KABALAYI © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Example

```
struct nodedata{
    char firstname[20];
    char lastname[20];
};
struct node{
    nodedata data;
    node *next;
};
void add(nodedata d){
    node *newnode=new node;
    newnode->data=d;
    . . .};
nodedata d={"ahmet","cetin"};
add(d);
```

In structures, since the = operation copies the whole content, the data part will be completely copied independent of content. Thus, it will not be necessary to update the code by performing different copying operations for different types. The node "newnode" is stored in a different location in memory.

31 İTÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadaş © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

Example

- Note: What if the node data is as follows?

```
struct nodedata{
    char firstname[20];
    char lastname[20];
};
```

```
struct nodedata{
    char *firstname;
    char *lastname;
};
```

newnode->data=d; ??? which operation will be performed?

The result of the operation in the first structure and the second structure will be different. In the first structure, array contents are copied to a new location, while in the second structure, only pointer values are copied. The values of **firstname** and **lastname** pointers will be copied. In other words, the address values they hold will be copied. In this case, the pointers in the **data** part of the "newnode" structure and pointers of the **d** structure will point to the same location in memory.

32 İTÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadaş © 2011 Function Calls in C, Dynamic Memory, Abstract Data Types

```
struct nodedata{
    char firstname[20];
    char lastname[20];
};
```

```
struct nodedata{
    char *firstname;
    char *lastname;
};
```

Name	Value
v	{...}
ad	0x0012f6e4 "ahmet"
soyad	0x0012fef8 "cetin"
yeni	0x00441bc0
veri	{...}
ad	0x00441ba0 "ahmet"
soyad	0x00441bb4 "cetin"
sonraki	0xcdcdcdcd

Name	Value
ad	0x0042e024 "ahmet"
soyad	0x0042e01c "cetin"
yeni	0x00441bc0
veri	{...}
ad	0x0042e024 "ahmet"
soyad	0x0042e01c "cetin"
sonraki	0xcdcdcdcd

33 İTÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadaş © 2011

To do: In recitation

- Solution to Homework 2
- Function calls
- Parameter passing
- Where in a program the variable may be used (scope)

34 İTÜ, BLG221E Data Structures, G. Eryiğit, S. Kabadaş © 2011

Homework

In the PhoneBook example, phonerecords were defined as constant-size arrays. Realize the necessary operations (in code) for defining dynamic sizes for these arrays.

Homework

The phonebook example was solved using an array. In this homework, you should use a dynamic array instead of a static array. In this structure, this is what you need to do: First, inside the phonebook structure, you should define a variable `size=5`. Within `Book.create`, an array of this size will be dynamically allocated. Then, every addition operation will check to see if the current index has reached `size`. If the array is full (that is, the index has reached `size`), then a function called `increase size` will be called. This function will allocate a dynamic array of `size*2` from the memory. We will call this new array. This function will first copy the elements in the array to the beginning of the new array using a loop. Then, the old array will be returned to the system. The array will be assigned to the new array. And the new size will be defined as two times as big.