Section 2

ENGLISH ORIGINAL TEXT УКРАЇНСЬКИЙ ПЕРЕКЛАД

|  |  |
| --- | --- |
| [MUSIC PLAYING] ALLISON BUCHHOLTZ-AU: Hey everyone. Welcome to your first official CS50 section. As you can, see this is section CS50-like, just to pull up our agenda for today. So, who am I, as I'm sure you're all wondering. I am your TF. I'm not just a random student who's impersonating your TF.  And I'm going to go through kind of have sections will flow, expectations we have, resources, so on and so forth. We're going to talk about arrays, ASCII functions, command-line arguments, and it's not on here, but I will also be helping you kind of think through your next pset for this week. Which I am sure you will all appreciate.  So, first question-- who am I, besides your exuberant TF here. I'm Alison Buchholtz-Au. This is my second year TFing CS50. I also TF CS51 in the Spring. You might see again there if you decide to take it. I'm also a PAF, so any Freshmen-- and if you're not a freshman, this is my third year as an advising fellow. I'm very well-versed in advising you on life and courses within CS and not within CS. I am a Computer Science Concentrator. I'm a senior in Adams House, best house. And before I switched to CS my Sophomore Spring, I was actually a biomedical engineer. I was going to go to Med school. I was going to be a trauma surgeon. And that has completely changed since I took CS50. I took the course my Sophomore Fall. It was my first introduction to CS ever. I was one of the 78% of you who had zero experience coming in, and it completely changed my life. And now I'm working at Microsoft, and your lovely TF. And CS50 is probably one of the best experiences that I've had here at Harvard-- both taking the class and being able to help teach students like you.  So I'm really excited that you're all here. In case you came in late, there is candy, which you should feel free to come grab, or send someone else to grab it for you. it's OK. I don't want to eat that. My room has enough chocolate, so y'all should try and finish that. I know there are 100 pieces, but like, 4 o'clock on a Monday, I think everyone could use some sugar.  So all of you who are officially in my section should have gotten an email from me with my phone number, email address-- feel free to add me on Gchat, feel free to add me on Facebook, and also for the rest of you, you can email me right here. There are two H's. Everyone always does like two L's or two C's. Two H's in the last name. Otherwise it's going to bounce, and I'm not going to get your email.  So feel free to email me, to contact me at any time. I may not get back to you within 24 minutes, but I promise to get back to you within 24 hours. If you call me half an hour before your pset is due, you being like, I have no idea what I'm doing Allison. Help me. I'm going to calm you down, but at the end of the day, if you're calling me half an hour before your pset is due with nothing written, I'm going to be like, well, maybe it's time to use that late day. So I will respond to all of your requests in a very timely manner. My phone is usually attached to my hands. I typically respond much quicker than 24 hours, but I can only guarantee a 24 hour response.  All right. So why are we here? Also, if you have questions at any time, please let me know. I talk a lot. I talk fast, but please feel free to interrupt me. It gives me breathing room as well. So sections are a time for us to really just dive in, get some hands-on experience, to go through topics that we mentioned in class or in study materials that we recommend to you guys online. And we'll actually go through some of those resources in a bit.  So some notes on section support. CS50-- one of things that makes it one of my favorite classes is the feeling that you're never alone. We have a staff of over 100 people who are here to help you. We have office hours Monday through Thursday. So there are so many people who love the class just as much as I do and who are really here because they want to be here. Most of us are students, and this is like a fifth class in addition to the rest of our work.  And it's a lot of work, but we do it because we love it, and we really love to teach you and help share our excitement for this subject and this class. So please take advantage. Come talk to us. I get lonely when my students don't talk to me, so if you want, come hang out with me. It'll be great.  So section is obviously one of your biggest things. We'll go through things that you learned in lecture, do some short examples when we have time, and generally kind of get an idea about things you should be thinking about for your problem set. Shorts-- how many people actually watched the video from your scratch short? Anyone recognize me? So those are very great. You should definitely watch those. A lot of work has been put into them. And they're just meant to be bite-size pieces for you to just watch for three or four minutes and get a better understanding of a concept. Walk throughs-- how many people have watched the walk-throughs for previous ones? Zamyla is amazing, right? Like, I wish I were Zamyla sometimes. So definitely use your walk-throughs. She will break it down into small, bite-size pieces. And when you have these huge specifications from your problem sets, it's going to be really important to be able to just find somewhere to start and work slowly through it.  All right, we also have Study50, which is study50.harvard.edu, I believe. You can just Google study, and it'll come up. This is one of the best resources we have. It is PowerPoints with notes and practice problems for you with solutions that you can actually walk through. So if you ever want more practice, more than we do in sections or more than your problem sets, this is really a place I encourage you to go. It was built last summer by some of my really good friends. And it's amazing. In fact, a lot of the slides that I'll be using for Section will come from Study50.  So a lot of the TFs use it. And finally, as I mentioned, office hours. If you're having trouble with homework, you're having trouble with a concept, come to office hours. Go early in the week if you can. Get out to the quad, because it is kind of far. No one likes to walk out there. But it is to your advantage, because then you're going to have all these TFs, [INAUDIBLE] surround you. And especially now, just a tip, Thursdays are very chill right now in Mather because your psets are due on Thursdays. And knowing wants to use your late days yet.  So if you're having trouble with concepts, there are lots of TFs who are there to help you. So come out to Mather on Thursday. If you want to see me, I'm going to be there. I'm typically doing my own homework, because no one wants my help. So come see us.  Meet us halfway. So how many people have attended lecture or watched it online? How many people went to super section last week? Cool. That's actually a fair number. How many of you have read your spec for this week, for your pset? Ooh, I'm proud of you guys! More candy for y'all.  Good, so what we mean "meet us halfway," is that section is really only going to be super useful to you if you come in having read your specification for your pset. Because when I go through an overview of things you should be looking out for, it's not going to make as much sense if you don't know what your problem set is going to be asking you to do.  If you don't come to section, obviously I can't be that useful to you. I'm not going to take it personally if you don't come to my section right now, but definitely you should. If you can't, watch them online. They're there for a reason. Mine will be right there. As you notice, we're being recorded, so it'll be right there for you guys. As well, going to lectures-- that's obviously where you're getting the start of your material here. So I will definitely try and help you as much as I can, but I can only meet you so far. You have to kind of meet us halfway there.  Grading-- so, all of you who got an email from me, you are my official section. I will be grading your psets. And I just want to say, one thing that you should really pay attention to are the comments. The comments are often more useful than the actual score we give you. And the comments are actually where I spend a lot of my time when I'm grading. So I would appreciate it if you read them. And they're actually how you're going to learn more about design and style and things that are a little less cut and dry.  So really pay attention to those comments. If you have questions about them, or questions about your score, please come talk to me either before section, I'll probably be hanging out in the lobby, or afterwards. If you want to schedule one-on-one meetings about how you can help improve later problem sets, just let me know.  And then just a couple of tips for you guys. So one of the biggest things I always stress to my section when you're learning how to code is to write things out on paper first. If you have a game plan for where your code needs to go and what it needs to do and it's broken down into little bits of pseudocode code you've written out, you're going to be less likely to make syntax errors or create an if loop that doesn't have an else. If you know where you're going overall, you're less likely to make these tiny mistakes that will sometimes take you hours to fix, because you're like, where am I missing this bracket?  On that note, please use Style50. Especially when you're going to office hours, if your code is all switched over to one side, it is course policy that we can say, fix it so that it looks like Style50 says it should, and then we'll help you. So it'll make your life easier. It'll make our lives easier. Everyone's happier. Everyone gets better grades. Isn't that what we all want?  So write things out on paper before you ever touch your computer. Talk things out at a high level, and make sure you know where you're going. And if you're unsure, sit down with someone and walk them through step by step what your code is supposed to do. And nine times out of 10, you'll be like, oh, I forgot an if condition or I forgot a semicolon here or I'm updating this variable wrong. So those are my tips for success.  So since about half of you look like you attended super section, I'm just going to very briefly go through loops, which weren't on our original agenda. But they are really important. And so I'm going to kind of speed through those before we get into our actual section. Before I do that, are there any questions-- logistically, personally, is there anything else you want to know about me or about section or class in general? All good? OK, cool. Lovely.  So loops-- you guys should all recognize these pieces from scratch. So loops are basically just a way for us to do something some number of times, some repeated action based on some conditional. So we have three different types. We have for loop, while, and do-while.  So for loops-- we just have a very general layout here of a for loop. And this is great for when you know how many times something needs to execute. When we talk about the other loops, you'll see why that's an important distinction. But for loops are for something set. You know you can either calculate the number or you know the number of times you want this repeated at the beginning.  So if you see here, we have just a general kind of skeleton framework for a for loop here. So for-initialization, this is where your variables are initialized. With Mario, I'm sure you guys did something like int i equals 0. That's where that would happen in blue. You have your condition, which is what's checked every time. If this condition is true, then the rest of the code executes. Then it'll run again-- and ask. And then we have update, where you're updating your variable. So, again, with like Mario, I'm sure you guys did something like i plus plus. So every time the loop ran, i got updated so that when we were checking it against some condition, it was changing.  Because if you just have a static variable, if it executes the first time, it's just going to execute infinitely. So you have to make sure you update your variable properly. And we also have just a visual representation there. Everyone good? For loops. Should have seen these in your pset. Cool.  So here's just an easy example. Print This is CS50! 10 times. And so we have our initialization, as we see there, with int i equals 0, for i is less than 10, and i plus plus. And it'll print that 10 times.  So while loops-- while loops are great when you don't necessarily have to know how many times it's going to update in the beginning. You just have some condition that's checked. And this could be something like while-- let's take an example from your pset. If Mario, you try to input a negative number. Right? You were supposed to re-prompt your user. So you can say, well, if the user inputs something less than zero, re-prompt them. And I'm sure that might have been something that some of you used in your code.  So it's a simple thing. You have while, some conditional that is checked every time the code goes to execute. If it evaluates to true, we run it. Otherwise we don't. And what's really important-- something I think that David talked about in lecture-- are the braces. Whatever's within the braces is what's executed. If you forget those braces, it's only going to be the line directly after the while was executed.  So if you have three things that are supposed to happen when this condition evaluates to true, and you don't have those braces, only the first thing is going to happen. So be very cognizant of where you put your braces. If you stick with Style50, this will definitely help you. Cool.  So this is a countdown from 10 to zero. And as you see here, we initialize some counter outside of it. One thing that's different is we're not initializing our variable within our while loop. It's initialized outside of it. We are simply just putting the condition in for our while. So in this case, it's while count is greater than zero. And we print out what our count is, and then we decrement our variable. And that's also another thing to notice. Our update doesn't happen within that first part of the while loop. It will actually happen within the braces, the body of your text.    So do-while loops-- do-while loops are great for user validation. So some of you might have also used this in your pset. You can say, do, like, ask the user for input. And then while, like, the input is less than some number.  So for an explicit case with Mario, it would be do printf, enter an integer, and then some integer equals getint. And then it'll actually execute that code first. You'll actually have some sort of integer. And then you could say, while that integer is less than zero. So what it's going to do is it's going to execute at least once. It's going to check the condition. If the condition is true, it'll run again.  So do-while loops are great for user validation, because you know the code is going to execute at least once, whereas with while loops, you're not guaranteed that it's going to execute once. It's going to check the condition first and then decide to execute it, while a do-while will execute the code first and then check to see whether you need to repeat it. Does that distinction make sense to everyone? OK. Cool.  So in this case, this is kind of what I was talking about, this re-prompts until you get a positive number. So we know that printf "enter a positive number" and actually asking for that input will happen at least once. If the user is evil and keeps entering a negative number, who knows how many times it'll execute. But this code is guaranteed to execute at least once. And that's why it's great for validating input. And you will use that quite a bit.  All right, any questions so far? We're all good? Am I talking too fast? We're good? OK. Awesome  So we're going to go ahead and talk about arrays. Cool. So arrays are basically just data structures that allow us to store things of the same type. So if you ever have an array, it's either just going to have ints or it's just going to have floats or it's just going to have chars. You're not going to have an int with a char with a float with a double. One thing. Arrays are just one size, or they're just one type.  So here we have an array of size three with three integers in it. They could floats, but we're going to say they're ints. So one thing to realize is that arrays are a set size when you initialize them, and they are not easily-- since you all are less comfortable, you should just think of them as not being able to extend in size. However big you set out your array in the beginning, that's the size it's going to stay, because arrays are continuous blocks of memory. And when you guys get into a little bit more of how memory's actually laid out on disks and in the heap and the stack, it'll make a little more sense.  But you could just think of like, it's just a row of spaces on your disk. And you can't guarantee that there's going to be free space after it. You could initialize an array of three and then maybe you initialize another array of five later, and it's right after that. So if you were to go past spot three in that first array, you would be writing over something else. So arrays are-- for you guys, just think of them as a fixed size.  So creating an array-- you're going to need to do this quite a bit. So in the same way that we have a general structure for our for loops, we have a nice general structure for our array. Because they are of one type, all the elements in an array are of one type, you need to initialize what that type is.  So, as you see here, we have a nice little bracket data type. So if we're creating an int array, that will be int. If we're creating a char array, it'll be char. If we're creating a string array, it'll be string. And then the name of your array, whatever you'd like it to be. So maybe it's test scores or maybe it's students or maybe it's candy. Whatever you decide to name your array, that's what it'll be.  And then in brackets, you'll have the size that you want. So, are we storing 10 students or are we storing 15 types of candy? What not. So in our example here, we're creating an array of size three, which you guys see right here on the right. And when we first initialize it, everything is set to zero. So it's just thought of like a blank slate. We have all these spaces, we have all these boxes we could put our data into, but they're just blank for the time being.  So if we want to actually assign them these values, we do so as right under here shows. So you have whatever the name of your array is and then what index you want. So the index just refers to, like, what slot we're looking at. And an important thing to notice is that arrays are zero-indexed. So if we want the first space in memory of our array, it's going to be zero. If we want the second, it'll be one. If we want the third, it'll be two. So on and so forth.  Which is also why, conventionally, when we do for loops-- I'm sure you guys were wondering, why do we start at 0 versus 1? And that's because when we transition into using arrays, it maps correctly. So if you want to iterate into an array, it makes a lot more sense to do i equals 0, because we know that will correspond to the first spot in memory. Everyone good with that? Cool.  And then on the bottom here is just another way to initialize an array. You still have your data type and the name, but instead of actually putting a size in there, you can just do empty brackets. And then with these curly braces at the bottom, you can just input the data that you want separated by commas. And that will automatically say, OK, I see that you have three things in these braces. So I know that I need to allocate three blocks of memory and then store those.  So the first version you might use if you're asking your user to input values so that you can iterate through the array and ask-- get some int to input them. If you know the values beforehand, it makes a lot more sense to use the second way. But in most cases, you might not know what those values are going to be. Cool. Any other questions? Alright.  So accessing elements-- so one of the great things about arrays is that they are random access, meaning that you don't have to look through every block. If you know that you want what's in block two, you can just say, give me block two. And that's why these indices are so important, and that's how we actually access them. So in this case, as we saw before when we were assigning values, in the one before, we had the name and the index we wanted to access, right? So in the same way, that's all we do to actually pull that data out. We have the name and we have the index that we want.  So in this case, the for loop down here at the bottom, anyone know what it's doing, what it would print out? Mmhmm? Exactly. So yeah, it's just iterating through. i is equal to zero-- we can walk through the code just quickly. i is equal to zero, i is less than three at this point, right? So that checks out. And we say, OK, print f whatever is in temperature i. i is zero right here when we first iterate, so we go to this first spot, and we say, OK, 65 is the number we want to print out. So it'll print out 65 and then do a new line. i will update, so it prints 87. It updates again, and it'll print 30. Everyone cool? Awesome.  All right. So here's kind of one thing I was saying how you can keep track of someone's score and why you would use the first way of initializing it instead of that second way. And this just goes through. And notice we have a class size of 30. And we're initializing this array of ints that is of size 30. And then we are iterating through and we're asking the user to input scores for each of these and then assigning it to a specific place in memory somewhere in that array. Cool? Does that make sense to everyone? Mmhmm?  So hashtag define class size 30 is a preprocessor directive, which just means it gets-- it has to do like the compiling process. You can think of it as a global variable. The way we do it is typically-- it allows your code to be more easily changed. So let's say that our class size suddenly goes from 30 to 15, if I hadn't defined it this way, I would have to go through my entire program and change every instance of 30 to 15. But with this, I get to change one spot, and everything else changes.  If you ever want to do a hash define in a case where you're keeping track of some set number of scores for a class or you're using a number that will be used, like, throughout a very long program, it's better to define that at the beginning so that if ever it changes, you get to change one spot instead of 100. Yes?  STUDENT: Between doing that and just declaring [INAUDIBLE] over at the top.  ALLISON BUCHHOLTZ-AU: So it has to do with efficient-- it's kind of outside the scope of what we can cover in this section. It has to do more with efficiency and how things actually work in the compiling process. If you want to really know about it, I'm happy to send you an email with resources about it. Hash define tends to be preferred for things. And as you code more, you kind of learn the nuances of when you should use a global versus the hash define. But for the time being, you don't really have to worry about it is that the short answer. Everyone good with that? And also, if you want to use a hash define, it's really important to notice that the name should be in all caps. We're not just doing CLASS SIZE to be dramatic. It should actually be in all caps. Cool. Anything else there? We're good? Lovely. Welcome.  OK, so I want you guys to take a look at this and see if you can find the bug. I'll give you a hint. It's somewhere in that for-loop. Mmhmm?  STUDENT: Should be less than equal to 2.  ALLISON: So it could be less than or equal to 2, or it could be less than 3. And what's the reasoning for that?  STUDENT: The [INAUDIBLE], 0, 1, 2.  ALLISON: Exactly. So in an array of size n, we only have indices of n minus 1. Cool. And then we can get really crazy and get multi-dimensional arrays. One of the problems when I took it in my year required multi-dimensional arrays, and I think one of them might require it this year, so be comfortable. Wrap your head around it now. It will come back to haunt you, but in a cool way. So you can really just think of multi-dimensional arrays as arrays of arrays. So you can kind of think of this top row as the first chunk of memory. And this one is the second chunk of memory, and the last row is the third chunk of memory. And within that, there's an array. But of course, it's easier to depict like this.  So you initialize it the same way. This is a character board of three by three. So you have three rows and three columns. We're representing it this way. And you would access it the same way, column by row. And so 1,1 as we see here. We assign a zero, zero up there. 2,0 and 0,2. So you would just access them-- if anyone's ever done linear algebra, the same way you access an element in a matrice, it's the same idea here. So you can relate it back to math. You don't have to worry too much about this right now. It's good to have exposure, to know that you can do it. You can create some crazy number-- you can create crazy arrays is all I'm going to say. [INAUDIBLE] It gets a little crazy, but it's really cool. Awesome.  And then, so we have an example here. It calculates a string length. So how many people knew that the strings that you're using are just arrays of characters? OK, yeah. So you guys may think that you haven't used arrays before, but any time you use getstring in the CS50 library, you're actually just asking for an array of characters. And we're taking care of all that in the back-end for you. But you have been using arrays since you started. You just didn't know it yet.  And whenever you have a character array or an array that's storing a string, the last thing is always what's called a null terminator, which is this right here. And that is at the end of every word that you're storing. So if we want to figure out the length of a string, we can say, well, you know, the contents of that block is not equal to our null terminator. That means that there is some character there that we actually care about that's part of the word. You increase your length. And then when we actually get to the end of the word, it'll terminate and it'll return our length for us. Mmhmm?  STUDENT: Does the space count as the null terminator?  ALLISON: So a space is not a null terminator. So if you have multiple-- a space is actually a specific ASCII value.  STUDENT: What's the exclamation equal again?  ALLISON: So, this is what you refer to. If you ever hear me in office hours, I always call it, like, bang equals. So bang is not. So this is not equals. So if you're trying to see if something's false, you know always do, bang whatever the variable is, and if it's false, it evaluates to true and you can do cool stuff with that. More on that later. Cool. Everything good there? Awesome.  So now it's your guys' time to work, since I've been talking. So I want you to just create an array with the integers one, two, and three, and then have them printed out. You don't have to do, like, main, blah, blah, blah, whatever. I just want you to initialize the array and then create a for loop to print them out-- or a while loop, up to you.  I'll just give you a couple minutes to work on that. I'm going to rest my voice. If you have any questions, I'm happy to come around and talk to you guys. Feel free to talk with each other. Get more candy. In fact, I'll just walk around with candy. How's that?  Do you want any? Anyone else in this room want candy? You can also take more than one, guys. Take a handful if you want. May as well. Everyone else good? OK.  Also, I'm going to create an anonymous Google Form, and you guys can just submit feedback after every section if there's something you want to improve upon or something you want done. If I'm a little too peppy for you, I can tone it down. I'll create that and send that out to you all afterwards.  All right. So let's start small. How would we initialize our array? What's the type of our array? An int, right? OK, so what do you want to call your array? Int array, cool. All right, so we have int int array equals, and what do we have after that?  STUDENT: [INAUDIBLE] brackets.  ALLISON: Braces. And then inside the braces? One comma two comma three. Cool. So that's all right. So now we have our for loop. So in the first part of our for loop, what do we have?  STUDENT: i equals 0?  ALLISON: So int i equals 0, and then what is our condition? What's i going to be less than? Less than three, and how we do we update i? i plus plus, updating it by one. And then we're going to have some printf of the integer, and what is that last part that's actually going to say what we should be printing? It would be the name of the array, which is int array, right? And what's in the brackets of int array? i. [? So I ?] called my example, but there you go. Not that bad. Everyone good? Cool.  So we're done with the arrays. Congrats. You managed to iterate through all the-- yes?  STUDENT: [INAUDIBLE]  ALLISON: Yes.  STUDENT: I have a question. Are you supposed to indent the braces?  ALLISON: So the braces should line up with the for loop, and then everything inside the braces should be indented.  STUDENT: OK, should the for loop be indented?  ALLISON: The for loop does not need to be indented at this point. If you were in main, if we actually had a main function here, it would be indented from main. But in this case, it's fine. Yes, question.  STUDENT: Do you need to have the brackets after example?  ALLISON: Yes, if you're initializing it that way. So remember, this is the second way of initializing an array where we have the braces and then our actual data separated by commas within.  STUDENT: I thought there were brackets for that example.  ALLISON: No, they're braces. They're braces. If you're initializing it that second way, it's braces. If we were to say, int example-- if we just wanted a blank array for ints, it would be int example brackets three. The brackets represent the size. When you have braces, it's the actual data you're putting into it in this way. We can scroll back really fast.  So in this one, this is just our initial array, initialization. And here, we are individually assigning spots to them, so this represents the index of our array, which is why we have brackets. But here, if you notice, we've left our brackets without a size, and we initialize it with the actual data all-in-one with braces.  STUDENT: So why don't we have brackets in this example? ALLISON: So, in which part?  STUDENT: Wouldn't we say, int example brackets equals braces [INAUDIBLE] brackets for example.  ALLISON: Oh, sorry. You're right. We do have brackets there. Sorry guys, my bad. Yes, you should have brackets after example. You're absolutely right.  STUDENT: [INAUDIBLE] not doing it.  ALLISON: No, you have to have brackets, because otherwise it's not going to declare an array.  STUDENT: [INAUDIBLE]. Sorry about that.  ALLISON: Sorry, you need brackets after example. Typo. Good catch, gold star for you. Also, if you are asking a question, if you guys would just tell me your names, I'd love that. I'd love to be able to know all your names. I'm not going to cold call you, I actually do just want to know your names. So please actually tell me your names.  LEAH: Leah.  ALLISON: Leah. OK, so functions-- I know in brief they talked about this during lecture. So functions are kind of just like these little bite-size things where you pass in inputs, something magical happens, and you get outputs. Cool. So you actually used a lot of these already. Get int, get string, print f. These are all functions where you just call them, there's lots of magical things going on in the background that you don't necessarily see, and you get out what you want. Or at least you get what you hope you want.  And basically the point of functions, and one of the main themes of CS, is to break your code into manageable pieces. When you start writing these really long programs, or in Scratch when you had this grand idea for a game, you need to be able to break it down to, like, OK, how do I start? What are the little pieces that I need? Oh, I need to ask the user for something. Now I need to print something. Oh, I need to calculate this value.  And learning how to break up your code and the large problems you have into these small pieces and creating functions is actually one of the big cornerstones of CS. So you can think of a function just as like a black box, a magical black box, that you put things into and you get some output. And the rest of the program doesn't need to necessarily know what's going on within that black box. All it cares about is what goes in and what comes out. Cool.  So why functions? Organization-- as I said, when you're dealing with very large code bases, how you organize your code will be much easier if you use functions. Because you'll be able to be, like, OK, this is what this function does and here's what another one does. And you can easily see how they all fit together. So breaking it up into all these manageable subparts.  So simplification-- I'm sure you guys all saw this, as I said, with Scratch. You have this grand idea, and you're like, how does all this work? But if you approach it piecemeal, you say, OK, how do I make one sprite float across the screen? That's a little bit easier. So good use of function makes your code much easier to read. It makes it easier to debug which as you get into your later problem sets, you're going to really want to be able to do. And they're also easier to design and implement. You can code up a small function relatively quickly and make sure it works versus trying to create this whole long program and then kind of go through and see what's working and what's not.  And then reusability. So functions only need to be written once, and then you can use them as many times as possible. So it's, like, eco-friendly in a sense. If you had things like print f, where you had to write out the magic that goes on behind print f every single time you wanted to print something, you would be pretty sick and tired of it by the end.  One of the things that you'll learn in later CS classes, or one of the best pieces of advice I get is, if you are copying and pasting code, it should probably be a function. If you have the exact same lines all throughout your code, if you factored them out, your code would probably be, like, five times shorter and be much more easy to read. And instead of trying to troubleshoot all these different places where things might go wrong, you have one function that you get to troubleshoot. And I promise, a lot of this might seem kind of abstract now, but as you get into later and later problem sets, it'll make a lot more sense and really be driven home. Are there any questions about functions so far? Why we're using them? I know we haven't gotten into the nitty gritty yet. So defining a function-- just like arrays, we need some sort of-- this is just the general output. So this is a function that's just going to cube some input. And on the next page, actually, we have all these awesome little things here. So, can everyone read that, out of curiosity? I know the purple and black might be a little hard.  But big things to know-- so the first one right here is our return type. So this is talking about the output of this function when, in this case, we put in some number, what we're getting is that number cubed. So it should be an int in this case. Maybe it would be a double or something else later, but in this case, it's an int.  With c, you always need a return type. It'll be an int. It'll be a float. It'll be a double. But you have to specify what this function is going to return. Otherwise it will yell at you, and it won't compile. You'll be sad, and I'll be sad. And it's just not good. OK.  And then we have our function name. And as you can see here, with c there's this very consistent paradigm. What's your type, what's the name, and then some other thing at the end. So we have our return type, our function name, and then we have our header with our parameter list.  So the parameter list is, what is this function going to take in? A parameter list is simply a synonym for, what are our inputs? And in the same way that we have to define our function and give it a return type, each of our inputs needs to have a type associated with it. So we know what our function can actually work with.  So in this case, we have some int input. So again, it'll be the type and what you're calling it. And then, as you see here, we have our body. So we have some int output, that is just our input times itself times itself, which just cubes it. And then we return that output.  So as you see here, we have an int times an int times an int, so it returns an int, which has been declared there. So everything is cohesive. Everything's happy. Your function will run. And this is just the general thing. So always have return type, name, and your parameter list. Each thing in your parameter list, or input, needs to have a type associated with it. And then you have your body here with whatever you want to do with your input.  And then obviously you want to return something. Sometimes functions will just return. They don't actually return something for you to use. But you have to return in some way. And when you're making your own functions, we can get into that a little deeper. Personally, if you want, there are a lot of different things you could do there. Everyone good? Anything on this list that you want me to go over, that you didn't understand? Everyone's good there? Cool. Awesome.  OK, so we're putting all this together now. So we have some int cube input, so this is a complete program here. Up until now, I've kind of been giving you guys snippets that might be going within a program. We've just been looking at functions. But here's an entire program. So how many of you remember the word prototype from lecture? Cool. We've got one. What's your name?  STEPH: Steph.  ALLISON: Steph? OK, awesome. So, do you remember what a prototype is?  STUDENT: You say [INAUDIBLE] before you actually deal with it.  ALLISON: Do you remember why?  STUDENT: No.  ALLISON: OK. Gold star. So yes, a prototype we have beforehand, because otherwise, our compiler is going to yell at us. It's going to say, OK, what is this cube function? Like, you literally have told me nothing about this. It's like when you walk into a classroom, and someone's like, there's a quiz today. And you're like, you never told me about this. I'm not happy with us. The prototype is basically like your syllabus saying, look. Heads up. There's going to be a quiz on this day. Don't freak out when you get to it. You're going to be fine.  So all the prototype does is tell main, I'm going to use this function. I promise I'm defining it later. Don't freak out at me. Just compile and do what I tell you to. So we have the prototype there just to make our compiler happy. And it's basically a promise that you have defined this function later and that you aren't just calling this random thing that it doesn't know what you're going to be doing.  So in this case, we have main here. We initialize some integer x. That's two. We're going to print out what x is. We're going to cube x. As you see, we have our function declaration down here that we talked about previously. It'll cube x, and then, if we remember, the cube function actually returns an integer to us, which is stored in x again so that we can print out eight, or cube x right now. Does that make sense to everyone? We're good? All right. Awesome.  All right. How many of you guys remember This so this is basically just your stack and your heap, just a visualization of how memory is stored here. So we just want to make sure that you understand how these are represented in memory. If you take classes like CS61 and stuff later, you get to learn this far more in depth, and it's really cool. I highly recommend it. But for now, I'll give you the broad overview so you don't have to know the nitty gritty.  So the top just a text segment which contains the actual zeros and ones, the binary for that. And this is used for storing global variables if you have any. As you move down, we have, as you see here, initialized data, uninitialized data, and then heap.  So we don't really talk about the heap right now. We'll get to it later. For now, I'm just going to wave my hands and be like, you don't need to know about this now. But we will talk a little bit about the stack. So the stack is where-- we have zoom in. This is actually how the program we just looked at occurs in memory.  So what happens is, every time we call a function, we get what's called a stack frame, which is one of these. So main's parameters. So those are the things that we pass into main. So they're right here at the bottom, because that's the first thing we call. And then we get to main's locals, and when we say that, we mean the local variables that are stored within main. So locals here would be, like, x is equal to two in this case. Because that's localized to main. Does everyone remember scope, going over that in lecture? OK. So, just the variables that are initialized within main. So that's why we have main [? vocals. ?]  And then within main, we call cube. Right? So we get another frame with cube's parameters. So in this case, cube's parameters are now the x that we passed in, the two that we passed in. And then cube's locals, which is where the actual cubing happens. And then it returns.  So what happens is as cube actually does what it's supposed to do, it returns. When it returns, this frame leaves, and its returned down to main. And then within main, we can actually print it. So when you're returning something, when your function returns, it's like passing on those values to the frame below it and then leaving. And things have to execute in order.  And when you get to bigger programs, we can make cooler and more complicated diagrams. But for now, this is just a general overview so you have kind of an understanding of what happens when you're calling a function and how that actually looks in memory. Cool? Everyone good? Awesome.  So this is one that is just trying to swap things. As we see here, we have our function prototype so that our compiler doesn't yell at us. We have some main, and we want to switch x and y. They haven't done this demo in lecture yet, have they? They haven't? OK. So we're going to go over this very briefly. You'll get into this example more in depth, I think, this week. And then next week we can really dive into why this doesn't work.  So we have this void function here-- swap. So void just means that nothing is returned. And we have swap int a and int b. And we have some temporary variable that's a. a gets assigned to b, and then b gets assigned to the temp so that a and b's values are now switched.  But, plot twist, this doesn't work. And part of it actually has to do with the fact that a and b here, the ones that get passed in here, are actually copies of x and y. So when the function actually returns, it switches the copies but not the actual x and y's. So one way to think about it is that-- pretend these are swap. OK? So in main, we have x and y initialized. But when we actually go up to these frames with swap, we're passing the values over to it, and they're initialized.  And they only ever live right here. So a and b live here. And they get swapped. But when we return, we don't do anything with a and b. a and b leave with our function. And so x and y stay the same. You'll get more into how to fix that and how we actually deal with that later.  But it's just one thing to kind of keep in mind. Use it for the future. Don't worry if that didn't make all the sense in the world. They are copies is the biggest thing. If you're going to take anything away from that, you passed in copies. So the originals stay the same. Everyone good? Cool.  So command-line arguments. I'm sure in the beginning you guys all had those great, like, int main voids. And you're like, OK cool. I don't really care. This is just what I have to write. But in your new programs, especially in this pset, and why is there chalk on the ground?  With your next pset, you're going to be seeing this. Int main, int arg c, string arc v, brackets. So, from what we just learned today, what do we think that second parameter or that second element is here? It's an array. What type of array? String array, yes. Cool. So that's how you're going to be declaring these now.  Does anyone remember what these stand for? No? Hmm?  STUDENT: arg c.  ALLISON: So arg c keeps a counter. It's an int. It's a number, right? So what do you think that number is of? Yeah. So arg c is the number of strings that make up the command line. So if we were to do-- actually, there are examples after this, so I won't get ahead of myself. It's a number of strings that just make up your command line.  So when you do, like, dot slash Mario, that's one string that makes it up. In this piece, you'll actually be feeding things into the command line, as I'm sure you guys who have read the spec saw. So in those cases, maybe you'll have two or three arguments. It's going to be a useful thing to use.  And then arg v, as we said, is just a string array. So that actually stores what you input into the command line. So we have these. You have some dot slash copy infile outfile. So, if arg c is the number of strings that we're passing into the command line, what is our arg c in this case? Three. Exactly.  So what's arg v of zero? So what's the first thing we've stored? Dot slash copy, exactly. And then the second would be infile. The third would be outfile. So what about arg v three? It would be null, because that's the end of our array, right? Cool.  And then what about the sixth one? It's kind of a trick question. Ish. Do we know what it is? It's undefined. We have no idea what that could be. It's whatever is right after the array in memory, and we have no clue what that is. And it's dangerous to touch those things, because for all you know, it's some part of memory that you shouldn't be accessing or null. And it can do crazy things. It's called over-indexing your bound to your array. Don't go outside the bounds of your array, or bad things can happen. You come back and, like, the laws of physics have been destroyed or something. Cool. Does that make sense to everyone? Not too bad.  So now, everyone's favorite part, pset review. Yay! OK. So for those of you who haven't read the pset spec, you are doing some really cool stuff with cryptography. You're going to create a Ceasar Cipher and a Vigenere Cipher. You should definitely read the spec to see how those work. And if you're having any trouble about what it should actually be doing, please come talk to me, email me or text me. I'm around.  So there are three main things here that we want to talk about-- just kind of an extension of lecture. Things that you might not know about, helpful hints and tools. So we're going to do a quick review of ASCII, because that's going to be super important for Vigenere's Cipher. We're going to conversion of command line inputs, which will be very helpful for Caesar Cipher. And then modulo. Cool.  So, ASCII maps characters to numbers. This is a great chart. You should have this bookmarked somewhere. You will want it for your first mid-term. I'm pretty sure everyone has this chart on their mid-term sheet. So learn it. Love it. Keep it handy. It'll be useful.  And all it is is an encoding that maps alphabetic, numeric, and other characters to numbers for our computer. Because of course, in the end, everything we store is going to get converted down to zeroes and ones, so we need some way to represent the text and characters that we're all used to seeing as some sort of number.  So as we see here, we have uppercase A, which is right there. It's 65. And lowercase A is 97. So you can figure out-- as I said earlier, if you had array of multiple strings, what each of them have a null terminator. It would be a space. Space has its own special-- I forget where it is here. Ah. 32 is the space. So everything maps to it.  So we have ASCII math. Pro tip-- in Vigenere's, you might be tempted to convert your numbers to integers, but it's actually better practice to be able to use the characters like this when you're actually manipulating them. So if you want to use numbers, you can. But a better way, or a way that we tend to like you guys to do it, is this way where you're actually subtracting characters.  So I want you guys to kind of figure these out. Why don't you try every other one? So do the first one, the third one, and the fifth one. Because I want to make sure that we talk about everything we need to talk about. I'm just going to say, one of the important things to-- oh wait, you guys haven't seen this one. OK so do the first three. Let's do that. Because we have to talk about modulo. I know. Math is hard. You can use a calculator. It's OK. Or pull up an ASCII table, because you're probably going to want that.  Cool. So I will quickly walk you guys through these. So people have ASCII tables pulled up? What is our numeric number for lowercase A?  STUDENT: Seven.  ALLISON: So lowercase A is 97 and uppercase A is 65. So 97 minus 65?  AUDIENCE: 32?  ALLISON: 62, yeah. So in this case, what would it print out? That first one? If we have percent d, what would that indicate?  STUDENT: A number.  ALLISON: We're printing out an actual number. So we're actually going to print out 32 here. And if this were percent c, 32 would give us a space. So understanding that characters can be printed both as numbers and as the actual characters is really important, and paying attention to the actual types that we're doing here. Cool. So for every other one of these, what are we going to be printing?  STUDENT: A character.  ALLISON: A character. Cool. So if you guys want to know, you can work these out on your own. If you're having trouble, email me. But the second one will print out a lowercase b. The third one will print out an uppercase B. The fourth one will print out an uppercase C, and the last one will be a lowercase A. And the last one-- we're actually going to get into what that crazy percent sign even means in a couple slides. So try those on your own. If you have trouble, please come talk to me. If you're typically in Adams D hall, you'll probably find me around.  So, atoi. How many of you have seen this function or heard of it at all? Anyone? Cool. So what it actually stands for is ASCII to integer. So what you can do is, with Caesar, for those who read the spec, you're going to do dot slash Caesar after you write your program, and then you're going to input some number that you want to encode your secret message with.  But, if we remember, all of our inputs are stored as strings. Right? We have an arg v array that is all type string. So if you just try to pull that one, it would think that that one or whatever number you used is actually a character. So you're going to get some crazy results. So if you actually want to turn this into an integer that you can use to manipulate your word or your message, you'll want to use atoi. atoi just converts your string to an int.  So if we have a string of 12, if we call atoi on 12, or whatever that input is, it will actually return to you the integer. Not the character or the string 12. Which, when you start to add that to numbers, will be very different, because the string 12 is some crazy number in ASCII, but the integer 12 is actually 12, which is what you want. So you want to make sure to use atoi.  You're going to want this in Caesar, because you need the int supplied by the user in the command line. But when they put it in the command line, it's stored as a string to begin with. Does that make sense? You don't necessarily need this for Vigenere. With Vigenere, as I said before, you should try and use ASCII math that looks more like this, where you're actually using the chars that we're given to you. Cool. Everyone good there? Awesome.  So modulo. So what if you're given this huge number for Caesar? You have this idea that if you're at Z and you're given a number two, that means you need-- Z becomes the second letter after itself, right? So you need to somehow wrap around, and modulo is the way to do that. So all it does is it gives you the remainder of the division of the first number by the second. And we have some examples to make that a little more concrete. But basically, you use modulo when you want to make something wrap around.  So if you only want the numbers one through eight, you can use modulo on any other number, and it will always return a number from zero to eight. So some examples-- if we have 55 modulo 10, it just gives you the remainder of 55 divided by 10, which would be 5. And then three modulo five, anyone guess what that would be? Three.  So if you have a smaller number before the modulo, it can't go in evenly. It's zero. So it just returns the number itself. So eight modulo eight would be?  STUDENT: Zero.  ALLISON: Zero. Because it goes in evenly. 16 modulo 15?  AUDIENCE: One.  ALLISON: Cool. And then this last one is just to show you-- you might be wondering, OK, what's the order of operations here? Do we divide first? Do we modulo first? So modulo holds the same precedence as division or multiplication, and its left associative. So it's in the same way. You would always do parentheses, then multiplication, division, and modulo in order from left to right. So standard rules. Just put it in the same category as division and multiplication.  So in this case, we would have 1 plus 2 gives us 3. We multiply that by 2, so we get 6. We modulo that by 2, which gives us?  STUDENT: 0.  ALLISON: 0. And then we add 2, so we get 2 in this last case. So modulo-- you're definitely going to be thinking about ways to incorporate that when you're wrapping around the alphabet. If you're at Z and you need to move forward three spaces to get to C, there's that whole concept of wrapping around. So I will leave it to you guys to figure out how exactly you're going to be using it. But definitely a useful tool for your pset this week. I really like this. This is one of my favorite psets. Then after you do it, if you have friends, you can, like, send each other secret messages and make sure it works. Because it'll decrypt it or whatever. Lots of fun.  And that is the end of section. I finished early. I still have 15 minutes with you guys, so if there's anything that you would like to go over further, I'd be happy to do that. Any other questions on your pset for those of you who have started or read the spec. Anything that we've talked about in the last hour and 15 minutes that you'd like me to kind of rehash, I'd be happy to. Or we can call it quits, and you can all leave and take more candy with you as you go. But if there are any lingering questions, please let me know. You can also come up and talk to me afterwards. I promise I don't bite. Anything else? Everyone's good? Everyone's feeling like they can handle this pset? You're going to be fine guys. Office hours are there for a reason. Cool. Alright.  Well, in that case, thank you all so much for coming. I hope to see you next week. There will be more candy. There might be other cool things. And I look forward to getting to know all of you this year. | [ГРАЄ МУЗИКА] ЕЛІСОН БУХГОЛЬЦ-АУ: Ласкаво просимо на перше семінарське заняття з CS50. Сьогодні я просто покажу вам план семінарського заняття. Я не звичайний студент, а ваш помічник з навчання.  Я збираюся розповісти про хід наших семінарських занять. Ми будемо говорити про масиви, функції ASCII, аргументи командного рядка, а також, хоча й не тут, я допоможу вам продумати наступне практичне завдання на цьому тижні. За що, я думаю, ви будете вдячні.  Я Елісон Бухгольц-Ау. Я помічник з навчання на курсі CS50 вже другий рік, а також помічник з навчання на курсі CS51 навесні. Я також консультант для першокурсників і це мій третій рік як консультанта. Я добре розбираюся в консультуванні, курсах з комп’ютерних наук і за їх межами. Я мешкаю в Adams House - найкращому гуртожитку. Вчилася я на біомедичного інженера і збиралася піти в медичну школу. Я збиралася бути лікарем-травматологом. І все повністю змінилося, коли я прийшла на CS50. Я пройшла курс весною на другому році навчання. Це було моє перше знайомство з комп’ютерними науками. Я була однією з 78% серед тих, в кого був нульовий досвід, і це повністю змінило моє життя. Зараз я працюю в Microsoft і є вашим помічником. І CS50, ймовірно, одне з найкращих вражень, які я мала тут в Гарварді: під час проходження курсу і можливості допомагати студентам.  Отже, я дуже рада, що ви всі тут. Якщо ви запізнилися, ви можете підійти взяти цукерки, або попросити, щоби вам передали. Мені не хотілось би їх доїдати — в мене вдома повно шоколаду. Тож вам варто спробувати доїсти їх. Я знаю, там 100 штук, але о четвертій дня в понеділок трохи цукру не завадить.  Усі ті, хто офіційно записані на мої семінарські заняття, мають отримати лист з моїм номером телефону, електронною адресою - не соромтеся додавати мене до Gchat і в Facebook, а інші можуть писати мені прямо сюди. Там дві букви «H». Всі завжди пишуть дві «L», або дві «C». Дві «Н» в прізвищі. В іншому випадку я не отримаю листа.  Так що не соромтеся, пишіть мені у будь-який час. Я обіцяю, що відповім протягом 24 годин. Якщо ви зателефонуєте мені за півгодини до того, як ваше практичне завдання має бути готовим, і скажете, що не маєте поняття, що робити, я вас заспокою. Але, зрештою, якщо ви телефонуєте мені за півгодини до того, коли практичне завдання має бути готовим, і нічого не написано, я скажу, що це вже запізно. А так я буду відповідати на всі ваші запити своєчасно. Як правило, я не випускаю свій телефон з рук. Зазвичай я відповідаю швидше, аніж за 24 години.  Добре. Так чому ж ми тут? Якщо у вас є питання, ставте їх. Я розмовляю швидко, але не соромтеся мене перервати. Таким чином, семінарські заняття для нас – це час для того, щоби отримати практичний досвід, пройтися по темах, які ви вчили в аудиторії або в навчальних матеріалах он-лайн.  Пару зауважень щодо підтримки семінарських занять. Одна з особливостей, яка робить CS50 одним з моїх улюблених курсів - це відчуття, що ви ніколи не самотні. Більш ніж 100 наших співробітників тут, щоби вам допомагати. Консультації з понеділка по четвер. Більшість з наших співробітників – студенти, і курс іде як п'ятий предмет на додаток до іншої нашої роботи.  Ми займаємося цим, бо ми дійсно любимо навчати вас і допомагаємо розділити наше захоплення цим предметом і курсом. Тому, будь ласка, скористайтеся цією можливістю. Приходьте і розмовляйте з нами. Це буде чудово.  Таким чином, очевидно, що семінарські заняття - одна з найважливіших речей. Ми пройдемося по тому, що ви дізналися на лекції, зробимо кілька коротких прикладів, якщо буде час, і, взагалі отримаємо уявлення про речі, які вам будуть потрібні для виконання практичного завдання. Короткі відео - скільки людей насправді дивилися відео зі Скретчу? Хтось впізнав мене? Ви маєте це подивитися — на них пішло багато часу. Вони допоможуть отримати краще розуміння концепції. Практичні відео - скільки людей подивилося практичні відео за минулий тиждень? Тож обов’язково скористайтесь ними. Вони розбиті на дійсно маленькі та зручні фрагменти. І коли ви отримаєте цей величезний опис практичних завдань, дуже важливо мати можливість просто знайти десь інформацію, щоб розпочати роботу.  Гаразд, у нас також є Study50 - study50.harvard.edu. Ви можете просто загуглити study і вийти на нього. Це один з кращих наших ресурсів. Це презентації з нотатками і практичні завдання з розв’язаннями. Тож, якщо ви захочете більше практики, більше, аніж ми даємо у завданнях, це те місце, яке вам потрібно. Його створили минулого літа мої хороші друзі. Насправді, багато слайдів, які я буду використовувати для практичних занять, запозичені зі Study50.  Багато помічників по навчанню його використовують. Щодо консультацій - якщо у вас виникли проблеми з домашнім завданням, з розумінням, приходьте на консультацію. Приходьте на початку тижня, якщо можете. Це доволі далеко, але це у ваших інтересах, тому що тоді ви будете мати всіх цих помічників по навчанню, що вас оточують. Маленька порада - зараз по четвергах в Метер Хаус дуже спокійно, тому що ваші практичні завдання мають бути здані в четвер.  Тож є багато помічників по навчанню, які вам допоможуть. Так що приходьте до Метер Хаус в четвер. Якщо ви хочете бачити мене, я туди теж збираюся. Як правило, я займаюсь своїми справами, тому що ніхто не потребує моєї допомоги. Приходьте поспілкуватись. Тож, хто відвідав лекцію або подивився її онлайн? Хто дійшов до супер-семінару минулого тижня? Це доволі багато. Хто читав завдання цього тижня? О, народ, я пишаюся вами! Ще більше цукерок всім.  Семінарське заняття дійсно буде дуже корисним для вас, якщо ви прийдете, попередньо прочитавши своє завдання. Тому що, коли я буду робити огляд того, що вам може стати в нагоді, це не матиме для вас сенсу, якщо ви не знаєте, що повинні зробити у практичному завданні.    Тож корисніше для вас прийти на семінарські заняття. Я не сприйму за образу, якщо ви не прийдете, але вам варто було б. Якщо немає такої можливості, дивіться їх онлайн. Як ви помітили, нас записують, так що я теж там буду. Крім того, ходіть на лекції, де отримаєте початкові знання з матеріалу. Хоча я спробую допомогти вам, наскільки зможу, але я можу тільки направити вас. Ви повинні докладати зусилля самостійно.  Щодо оцінювання - всі ті, хто отримав листа від мене, офіційно в моїй семінарській групі. Я буду оцінювати ваші практичні завдання. І я хочу сказати одне - ви повинні звертати увагу на коментарі. Коментарі часто є кориснішими за ту оцінку, яку ми виставляємо, і забирають в мене багато часу під час перевірки. Тож я буду вдячна, якщо ви їх будете читати. Так ви дізнаєтесь більше про дизайн і стиль, і якісь більш нетривіальні речі.  Якщо у вас є питання щодо коментарів, або питання з приводу оцінки, будь ласка, приходьте поговорити зі мною, до семінару чи після. Я швидше за все буду в холі. Якщо ви хочете поговорити саме зі мною, як вдосконалити ці практичні завдання - просто дайте мені знати.  І тепер декілька порад для вас. Одна з найважливіших речей, на якій я наголошую під час семінарського заняття з програмування – спочатку напишіть код на папері. Якщо ви продумали, що має робити цей код, і розіб’єте його на маленькі псевдокоди, які ви запишете, буде значно менше шансів наробити синтаксичних помилок, чи зробити неправильну конструкцію *if*. Якщо ви знаєте, що маєте отримати в цілому, зменшується ймовірність зробити помилки, які іноді потребують години на виправлення, наприклад, - де це я загубив цю дужку?  Зазначте, будь ласка, використання Style50. Коли ви збираєтеся на консультацію, і весь ваш код відформатовано по одній стороні, згідно наших правил ми можемо сказати - виправте його так, щоби він виглядав відповідно до стилю Style50. Тоді ми вам допоможемо. Кожен отримає кращі оцінки. Хіба це не те, що ми всі хочемо?  Отже, пишіть на папері, перш ніж сідати за комп'ютер. Продумайте все наперед і переконайтеся, що ви знаєте свою мету. А якщо ви не впевнені, сядьте з кимось і продумайте крок за кроком те, що ваш код має зробити. 9 з 10 помилок - це забув поставити умову або забув крапку з комою, або перевизначив цю змінну неправильно. Це мої поради для успіху.  Оскільки приблизно половина з вас відвідала супер-семінари, я хочу коротко пройтися по циклах, про які не йшлося в нашому початковому розкладі, але вони дуже важливі. Перед цим, є які-небудь питання - технічні, особисті, будь-що, що ви ще хочете знати про мене чи про семінарські заняття або курс в цілому? Прекрасно.  Отже, цикли – ви всі повинні впізнати їх зі Скретчу. Цикли - це спосіб, щоб зробити щось деяку кількість разів, деяку повторну дію, що базується на певних умовах. Є три різних типи. У нас є цикл *for*, *while* та *do-while*.  Для циклів *for* ми маємо досить загальний макет. Чудово, якщо ви знаєте, скільки разів потрібно щось виконати. Коли ми будемо говорити про інші типи циклів, ви побачите, чому це важливо. Цикли *for* застосовуються для чогось стабільного. Ви одразу знаєте або можете підрахувати кількість необхідних повторів ще до початку циклу.  Це загальна схематична структура циклу *for*. Ініціалізація відбувається в блоці *for-initialization*. Я впевнена, що з Маріо ви робили щось на кшталт *int і* = 0. Це те, що має бути в синьому блоці. У вас є умова, що перевіряється кожного разу. Якщо ця умова істинна, то інша частина коду виконується. Тоді він працюватиме знову і перевірятиме умову. Потім йде оновлення, де ви оновлюєте вашу змінну. Знову ж таки, з Маріо ви робили щось на кшталт *і*++ (“і плюс плюс”). Таким чином, кожен раз при проходженні циклу ми перевіряємо умову і змінна оновлюється, якщо відповідає цій умові.  Якщо у вас статична змінна і код виконується вперше, він буде виконуватися безкінечно. Ви повинні переконатися, що змінна оновлюється належним чином. У нас є графічна презентація. Ви мали бачити цикли *for* в своїх практичних завданнях.  Ось простий приклад. Надрукувати «*This is CS50*!» 10 разів. Є ініціалізація *int i* дорівнює *0*, де *і* менше за 10, *і*++. Це те, що надрукує фразу 10 разів.  Цикли *while* краще використовувати, якщо ви не знаєте одразу, скільки разів вам необхідно повторюватись. Є тільки умова, що перевіряється. І це може бути щось на зразок *while* — візьмемо приклад з вашого практичного завдання. Якщо ви спробужте ввести від'ємне число. Вам потрібно надати повторний запит користувачеві. Якщо користувач вводить щось менше нуля, спитай його ще раз. Я впевнена, що деякі з вас використовували цей цикл у вашому коді.  Це просто. Такий тип циклу використовується при умові, що має перевірятися перед кожним запуском коду. Якщо значення істина, код запускається. В іншому випадку - ні. Дуже важлива річ – те, про що Девід казав в лекції, – фігурні дужки. Виконується весь код всередині фігурних дужок. Якщо ви забудете їх поставити, буде виконуватися тільки рядок безпосередньо після *while*.  Отже, є три речі, що мають відбутися, якщо умова дорівнює істині, і якщо ви не поставите фігурні дужки, виконається тільки перша. Тож будьте з ними уважними. Найкраще для вас дотримуватися стилю Style50.  Отже, це — зворотний відлік від 10 до 0. Як бачите, ми ініціалізуємо змінну за межами циклу. Це відмінність циклу з передумовою. Ми ставимо умову в блок *while*. У цьому випадку умова виконується, поки лічильник більше нуля. Ми друкуємо значення нашого лічильника, а потім зменшуємо нашу змінну на одиницю. Зверніть увагу: оновлення відбувається не в першій частині циклу з передумовою, а у фігурних дужках, у тілі циклу.  Цикли *do-while* можна використовувати для перевірки вводу. Можливо, дехто з вас використовував це в практичних завданнях. Ви можете надіслати запит на введення числа і задати умову, що введене число менше якогось заданого.  У випадку з Маріо це буде *do* *printf,* ввести ціле число, і потім деяке число дорівнює *getint*. Тоді спочатку буде запущений код. Ви отримаєте якесь ціле число і зможете визначити, чи воно менше нуля. Код виконається принаймні один раз. Потім перевіряється умова і якщо умова істина, код буде працювати знову.  Тож *do-while* цикли чудово підходять для перевірки вводу, тому що ви знаєте, що код виконається принаймні один раз, тоді як цикл *while* може не виконатись жодного разу. Він спочатку перевірить умову, а потім вирішить, чи виконувати код, а цикл *do-while* виконає код, а потім перевірить, чи потрібно його повторювати. Ця різниця всім зрозуміло? Добре.  У цьому прикладі це те, про що я казала, - повторення запитів користувачу, поки ви не отримаєте додатне число. Тобто *printf "введіть позитивне число"*, що виконує цей запит, виведеться принаймні один раз. Якщо користувач не в гуморі і продовжує вводити від’ємне число, хтозна, скільки разів буде виконуватись ця частина коду. Але він точно виконається принаймні один раз.  Ви будете користуватись цим не раз.  Гаразд, будь-які питання? Все добре? Я говорю занадто швидко? Добре.  Поговоримо про масиви. Масиви – це структура даних, яка дозволяє нам зберігати елементи одного типу. Ваш масив може складатися або чисел типу *int*, або дійсних чисел типу *float* , або з символів типу *char*. В одному масиві не може бути одночасно цілих, дійсних і символьних даних. Він має визначений розмір і тип.  Тож у нас є масив розміром в три елементи з трьома цілими числами в ньому. Важлива річ, яку необхідно зрозуміти, це те, що масив має визначений розмір, коли ви його ініціалізуєте. Змінити його розмір не просто, а поки ви не впевнені в цьому, - думайте, що це неможливо. Якого розміру ви створите свій масив від початку, таким він і залишиться, оскільки масиви є безперервними елементами в пам'яті. Це стане зрозумілішим, коли ви отримаєте більше інформації про роботу пам’яті на диску і що таке купи та стеки.    Ви можете просто уявити масив, як ряд комірок на диску. Нема гарантій, що там буде вільний простір після оголошення масиву. Ви можете ініціалізувати масив з трьох елементів, а потім інший масив з п'яти елементів. Якби ви пройшли через третій елемент першого масиву і почали щось записувати, то могли б потрапити вже в інший масив. Тож сприймайте їх, як структури фіксованого розміру.  Отже, ваше завдання - створення. Як і для циклів, тут є загальна структура для масиву. Оскільки всі елементи масиву одного типу, при створенні масиву вам потрібно визначити цей тип.  Як бачите, в маленьких дужках вказаний тип даних. Якщо ми створюємо масив цілих чисел – тип буде *іnt*. Якщо ми створюємо масив символів, то це буде *char*. Якщо ми створюємо масив символьних рядків, це буде *string*. Потім задається назва вашого масиву – будь-яка. Це можуть бути результати тестів, студенти або цукерки. Все на ваш вибір.  В квадратних дужках вказується розмір масиву. Будемо зберігати 10 студентів чи 15 типів цукерок? Будь-що. Тут створюється масив з трьох елементів, як ви бачите справа. При першій ініціалізації всі елементи дорівнюють нулю, як з чистого аркуша. Ці комірки – це місця, в яких би могли зберігатись наші дані, але вони поки що порожні.  Отже, якщо нам необхідно приписати їм якесь значення, ми робимо це ось так. Необхідно вказати назву масиву, потім індекс елементу, якому ви хочете приписати значення. Необхідно пам’ятати, що індексація масиву починається з нуля. Тобто перший елемент нашого масиву буде мати індекс 0. Другий – індекс 1. Третій – індекс 2. І так далі.  Це те саме, що ми робимо в циклах *for*. Якщо вам цікаво, чому ми починаємо з 0, а не з 1, то це тому, що, коли відбувається перехід до користування масивом, він відображається правильно. Якщо потрібно робити пройти по масиву, має сенс зробити *i* рівно 0, що відповідає першому місцю в пам’яті. Всім зрозуміло? Добре.  Тут внизу показаний ще один спосіб ініціалізації масиву. Знову є тип даних та ім'я, розмір масиву не вказується, лишаємо порожні дужки. А потім у фігурних дужках вводяться дані, які ви хочете додати до масиву, розділені комами. І це автоматично буде означати: добре, я бачу, що у вас є три елементи в цих дужках, так що я знаю, що мені потрібно виділити три комірки в пам'яті і зберегти в них елементи.  Тож перший варіант ви можете використовувати, якщо є необхідність надіслати запит користувачеві на вхідні значення, так що ви можете переміщатися по масиву і у відповідь на запити отримувати значення і заповнювати масив. Якщо ви заздалегідь знаєте значення, краще використовувати другий варіант. В більшості випадків,значень ви знати не будете.  Одна з переваг масивів - вони дають вільний доступ до будь-якого елементу, вам не потрібно перевіряти кожну комірку. Якщо ви знаєте, що вам потрібна комірка №2, можна просто отримати її значення. Тому індекси дуже важливі, бо ми отримуємо доступ саме через них. Так, як ми бачили раніше, коли присвоювали значення елементу за допомогою індексу, так само ми через індекс дістаємо значення елементу. Для цього потрібно лише назву масиву і індекс необхідного елементу.  Хто скаже, що робить цикл *for* тут знизу? Точно. Це просто перебір значень, *і* дорівнює нулю – ми просто швидко проходимо код. *і* дорівнює нулю, і менше трьох, правильно? Умова виконується. Добре, хай виводить “*temperature і*”. *і* дорівнює нулю на початку ітерації, тому першим друкуємо значення з першої комірки – 65. Потім переходимо на новий рядок, значення індексу оновлюється і друкується 87. Знову значення оновлюється, і з нового рядка друкується 30. Зрозуміло?  Добре. Я вже казала, як ви можете зберігати оцінки нашого класу і чому необхідно використовувати перший спосіб ініціалізації замість другого. Зверніть увагу, розмір нашого класу 30. І ми ініціалізуємо цей масив цілих чисел розміром 30. Потім методом перебору просимо користувача ввести оцінки кожного, а потім присвоюємо їм місце в пам'яті цього масиву. Тут все зрозуміло?  *Define class size* 30 — це директива препроцесора, яка означає, що він має робити в процесі компіляції. Ви можете розглядати це, як глобальну змінну. Це дозволяє легко змінювати ваш код. Якщо наш клас раптом змінить свій розмір з 30 на 15, якби це не було визначено таким чином, мені б довелося перейтися через всю програму і змінити кожен екземпляр з 30 на 15. А так я міняю тут і решта так само зміняться.  Якщо хочете використовувати *define* для відстеження набору оцінок класу або будь-якого числа, набагато краще було б визначити це число на початку програми, щоби, якщо воно зміниться, ви б вносили корективи в одному місці замість 100. Так?  СТУДЕНТ: Зробивши це і просто оголосивши [НЕРОЗБІРЛИВО] на самому початку.  ЕЛІСОН БУХГОЛЬЦ-АУ: Він має підвищувати ефективність в процесі компіляції - це трохи за межами нашого семінарського заняття. Якщо ви хочете дізнатися більше, я із задоволенням відправлю вам листа зі списком ресурсів по цій темі. Хеш-функція визначає те, що має перевагу. Коли у вас буде більше досвіду, ви дізнаєтесь, коли краще використовувати глобальну змінну чи хеш-функцію. Якщо ви хочете використати хеш-функцію, варто зауважити, що ім'я має бути у верхньому регістрі. Воно так написано не для драматичності, воно дійсно має бути великими буквами. Щось ще? Прекрасно.  Будь ласка.  ОК, я хочу, щоб ви глянули на це і знайшли помилку. Я дам вам підказку. Вона десь в циклі. Так?  СТУДЕНТ: Має бути менше або дорівнювати 2.  ЕЛІСОН: Так, має бути менше або дорівнювати 2, або може бути менше 3. А чому?  СТУДЕНТ: [НЕРОЗБІРЛИВО], 0, 1, 2.  ЕЛІСОН: Саме так. Таким чином, в масиві розміру n є індекси n-1. Добре. Також ми можемо трохи збожеволіти і створити багатовимірні масиви. Це була одна з проблем того року, коли я проходила курс, і я думаю, що цього року вам також це знадобиться. Ви ще натрапите на це. Сприймайте багатовимірний масив, як масив масивів. Таким чином, ви маєте першу комірку пам’яті в верхньому ряду, другу – в другому ряду і останню – в третьому ряду. В кожній комірці зберігається масив. Це легше зрозуміти в схематичному вигляді, як тут.  Ініціалізується він таким же чином. Це символьний масив «дошка» (board) три на три. В нас є три рядки і три стовпці. Отримати доступ можна так само через стовпець і рядок. І так, комірка 1,1, як тут. Присвоїмо їй 0, і 0 сюди. 2,0 і 0,2. Отримаємо доступ до комірок - якщо хто-небудь вивчав лінійну алгебру, ідея така ж, як доступ до елементів в матриці. Так що можна згадати математику. Достатньо просто мати розуміння, щоби це зробити. Тепер ви можете створити будь-який божевільний масив. [НЕРОЗБІРЛИВО]. Трошки божевільно, але дійсно круто.  Ось ще один приклад. Це обчислення довжини рядка. Хтось знав, що рядки, які ви використовуєте, це просто масиви символів? Ви можете думати, що не використовували масиви раніше, але ви постійно користуєтесь функцією GetString в бібліотеці CS50 і надсилаєте запит на масив символів. А ми беремо на себе те, що ви отримуєте в результаті.  Кожного разу, коли у вас є масив символів або масив, який зберігає рядок, в кінці завжди є елемент, який називається символом кінця рядку. Він є в кінці кожного слова, яке ви зберігаєте. Тобто, якщо ми хочемо з'ясувати довжину рядка, ми знаємо, що вміст цього блоку не дорівнює символові кінця рядку. Це означає, що є символ, який є частиною слова і він збільшує його довжину. Коли ми досягаємо цього символу, це означає, що ми досягнули кінця слова, і він повертає нам його довжину. Так?  СТУДЕНТ: Чи рахується пробіл, як символ кінця рядку?  ЕЛІСОН: Ні, пробіл не є символом кінця рядку. Так що, якщо у вас є кілька – пробіл, насправді, має специфічне ASCII значення. СТУДЕНТ: Що таке знак оклику і дорівнює?  ЕЛІСОН: Це те, з чим ви порівнюєте. Якщо ви коли-небудь були в мене на консультаціях, я завжди називаю це вибуховим дорівнює. Це «не дорівнює». Якщо ви намагаєтеся визначити, чи є щось хибним, ви можете використовувати цей оператор. Якщо воно визначається, як істина, ви можете робити з ним класні речі. Детальніше про це пізніше. Круто. Все добре? Чудово.  Тож прийшов час попрацювати. Створіть масив з цілими числами один, два, і три, і виведіть їх на екран. Просто ініціалізуйте масив, а потім створіть цикл з лічильником або цикл з передумовою для виведення на екран – як завгодно.  Даю вам кілька хвилин, щоб попрацювати над цим. А я відпочину. Якщо у вас є які-небудь питання, я з радістю з вами поговорю. Не соромтеся говорити один з одним. Тримайте ще цукерок. Я тут пройдуся з цукерками.  Ви хочете? Хто-небудь ще в цій кімнаті хоче цукерок? Ви можете брати більше ніж одну. Візьміть жменю, якщо хочете.    Я збираюся створити анонімну форму Google, і ви зможете лишити свої враження після кожного семінару, якщо є щось, що треба покращити або додати. Якщо я занадто жвава для вас, я можу трохи знизити тон. Я створю її і відправлю вам згодом.  Добре. Отже, давайте потихеньку починати. Як ми ініціалізіуємо наш масив? Який тип нашого масиву? int, так? Як ви хочете назвати наш масив? Int array, круто. Добре, тож ми маємо int int array =, і що після цього?  СТУДЕНТ: [НЕРОЗБІРЛИВО] дужки.  ЕЛІСОН: Фігурні дужки. А потім в фігурних дужках? Один кома, два кома, три. Круто. Так що все в порядку. Тепер має бути цикл з лічильником. Що в першій частині циклу?  СТУДЕНТ: і = 0?  ЕЛІСОН: Так, int і = 0, яка в нас потім умова? Менше чого має бути і? Менше ніж три, і як ми будемо змінювати і? і++, додаючи до нього по одному. І тоді ми матимемо ціле число для функції printf, і що буде в останній частині, яка буде визначати, що саме виведеться на екран? Це буде назва масиву, тобто int array, так? І що в дужках? і. Мій масив називається example, у вас може бути інакше.  Отже, ми закінчили з масивами. Вітаю. Вам вдалося перебрати всі значення -- так?  СТУДЕНТ: [нерозбірливо]  ЕЛІСОН: Так.  СТУДЕНТ: У мене є питання. Потрібно робити відступи для фігурних дужок?  ЕЛІСОН: Дужки мають бути вбудовані в цикл з лічильником, і все, що знаходиться в дужках, має бути з відступом.  СТУДЕНТ: ОК, цикл має бути з відступом?  ЕЛІСОН: Цикл не має бути з відступом в цьому випадку. Як би ми були в середині main – основної функції, він мав би мати відступ. Але в цьому випадку можна так. Так, питання.  СТУДЕНТ: Чи потрібні дужки після example?  ЕЛІСОН: Так, якщо ви його ініціалізуєте таким чином. Пам’ятайте, що при другому способі ініціалізації масиву, де є фігурні дужки, наші фактичні дані знаходяться всередині, розділені комами.  СТУДЕНТ: Я думала, що для цього використовуються квадратні дужки.  ЕЛІСОН: Ні, фігурні. Якщо ви його ініціалізуєте другим способом – це фігурні дужки. Якби ми хотіли отримати пустий масив для цілих чисел, було би int example [3] – 3 в квадратних дужках. В таких дужках визначається розмір. У фігурних дужках знаходяться фактичні значення Ми можемо дуже швидко прокрутити назад.  Отже, це - початковий масив, ініціалізація. Ми індивідуально присвоюємо значення елементам масиву, вказуючи їх індекс в квадратних дужках. Але тут, якщо ви помітили, ми залишили наші дужки з розміром без значення, - ми присвоюємо значення і визначаємо розмір масиву.  СТУДЕНТ: Так чому не має квадратних дужок в цьому прикладі? ЕЛІСОН: Де саме?  СТУДЕНТ: Хіба не має бути, int example квадратні дужки дорівнює фігурні дужки [НЕРОЗБІРЛИВО], наприклад.  ЕЛІСОН: О, вибачте. Ви маєте рацію. Тут мають бути квадратні дужки. Вибачте, моя помилка. Повинні бути дужки після example.  СТУДЕНТ: [НЕРОЗБІРЛИВО] не робити цього.  ЕЛІСОН: Ні, вони повинні бути обов’язково, бо в іншому випадку масив не буде оголошений.  СТУДЕНТ: [НЕРОЗБІРЛИВО]. Вибачте.  ЕЛІСОН: Вибачте, дужки після example потрібні. Друкарська помилка. Добре підловили, для вас золота зірочка. І ще, коли ставите питання, кажіть мені ваші імена, мені це подобається. Я би хотіла знати всі ваші імена. Я не збираюся вас викликати, просто хочу знати ваші імена.  ЛЕА: Леа.  ЕЛІСОН: Леа. Отже, функції - це свого роду, такі місця, куди ви вставляєте свої вхідні дані, і – чудо! - отримуєте вихідні дані. Круто. Ви використовували вже багато функцій. Get int, get string, print f. В цих функціях, коли ви їх використовуєте, в фоновому режимі відбувається щось, що ви навіть не бачите, і на виході ви отримуєте те, що хочете.  Основною метою функцій, а також однією з головних тем CS, є необхідність розділити код на частини, якими зручно керувати. Коли ви починаєте писати ці дійсно великі програми, або в Скретчі у вас є чудова ідея для гри, вам необхідно розбити її. Як почати? Які маленькі частини коду потрібні? О, треба щось запитати у користувача. Тепер мені потрібно щось вивести на екран. Мені потрібно обчислити це значення.  Навчитися розділяти свій код і великі завдання, над якими ви працюєте, на маленькі частини, і створювати функції є насправді одним з наріжних каменів CS. Функції – це як чарівний чорний ящик, в який ви кладете вхідні дані, а отримуєте вихідні. Іншій частині програми не обов’язково знати, що відбувається всередині. Вона відповідає тільки за те, що в нього входить і те, що виходить. Круто.  Чому функції? Організація - як я вже казала, коли ви маєте справу з великими базами коду, організація вашого коду буде набагато легшою за умови використання функцій. Ви будете розуміти, що робить та чи інша функція. А також буде добре видно, чи вони підходять одна одній. Тому це зручно – розбивати все на частини.  Спрощення - я впевнена, всі бачили це на прикладі Скретчу. Ви вже маєте загальне розуміння і як це працює? Якщо ваш підхід поступовий, скажіть, як мені зробити так, щоби один спрайт переміщувався по екрану? Виважене використання функцій робить ваш код більш читабельним. Це робить його легшим для відладки – з чим ви зустрінетеся в майбутніх практичних завданнях. Функції також легше спроектувати і реалізувати. Ви можете написати невелику функцію відносно швидко і переконатися, що вона працює, замість того, щоб створювати цілу довгу програму, а потім шукати, що в ній працює, а що ні.  І повторне використання. Функцію потрібно написати тільки один раз, а потім ви можете використовувати їх стільки разів, скільки це можливо. Тож це екологічно, в якомусь сенсі. Якщо у вас є якась функція на зразок print f, вам не потрібно кожного разу виписувати її, коли потрібно щось вивести на екран, інакше ви б дуже втомився від цього.  Одна з речей, які ви дізнаєтеся пізніше, займаючись на CS, і одна з найкращих порад, яку я отримала - якщо ви копіюєте і вставляєте код, це має бути функція. Якщо у вас є одні і ті самі рядки протягом всього коду, він буде в п'ять разів коротший і набагато зручніший, якщо ви їх винесете. І замість того, щоб шукати проблему в різних місцях, є одна функція, яку можна перевірити і усунути помилку. І я обіцяю, що пізніше, коли ви зануритеся в майбутні практичні завдання, функції матимуть набагато більше сенсу і допоможуть легше з ними впоратися. Чому ми їх використовуємо? Я знаю, що ми ще не пояснювали основні моменти. Тож визначення функції, як і масивів - це загальні вихідні дані. Це функція, яка підносить в третю степінь якісь вхідні дані. Щось видно, просто цікаво? Я знаю, фіолетовий і чорний - може бути трохи важко.  Треба запам’ятати важливу річ, спершу – тип значення, що повертається. Тож якщо вхідними даними є число, то на виході ми отримаємо це число в кубі. В нашому випадку це буде int. Воно може стати бути double чи чимось іншим пізніше, але зараз це int.  На с завжди потрібно вказувати тип значення, що повертається. Це буде int, float, double. Ви повинні вказати, що саме ця функція буде повертати. В іншому випадку програма не зкомпілюється.  Тут є ім'я функції. І, як ви бачите, на с дуже послідовна модель - який має бути тип, яке ім’я, і кілька інших речей. Тож у нас є тип значення, що повертається, ім’я функції, а також заголовок з переліком параметрів.  Перелік параметрів - це те, що ця функція буде використовувати? Перелік параметрів є просто синонімом наших вхідних даних? Ми повинні визначити нашу функцію і тип значення, що повертається, і кожен вхідний елемент має отримати пов’язаний з цим тип.  У цьому випадку у нас int на вході. Ще раз, це буде тип і ім’я. А тут, як ви бачите, у нас є тіло функції. Отже, ми маємо якийсь int на виході, це всього лише наші вхідні дані множать себе 3 рази – підносять в куб. І тоді ми повертаємо значення, що отримали, на вихід.  Отже, як ви бачите, у нас є int помножений на int помножений на int, що повертає нам також int, і це було оголошено ще там. Тож все пов’язане. У вас завжди є тип значення, що повертається, назва і перелік параметрів. Кожен елемент в переліку параметрів чи вхідні значення мають мати тип, пов'язаний з ним. Тоді в рамках функції ви можете робити з вхідними значеннями що завгодно.  Очевидно, що ви захочете отримати щось на виході. Деякі функції нічого не повертають. Але якимось чином вони повертають результат. Коли ви будете робити власні функції, ми зможемо більше розібратися в цьому. Все добре? Є щось, що вам не зрозуміло і мені потрібно перейтися ще раз? Все добре? Круто. Чудово.  Отже, підсумовуємо. У нас є функція int cube, це повна програма. Досі я давала вам обривки, що були частинами програми. Але ми вже розглянули функції. Ось ціла програма. Тож, хто пам’ятає прототип з лекції? Круто. Є один. Як вас звати?  СТЕФ: Стеф.  ЕЛІСОН: Стеф? ОК, чудово. Тож ви пам'ятаєте, що таке прототип?  СТУДЕНТ: Ви говорите [НЕРОЗБІРЛИВО] перш, ніж насправді працювати з цим.  ЕЛІСОН: Ви пам'ятаєте, чому?  СТУДЕНТ: Ні.  ЕЛІСОН: Ок. Молодець. Прототип був перед цим, бо в іншому випадку наш компілятор був би незадоволений. Він би не розумів, чому тут це функція cube. Це так само, коли ви заходите в клас, і вам кажуть, що сьогодні тестування. А вас не попереджали. Прототип начебто каже: готуйся. У цей день буде тест. Не хвилюйтеся, все буде в порядку.  Тож те, що робить прототип – він каже main (головному методу), що збирається використовувати цю функцію і просить зкомпілювати цю функцію, яку він визначить пізніше.  Таким чином, в цьому випадку main – це головна функція. Ми ініціалізуємо якесь ціле число х. Це два. Виводимо на екран значення х. Збираємося піднести х в куб. Як бачите, функція оголошена тут, це те, про що ми говорили раніше. Вона піднесе х в куб, а потім функція cube поверне нам ціле число, яке збережеться в х знову і ми зможемо вивести на екран вісім або х в кубі прямо зараз. Зрозуміло? Добре.  Добре. Це візуалізація, як пам’ять тут зберігається. Таким чином, ми хочемо переконатися, що ви розумієте, як це виглядає в пам'яті. Якщо ви приєднаєтесь до курсу CS61 або старших курсів, ви будете вивчати це глибше. Але зараз я даю вам загальний огляд, тож вам не потрібні подробиці.  Зверху просто текстовий фрагмент, який містить нулі і одиниці, бінарний запис. Це використовується для зберігання глобальних змінних, якщо такі є. Унизу бачимо ініціалізовані дані, неініціалізовані дані, а потім купу.  Ми насправді не говоримо про купу прямо зараз, а повернемося до неї пізніше. Але ми поговоримо про стек. Стек буде там, де ми зробимо зум. Це, власне, те, як програма, яку ми тільки що розглядали, виглядає в пам'яті.  Кожного разу, коли ми викликаємо функцію - ми отримуємо те, що називається стековий кадр. Це - основні параметри, які ми передаємо в main. Таким чином, тут внизу показано, що це - перше, що ми викликаємо. І тоді в main ми отримуємо локальні змінні, які там зберігаються . Тож локальною змінною тут буде х, що дорівнює двом в цьому випадку. Це локалізується в main.  Потім у main ми викликаємо функцію cube. Правильно? Ось ще один кадр з параметрами функції cube. Таким чином параметрами функції cube тепер стали х і два, які ми пройшли. Вони стали локальними змінними cube, де фактично відбувається піднесення в третю степінь. А потім повертається отримане значення.  Тож cube повертає значення. Коли він повертає це значення, цей кадр йде повертається в main. А вже в main ми можемо вивести його на екран. Тож коли ваша функція щось повертає, це ніби передавання цих значень в стековий кадр під ним, а сам цей кадр йде геть. Все має виконуватися по порядку.  У більших програмах ми зможемо зробити крутіші і складніші діаграми. Зараз це просто загальний огляд, щоби ви мали уявлення про те, що відбувається, коли ви викликаєте функцію, і як це виглядає в пам'яті. Круто? Все добре? Чудово.  Це програма, що міняє елементи місцями. Як бачимо, у нас є прототип функції, щоби компілятор не сердився. У нас є main, і ми хочемо поміняти х та у. Ми пройдемося по цьому дуже коротко. Ви розберетеся в цьому прикладі детальніше на цьому тижні. І тоді наступного тижня ми дійсно зможемо зрозуміти, чому він не працює.  Отже, ми маємо функцію з типом void (що нічого не повертає). У нас є функція swap з параметрами int a і int b. Є якась тимчасова змінна, якій присвоєне значення а. Потім а присвоюється значення b, а b присвоюється tmp, тож a і b тепер помінялися.  Ага, і раптом воно не працює. Частково це пов’язано з тим, що a і b ось тут, що перейшли сюди, є насправді копіями х та у. Тому коли функція фактично повертає значення, вона міняє місцями копії, але не x та y. Ми можемо поміняти це. OK? Таким чином, у нас є ініціалізовані х і у. Але коли ми підіймаємося на цей кадр з функцією swap, ми передаємо значення йому, і вони ініціалізуються тут.  Тож a і b знаходяться тут. Вони міняються місцями. А коли ми повертаємося, ми нічого не робимо з a і b. а і b йдуть разом з нашою функцією. А х та у лишаються тими ж самими.  Але пам’ятайте, що вони є копіями. Якщо ви збираєтеся передати щось звідси, ви передасте копії. Таким чином, оригінали залишаться без змін. Все добре? Круто.  Тож аргументи командного рядка. Я впевнена, що на початку у всіх були ці int main voids, і вас це не турбувало. Це було те, що потрібно переписати. Але у ваших нових програмах, особливо в цьому практичному завданні, і чому крейда на землі?  В вашому наступному практичному завданні ви побачите. Int main, int arg c, string arc v, дужки. Що з цього ми дізналися сьогодні, як ви вважаєте, що таке другий параметр або другий елемент тут? Це масив. Який тип масиву? Масив рядків, так. Круто.  Хтось пригадує, що це означає? Ні? Так?  СТУДЕНТ: arg с.  ЕЛІСОН: Тож arg с містить лічильник. Це int. Це число, так? Це число чого, як ви думаєте? arg с – це число символьних рядків, які складають командний рядок. Це кількість символьних рядків, які складають ваш командний рядок.  Тому коли ви пишете: . / Mario - це один символьний рядок. У цій частині ви будете вводити елементи в командний рядок, і я впевнена, що ті, хто читав завдання, бачили таке. В деяких випадках, ви можете мати два або три аргументи. Це практично.  І arg v - це просто масив рядків. Він зберігає те, що ви вводите в командному рядку. Отже, у вас є . / copy infile outfile. Таким чином, якщо arg c - це число рядків, які ми вводимо в командний рядок, яким буде arg c в цьому випадку? Три. Точно.  Що буде в arg v з індексом нуль? Що ми зберегли першим?  . / copy, точно. Другим буде infile. Третім буде outfile. Як щодо arg v з індексом три? Він буде порожнім, тому що це кінець нашого масиву. Круто.  А що в номері шість? Це каверзне питання. Це будь-що, що зберігається в пам'яті відразу після масиву, і ми не знаємо, що це таке. Тут знаходиться та частина пам'яті, до якої ви не повинні отримувати доступ. Це називається вихід за межі масиву. Не виходьте за межі вашого масиву, бо можуть трапитися погані речі, а то й порушитися закони фізики. Круто. Всім зрозуміло? Непогано.  А тепер, ваша улюблена частина - огляд практичного завдання. Ура! Для тих з вас, хто не читав специфікацію: ви зробите деякі дійсно класні речі з криптографії. Ви маєте створити шифр Цезаря і шифр Віженера. Вам треба прочитати специфікацію, щоб побачити, як це зробити. У випадку проблем чи недорозумінь, приходьте поговорити зі мною, надішліть мені повідомлення електронкою або на телефон. Я поряд.  Тож три основні речі, про які я хочу поговорити – продовження теми, що була на лекції. Ми зробимо короткий огляд ASCII, тому що це важливо для шифру Віженера, і розберемо перетворення вхідних даних командного рядка, які будуть дуже корисні для шифру Цезаря. А потім перейдемо до ділення з остачею (ділення по модулю).  ASCII (Американський стандартний код для інформаційного обміну) відображає символи числами. Ця діаграма знадобиться вам для тестів в середині семестру.  Це кодування, яке переводить літерні, цифрові та інші символи в числа для нашого комп'ютера. Зрештою, все, що в ньому зберігається, зводиться до нулів і одиниць, тому ми повинні перетворювати текст і символи, до яких звикли, в якесь число.  Отож, тут велика А - це 65, мала - 97. Таким чином, ви можете з'ясувати - як я вже казала раніше, якщо у вас масив з кількох рядків, то кожен з них має символ кінця рядку, наприклад, пробіл. Пробіл має своє власне значення - 32. Все переводиться тут.  Отже, це математика ASCII. Порада - у шифрі Віженера ви могли б спробувати конвертувати ваші номери до цілих чисел, щоб мати можливість використовувати символи при роботі з ними. Можна використовувати номери, але краще для вас перейти до віднімання символів.  Спробуйте. Чому б не спробувати через один? Спробуйте перший, третій і п'ятий. Я хочу переконатися, що ми проговорили все, що потрібно. Я просто хочу сказати одну важливу річ - Ой, почекайте, цього ви ще не бачили. Тоді давайте перші три. Ми маємо ще поговорити про ділення по модулю. Математика складна, але ви можете використовувати калькулятор. Або відкрийте таблицю ASCII, вона може знадобитися.  Круто. Я швиденько пройдуся по цьому. Ви відкрили таблицю ASCII? Яке число відповідає малій а?  СТУДЕНТ: Сім.  ЕЛІСОН: Тож велика А 97, мала - 65. 97 мінус 65?  АУДИТОРІЯ: 32?  ЕЛІСОН: 62, так. Що зараз першим виведеться на екран? Якщо у нас є %d, що це означає?  СТУДЕНТ: Число.  ЕЛІСОН: Ми виведемо на екран саме число – 32. Якби тут було %с – це був би пробіл. Розуміння того, що символи можуть бути виведені як у вигляді чисел, так і символів є дійсно важливим, звертайте увагу на фактичні типи, які вказані тут. Таким чином, що ми виведемо на екран для решти?  СТУДЕНТ: Символ.  ЕЛІСОН: Символ. Тож ви можете пропрацювати це самостійно. Якщо виникнуть проблеми, пишіть мені. Другий рядок виведе малу b. Третій - велику В. Четвертий виведе велику С, і останній – малу а. І останній – ми розберемо, що означає цей знак відсотків через пару слайдів. Спробуйте розібратися самостійно. Якщо ви буваєте в холі Adams D, то можете знайти мене там.  Отже, atoi. Хто з вас чув про цю функцію? Це функція для конвертації ASCII в ціле число. У шифрі Цезаря (для тих, хто читав специфікацію), ви поставите ./Caesar після написання програми, а потім введете якесь число, щоб закодувати своє секретне повідомлення.  Якщо ви пам'ятаєте, наші вхідні дані зберігаються у вигляді рядків. У нас є масив arg v, який має тип string. Якщо ви спробуєте просто вибрати одиницю, він подумає, що одиниця або будь яке інше число, яке ви використаєте – це символ. Ви отримаєте божевільні результати. Якщо ви захочете перетворити одиницю на ціле число, щоб її можна було використати для перетворення слова або вашого повідомлення, потрібно використовувати atoi. Аtoi перетворює string у int.  Якщо у нас є рядок 12, ми використовуємо atoi на нього, або будь-які інші вхідні дані, і вона поверне ціле число. Не символ або рядок 12, які, коли почати додавати їх до чисел, будуть сильно відрізнятися, тому що рядок 12 має божевільне число в ASCII, а число 12 є насправді числом 12. Отже, використовуйте atoi.  Це вам знадобиться в шифрі Цезаря, тому що вам знадобиться ціле число, яке користувач вводить в командному рядку. Коли він його введе, то воно збережеться у вигляді рядка. Зрозуміло? Вам це може не знадобитися в шифрі Віженера. В шифрі Віженера, як я вже говорила, ви повинні спробувати математику ASCII, схожу на ці символи, які ми розглядали.  Ділення з остачею. Що робити, якщо у вас такий величезний номер для Цезаря? Ви розумієте, що, якщо ви на Z, а у вас номер два, це означає, що вам потрібна друга буква після Z, так? Ділення по модулю і є способом цього досягти. Воно повертає залишок від ділення першого числа на друге. Ось кілька прикладів для більшої конкретики. Але загалом ви маєте використовувати ділення по модулю, якщо хочете пройти алфавіт по колу.  Якщо вам потрібні тільки цифри від одного до восьми, ви можете використовувати ділення по модулю для будь-яких інших чисел – у вас завжди будуть числа від нуля до восьми. Приклади - якщо у нас є 55, розділене по модулю на 10, тоді остачею є 5. А потім 3, ділене по модулю на 5, хто здогадався, що це буде? Три.  Якщо у вас є менше число перед знаком ділення по модулю, він не може його поділити націло. Це нуль, тому число просто повертається. Тож 8, ділене по модулю на 8, буде?  СТУДЕНТ: Нуль.  ЕЛІСОН: Нуль. Тому що ділиться націло. 16, ділене по модулю на 15?  АУДИТОРІЯ: 1.  ЕЛІСОН: Тоді останній, просто для прикладу – як ви думаєте, який тут порядок операцій? Спочатку ділимо чи ділимо по модулю? Ділення по модулю має той же пріоритет, що й ділення чи множення, це лишається асоціативним. Той самий порядок. Ви завжди спочатку робите те, що в дужках, потім множення, ділення, і ділення по модулю зліва направо. Просто додайте цю операцію до тієї ж категорії, що ділення і множення.  У цьому випадку, 1 + 2 дає нам 3. Помножимо на 2, отримуємо 6. Ділимо по модулю на 2, отримуємо?  СТУДЕНТ: 0.  ЕЛІСОН: 0. Тоді ми додаємо 2, тому в останньому прикладі отримуємо 2. Щодо ділення по модулю - ви маєте подумати над тим, як пересуватися по алфавіту. Якщо ви знаходитесь на Z, ви маєте пересунутися вперед на три, щоб дістатися до C, - це і є концепція руху по колу. Це, безумовно, корисний інструмент для вашого практичного завдання на цьому тижні. Після того, як ви зробите це завдання, то зможете посилати своїм друзям секретні повідомлення і переконатися, що він буде розшифрувати його. Дуже весело.  Ось і кінець семінару. Я рано закінчила. Ще є 15 хвилин, тож якщо ви хочете пройти щось ще раз – я буду рада це зробити. Будь-які питання з приводу практичних завдань від тих, хто почав їх робити або читав специфікацію. Або ми можемо закінчити, і ви підете з цукерками. Ви також можете прийти і поговорити зі мною після семінару. Я обіцяю не кусатися. Всі відчувають, що впораються з цим практичним завданням? Все буде добре. Якщо що, є години консультацій.  Що ж, у такому випадку, дякую всім, що прийшли. Я сподіваюся побачити вас наступного тижня. Там буде багато цікавого, і я з нетерпінням чекаю можливості познайомитися з усіма вами цього року. |