In signing this statement, I hereby certify that the work on this exam is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exam and will be subject to possible disciplinary action.

Signature:

You must sign here. Otherwise you will receive a 2-point penalty.

Read the questions carefully.

This is an open-book, open-note exam.

Laptops, smartphones, calculators, and all other electronic devices are prohibited. You may not borrow books, notes, or anything else from other students during the exam.

The exam is 60 minutes long.

The Recursion Learning Objective is tested. You must obtain 50% or more points on question #2 to pass this objective.

The Structures Learning Objective is tested. You must obtain 50% or more points on the total of questions #1 and #3 to pass this objective.

The Files Learning Objective is tested. You must obtain 50% or more points on the total of parts 1, 2, and 4 of question #4 to pass this objective.
Contents

1 Struct types (2 points) ................................................. 3

2 Recursion: Recursive Function (6 points) ...................... 5
  2.1 Write the actual output (3 points) ............................. 6
  2.2 Correct the code (3 points) ...................................... 7

3 Sort Vector Array (6 points) ..................................... 8

4 Binary files (6 points) .............................................. 11

Learning Objective 1 (Recursion) Pass Fail
Learning Objective 2 (Structures) Pass Fail
Learning Objective 4 (Files) Pass Fail

Total Score: 2
1 Struct types (2 points)

In the following program, the `char_at_v1`, `char_at_v2`, `char_at_v3`, and `char_at_v4` functions all return the character at index `i` in the content of the given `Text` object, unless `i` is $\geq$ `length` field, in which case they return `other`.

Fill in the body of each one. The only difference between them is the arguments and the restrictions on how you must write them. The main function and sample output are on the next page.

Hint: Each of these can be written in a single line of code using the `(c ? a : b)` syntax. (If condition `c` is true, then the value of the expression is `a`. Otherwise, the value is `b`.)

typedef struct {
    int length;
    char* content;
} Text;

char char_at_v1(Text t, int i, char other) { /* 0.5 points */
    // No restrictions. Use any operators you like.
}

char char_at_v2(Text* p_t, int i, char other) { /* 0.5 points */
    // No restrictions. Use any operators you like.
}

char char_at_v3(Text t, int i, char other) { /* 0.5 points */
    // Restriction: Do not use the "." operator. (You may use ")->".)
}

char char_at_v4(Text* p_t, int i, char other) { /* 0.5 points */
    // Restriction: Do not use the "->" operator. (You may use ")

```c
int main(int argc, char** argv) {
    Text t = {3, "abc"};
    printf("char_at_v1(t, 0, '?') == '%c'\n", char_at_v1(t, 0, '?'));
    printf("char_at_v1(t, 5, '?') == '%c'\n", char_at_v1(t, 5, '?'));
    printf("char_at_v2(&t, 0, '?') == '%c'\n", char_at_v2(&t, 0, '?'));
    printf("char_at_v2(&t, 5, '?') == '%c'\n", char_at_v2(&t, 5, '?'));
    printf("char_at_v3(t, 0, '?') == '%c'\n", char_at_v3(t, 0, '?'));
    printf("char_at_v3(t, 5, '?') == '%c'\n", char_at_v3(t, 5, '?'));
    printf("char_at_v4(&t, 0, '?') == '%c'\n", char_at_v4(&t, 0, '?'));
    printf("char_at_v4(&t, 5, '?') == '%c'\n", char_at_v4(&t, 5, '?'));
    return 0;
}
```

Output:

```
c char_at_v1(t, 0, '?') == 'a'
c char_at_v1(t, 5, '?') == '?'
c char_at_v2(&t, 0, '?') == 'a'
c char_at_v2(&t, 5, '?') == '?'
c char_at_v3(t, 0, '?') == 'a'
c char_at_v3(t, 5, '?') == '?'
c char_at_v4(&t, 0, '?') == 'a'
c char_at_v4(&t, 5, '?') == '?'
```

**ANSWER**

```c
char char_at_v1(Text t, int i, char other) {        /* 1 point */
    return (i < t.length ? t.content[i] : other);
}
```

```c
char char_at_v2(Text* p_t, int i, char other) {    /* 1 point */
    return (i < p_t->length ? p_t->content[i] : other);
}
```

```c
char char_at_v3(Text t, int i, char other) {       /* 1 point */
    return (i < (&t)->length ? (&t)->content[i] : other);
}
```

```c
char char_at_v4(Text* p_t, int i, char other) {    /* 1 point */
    return (i < (*p_t).length ? (*p_t).content[i] : other);
}
```
2 Recursion: Recursive Function (6 points)

The following is an attempt to print integer partitions of monotonically decreasing numbers. However, this program has one (or several) mistakes. (The program has no syntax errors.) The intended output is shown on the next page.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void printPartition(int * arr, int length) {
    int ind;
    for (ind = 0; ind < length - 1; ind ++)
        printf("%d␣+␣", arr[ind]);
    printf("%d\n", arr[length - 1]);
}

void partition(int * arr, int ind, int left) {
    int val;
    if (left == 0)
        printPartition(arr, ind);
    return;
    int max = (ind == 0 ? left : arr[ind] - 1);
    for (val = 1; val <= max; val ++)
        arr[ind] = val;
    partition(arr, ind + 1, left - val);
}

int main(int argc, char * argv[]) {
    int n;
    for (n = 1; n <= 8; n ++)
        int * arr = malloc(sizeof(int) * n);
    printf("---partition␣%d---\n", n);
    partition(arr, 0, n);
    free (arr);
    return EXIT_SUCCESS;
}
```
2.1 Write the actual output (3 points)

You may modify the output above, if you think that is appropriate, or else write the output separately to the side. If the program has a run-time error, write the output before the error occurs and then describe the error (e.g., "invalid memory access", etc.).
Due to a human error just before printing, the exam had an unintentional bug in line 18 that made this unanswerable. All students received full credit (3 points) for part 1.

Line 18, as printed (with both intentional and unintentional bugs):

```c
int max = (ind == 0 ? left : arr[ind] - 1);
```

With that, the output is unpredictable because arr[ind] will contain garbage.

Line 18, as intended (still with an intentional bug planted):

```c
int max = (ind == 0 ? left : arr[ind - 1]);
```

With that, the answer would have been as follows:

---partition 1---
1
---partition 2---
1 + 1
2
---partition 3---
1 + 1 + 1
2 + 1
Segmentation fault (core dumped)

The recursion does not end and the ind becomes too large.

2.2 Correct the code (3 points)

Mark the corrections on the code on the previous page so that it matches the intended output. Your corrections should be at most a few lines. If you write a lot of code, your answer is probably wrong. The code has no syntax errors.

The corrections are as follows (lines 18-19):

```c
int max = (ind == 0 ? left + 1 : arr[ind - 1]);
for (val = 1; val < max; val ++) {
```
3 Sort Vector Array (6 points)

This question asks you to implement several functions for the Vector structure. The structure is defined as follows:

```c
#ifndef _VECTOR_H_
#define _VECTOR_H_
typedef struct
{
    double x;
    double y;
    double z;
} Vector;
double Vector_innerProduct(Vector v1, Vector v2);
double Vector_length(Vector v);
void Vector_sortByLength(Vector *arr, int len);
#endif
```

Please fill the blank. Do not worry about the main function.

```c
#include "vector.h"
#include <stdlib.h>
#include <math.h>
// Suppose v1's attributes are (x1, y1, z1) and
// v2's attributes are (x2, y2, z2).
// Return x1 * x2 + y1 * y2 + z1 * z2.
// 1 point
double Vector_innerProduct(Vector v1, Vector v2)
{
}
```

```c
// Suppose v's attributes are (x, y, z).
// Return sqrt(x * x + y * y + z * z).
// 1 point
double Vector_length(Vector v)
{
}
```
// Sort an array of Vector objects by their lengths in ascending order. This function MUST use qsort. You will need to write the comparison function, which will be called by qsort.
// 2 points
int Vector_compare(const void * arg1, const void * arg2)
{
    // comparison code
}
// 2 points
void Vector_sortByLength(Vector *arr, int len)
{
    // sorting code
}

qsort reference

The following is an excerpt of qsort’s manual for your reference.

```c
void qsort(void *base, size_t nmemb, size_t size,
            int (*compar)(const void *, const void *));
```

The qsort() function sorts an array with nmemb elements of size size. The base argument points to the start of the array.

The contents of the array are sorted in ascending order according to a comparison function pointed to by compar, which is called with two arguments that point to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.
#include "vector.h"
#include <math.h>
#include <stdlib.h>

double Vector_innerProduct(Vector v1, Vector v2) {
    return v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
}

double Vector_length(Vector v) {
    return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}

int Vector_compare(const void * arg1, const void * arg2) {
    const Vector * ptr1 = (const Vector *) arg1;
    const Vector * ptr2 = (const Vector *) arg2;
    const Vector v1 = * ptr1;
    const Vector v2 = * ptr2;
    double len1 = Vector_length(v1);
    double len2 = Vector_length(v2);
    if (len1 < len2) {
        return -1;
    }
    if (len1 > len2) {
        return 1;
    }
    return 0;
}

void Vector_sortByLength(Vector *arr, int len) {
    qsort(arr, len, sizeof(Vector), Vector_compare);
}
4 Binary files (6 points)

Consider the following program.

```c
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

typedef struct {
    uint8_t x;
    uint8_t y;
    uint8_t z;
} S;

int main(int argc, char *argv[]) {
    S o;
    o.x = 100;
    o.y = 101;
    o.z = 102;
    fwrite(&o, sizeof(S), 1, stdout);

    FILE* fp = fopen("file.txt", "wb");
    if(fp != NULL) {
        fwrite("mo", sizeof(char), 2, fp);
        fwrite(&o, sizeof(S), 1, fp);
        fclose(fp);

        fp = fopen("file.txt", "rb");
        if(fp != NULL) {
            fgetc(fp);
            fread(&o, sizeof(S), 1, fp);
            fclose(fp);
        }
    }
    fwrite(&o, sizeof(S), 1, stdout);
    return EXIT_SUCCESS;
}
```
An ASCII table and GDB reference card are included at the end of this exam.

You may assume the following:

- A character occupies one byte, i.e., `sizeof(char) == 1`.
- Opening the file is successful, i.e., `fopen()` never returns `NULL`.
- Compiler adds no extra padding to structs, i.e., `#pragma pack(1)`.

1. What does does the code print (if anything)? (3 points)

   ANSWER
   defode

2. After running this program, what will be the output of `cat file.txt`? (1 point)

   ANSWER
   modef

3. Suppose you would like to use gdb to view the memory occupied by `o` in hex format. You would like to examine that memory at line 17 and again at line 29. Write the `gdb commands` you would use to accomplish this within gdb.

   Hint: You will need to use the `x` command twice. (1 point)

   ANSWER
   b 17
   b 29
   r
   x/3xb \&o
   c
   x/3xb \&o

4. Fill in the output of the two `x` commands. (1 point)

   ANSWER
   contents of memory occupied by o at line 17
   0x7fffffffdd80: 0x64 0x65 0x66
   contents of memory occupied by o at line 29
0x7fffffffdd80: 0x6f  0x64  0x65
# ASCII Table

<table>
<thead>
<tr>
<th>Char Dec Hex</th>
<th>Char Dec Hex</th>
<th>Char Dec Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>' ' 32 20</td>
<td>' @ ' 64 40</td>
<td>' ' 96 60</td>
</tr>
<tr>
<td>! ' 33 21</td>
<td>' A ' 65 41</td>
<td>' a ' 97 61</td>
</tr>
<tr>
<td>, ' 34 22</td>
<td>' B ' 66 42</td>
<td>' b ' 98 62</td>
</tr>
<tr>
<td># ' 35 23</td>
<td>' C ' 67 43</td>
<td>' c ' 99 63</td>
</tr>
<tr>
<td>$ ' 36 24</td>
<td>' D ' 68 44</td>
<td>' d ' 100 64</td>
</tr>
<tr>
<td>, , 37 25</td>
<td>' E ' 69 45</td>
<td>' e ' 101 65</td>
</tr>
<tr>
<td>&amp; ' 38 26</td>
<td>' F ' 70 46</td>
<td>' f ' 102 66</td>
</tr>
<tr>
<td>' ' 39 27</td>
<td>' G ' 71 47</td>
<td>' g ' 103 67</td>
</tr>
<tr>
<td>( ' 40 28</td>
<td>' H ' 72 48</td>
<td>' h ' 104 68</td>
</tr>
<tr>
<td>) ' 41 29</td>
<td>' I ' 73 49</td>
<td>' i ' 105 69</td>
</tr>
<tr>
<td>* ' 42 2A</td>
<td>' J ' 74 4A</td>
<td>' j ' 106 6A</td>
</tr>
<tr>
<td>+ ' 43 2B</td>
<td>' K ' 75 4B</td>
<td>' k ' 107 6B</td>
</tr>
<tr>
<td>, ' 44 2C</td>
<td>' L ' 76 4C</td>
<td>' l ' 108 6C</td>
</tr>
<tr>
<td>- ' 45 2D</td>
<td>' M ' 77 4D</td>
<td>' m ' 109 6D</td>
</tr>
<tr>
<td>, , 46 2E</td>
<td>' N ' 78 4E</td>
<td>' n ' 110 6E</td>
</tr>
<tr>
<td>/ ' 47 2F</td>
<td>' O ' 79 4F</td>
<td>' o ' 111 6F</td>
</tr>
<tr>
<td>0 ' 48 30</td>
<td>' P ' 80 50</td>
<td>' p ' 112 70</td>
</tr>
<tr>
<td>' 49 31</td>
<td>' Q ' 81 51</td>
<td>' q ' 113 71</td>
</tr>
<tr>
<td>2 ' 50 32</td>
<td>' R ' 82 52</td>
<td>' r ' 114 72</td>
</tr>
<tr>
<td>3 ' 51 33</td>
<td>' S ' 83 53</td>
<td>' s ' 115 73</td>
</tr>
<tr>
<td>4 ' 52 34</td>
<td>' T ' 84 54</td>
<td>' t ' 116 74</td>
</tr>
<tr>
<td>5 ' 53 35</td>
<td>' U ' 85 55</td>
<td>' u ' 117 75</td>
</tr>
<tr>
<td>6 ' 54 36</td>
<td>' V ' 86 56</td>
<td>' v ' 118 76</td>
</tr>
<tr>
<td>7 ' 55 37</td>
<td>' W ' 87 57</td>
<td>' w ' 119 77</td>
</tr>
<tr>
<td>8 ' 56 38</td>
<td>' X ' 88 58</td>
<td>' x ' 120 78</td>
</tr>
<tr>
<td>9 ' 57 39</td>
<td>' Y ' 89 59</td>
<td>' y ' 121 79</td>
</tr>
<tr>
<td>, ' 58 3A</td>
<td>' Z ' 90 5A</td>
<td>' z ' 122 7A</td>
</tr>
<tr>
<td>; ' 59 3B</td>
<td>' [ ' 91 5B</td>
<td>' { ' 123 7B</td>
</tr>
<tr>
<td>&lt; ' 60 3C</td>
<td>' \ ' 92 5C</td>
<td>'</td>
</tr>
<tr>
<td>= ' 61 3D</td>
<td>' ] ' 93 5D</td>
<td>' } ' 125 7D</td>
</tr>
<tr>
<td>&gt; ' 62 3E</td>
<td>' ^ ' 94 5E</td>
<td>' ~ ' 126 7E</td>
</tr>
<tr>
<td>? ' 63 3F</td>
<td>' _ ' 95 5F</td>
<td></td>
</tr>
</tbody>
</table>
---

### Essential Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gdb program [core]</code></td>
<td>debug program [using coredump core]</td>
</tr>
<tr>
<td><code>b [file]:function</code></td>
<td>set breakpoint at function [in file]</td>
</tr>
<tr>
<td><code>run</code></td>
<td>start your program with arglist</td>
</tr>
<tr>
<td><code>bt</code></td>
<td>backtrace: display program stack</td>
</tr>
<tr>
<td><code>p expr</code></td>
<td>display the value of an expression</td>
</tr>
<tr>
<td><code>c</code></td>
<td>continue running your program</td>
</tr>
<tr>
<td><code>n</code></td>
<td>next line, stepping over function calls</td>
</tr>
<tr>
<td><code>s</code></td>
<td>next line, stepping into function calls</td>
</tr>
</tbody>
</table>

### Starting GDB

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gdb</code></td>
<td>start GDB, with no debugging files</td>
</tr>
<tr>
<td><code>gdb program</code></td>
<td>begin debugging program</td>
</tr>
<tr>
<td><code>gdb program core</code></td>
<td>debug coredump core produced by program</td>
</tr>
<tr>
<td><code>gdb --help</code></td>
<td>describe command line options</td>
</tr>
</tbody>
</table>

### Stopping GDB

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>quit</code></td>
<td>exit GDB; also q or EOF (eg C-d) (eg C-c) terminate current command, or send to running process</td>
</tr>
</tbody>
</table>

### Getting Help

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>help</code></td>
<td>list classes of commands</td>
</tr>
<tr>
<td><code>help class</code></td>
<td>one-line descriptions for commands in class</td>
</tr>
<tr>
<td><code>help command</code></td>
<td>describe command</td>
</tr>
</tbody>
</table>

### Executing Your Program

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>run arglist</code></td>
<td>start your program with arglist</td>
</tr>
<tr>
<td><code>run</code></td>
<td>start your program with current argument list</td>
</tr>
<tr>
<td><code>run ... &lt;inf&gt; &gt;outf</code></td>
<td>start your program with input, output redirected</td>
</tr>
<tr>
<td><code>kill</code></td>
<td>kill running program</td>
</tr>
<tr>
<td><code>tty dev</code></td>
<td>use dev as stdin and stdout for next run</td>
</tr>
<tr>
<td><code>set args arglist</code></td>
<td>specify arglist for next run</td>
</tr>
<tr>
<td><code>set args</code></td>
<td>specify empty argument list</td>
</tr>
<tr>
<td><code>show args</code></td>
<td>display argument list</td>
</tr>
<tr>
<td><code>show env</code></td>
<td>show all environment variables</td>
</tr>
<tr>
<td><code>show env var string</code></td>
<td>set environment variable var</td>
</tr>
<tr>
<td><code>unset env var</code></td>
<td>remove var from environment</td>
</tr>
</tbody>
</table>

### Shell Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cd dir</code></td>
<td>change working directory to dir</td>
</tr>
<tr>
<td><code>pwd</code></td>
<td>Print working directory</td>
</tr>
<tr>
<td><code>make ...</code></td>
<td>call <code>make</code></td>
</tr>
<tr>
<td><code>shell cmd</code></td>
<td>execute arbitrary shell command string</td>
</tr>
</tbody>
</table>

---

### Breakpoints and Watchpoints

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>break [file]:line</code></td>
<td>set breakpoint at line number [in file]</td>
</tr>
<tr>
<td><code>b [file]:line</code></td>
<td>eg <code>break main.c:37</code></td>
</tr>
<tr>
<td><code>break [file]:func</code></td>
<td>set breakpoint at function [in file]</td>
</tr>
<tr>
<td><code>break +offset</code></td>
<td>set break at offset lines from current stop</td>
</tr>
<tr>
<td><code>break -offset</code></td>
<td>set break at offset lines from current stop</td>
</tr>
<tr>
<td><code>break addr</code></td>
<td>set breakpoint at address addr</td>
</tr>
<tr>
<td><code>break ... if expr</code></td>
<td>break conditionally on nonzero expr</td>
</tr>
<tr>
<td><code>cond n [expr]</code></td>
<td>new conditional expression on breakpoint n; make unconditional if no expr</td>
</tr>
<tr>
<td><code>tbreak</code></td>
<td>temporary break; disable when reached</td>
</tr>
<tr>
<td><code>rbreak Regex</code></td>
<td>break on all functions matching regex</td>
</tr>
<tr>
<td><code>watch expr</code></td>
<td>set a watchpoint for expression expr</td>
</tr>
<tr>
<td><code>catch event</code></td>
<td>break at event, which may be catch, throw, exec, fork, vfork, load, or unload</td>
</tr>
<tr>
<td><code>info break</code></td>
<td>show defined breakpoints</td>
</tr>
<tr>
<td><code>info watch</code></td>
<td>show defined watchpoints</td>
</tr>
<tr>
<td><code>clear</code></td>
<td>delete breakpoints at next instruction</td>
</tr>
<tr>
<td><code>clear [file]:fun</code></td>
<td>delete breakpoints at entry to fun()</td>
</tr>
<tr>
<td><code>clear [file]:line</code></td>
<td>delete breakpoints on source line</td>
</tr>
<tr>
<td><code>delete n</code></td>
<td>delete breakpoints [or breakpoint n]</td>
</tr>
<tr>
<td><code>disable n</code></td>
<td>disable breakpoints [or breakpoint n]</td>
</tr>
<tr>
<td><code>enable n</code></td>
<td>enable breakpoints [or breakpoint n]</td>
</tr>
<tr>
<td><code>enable once n</code></td>
<td>enable breakpoints [or breakpoint n]; disable again when reached</td>
</tr>
<tr>
<td><code>enable del n</code></td>
<td>delete when reached</td>
</tr>
<tr>
<td><code>ignore n count</code></td>
<td>ignore breakpoint n, count times</td>
</tr>
<tr>
<td><code>commands n</code></td>
<td>execute GDB command-list every time breakpoint n is reached. silent suppresses default display</td>
</tr>
<tr>
<td><code>command-list n</code></td>
<td>end of command-list</td>
</tr>
</tbody>
</table>

### Program Stack

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>backtrace n</code></td>
<td>print trace of all frames in stack; or of n frames—innermost if &gt;0, outermost if n&lt;0</td>
</tr>
<tr>
<td><code>bt n</code></td>
<td>select frame number n or frame at address n; if no n, display current frame</td>
</tr>
<tr>
<td><code>frame n</code></td>
<td>select frame n frames up</td>
</tr>
<tr>
<td><code>up n</code></td>
<td>select frame n frames down</td>
</tr>
<tr>
<td><code>down n</code></td>
<td></td>
</tr>
<tr>
<td><code>info frame [addr]</code></td>
<td>describe selected frame, or frame at addr</td>
</tr>
<tr>
<td><code>info args</code></td>
<td>arguments of selected frame</td>
</tr>
<tr>
<td><code>info locals</code></td>
<td>local variables of selected frame</td>
</tr>
<tr>
<td><code>info reg [rn]</code></td>
<td>register values for regs rn in selected frame; all-reg includes floating point</td>
</tr>
<tr>
<td><code>info all-reg [rn]</code></td>
<td></td>
</tr>
</tbody>
</table>

### Execution Control

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>continue [count]</code></td>
<td>continue running; if count specified, ignore this breakpoint next count times</td>
</tr>
<tr>
<td><code>c [count]</code></td>
<td></td>
</tr>
<tr>
<td><code>step [count]</code></td>
<td>execute until another line reached; repeat count times if specified</td>
</tr>
<tr>
<td><code>s [count]</code></td>
<td>step by machine instructions rather than source lines</td>
</tr>
<tr>
<td><code>stepi [count]</code></td>
<td>execute next line, including any function calls</td>
</tr>
<tr>
<td><code>next [count]</code></td>
<td>next machine instruction rather than source line</td>
</tr>
<tr>
<td><code>nexti [count]</code></td>
<td></td>
</tr>
<tr>
<td><code>ni [count]</code></td>
<td></td>
</tr>
<tr>
<td><code>until [location]</code></td>
<td>run until next instruction (or location)</td>
</tr>
<tr>
<td><code>finish</code></td>
<td>run until selected stack frame returns</td>
</tr>
<tr>
<td><code>return expr</code></td>
<td>pop selected stack frame without executing [setting return value]</td>
</tr>
<tr>
<td><code>signal num</code></td>
<td>resume execution with signal s (none if 0)</td>
</tr>
<tr>
<td><code>jump line</code></td>
<td>resume execution at specified line number or address</td>
</tr>
<tr>
<td><code>set var=expr</code></td>
<td>evaluate expr without displaying it; use for altering program variables</td>
</tr>
</tbody>
</table>

### Display

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print [file]:expr</code></td>
<td>show value of expr [or last value] according to format f</td>
</tr>
<tr>
<td><code>p [file]:expr</code></td>
<td>x hexadecimal</td>
</tr>
<tr>
<td><code>x [file]:expr</code></td>
<td>signed decimal</td>
</tr>
<tr>
<td><code>u [file]:expr</code></td>
<td>unsigned decimal</td>
</tr>
<tr>
<td><code>c [file]:expr</code></td>
<td>octal</td>
</tr>
<tr>
<td><code>t [file]:expr</code></td>
<td>binary</td>
</tr>
<tr>
<td><code>a</code></td>
<td>address, absolute and relative</td>
</tr>
<tr>
<td><code>c</code></td>
<td>character</td>
</tr>
<tr>
<td><code>f</code></td>
<td>floating point</td>
</tr>
<tr>
<td><code>call [file]:expr</code></td>
<td>like print but does not display void</td>
</tr>
<tr>
<td><code>x [file]:expr</code></td>
<td>examine memory at address expr; optional format spec follows slash</td>
</tr>
<tr>
<td><code>disassemble</code></td>
<td>display memory as machine instructions</td>
</tr>
<tr>
<td><code>disassemble addr</code></td>
<td></td>
</tr>
<tr>
<td><code>disassemble</code></td>
<td>display memory as machine instructions</td>
</tr>
</tbody>
</table>

### Automatic Display

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>display [file]:expr</code></td>
<td>show value of expr each time program stops [according to format f]</td>
</tr>
<tr>
<td><code>dump</code></td>
<td>display all enabled expressions on list</td>
</tr>
<tr>
<td><code>dump n</code></td>
<td>remove number(s) n from list of automatically displayed expressions</td>
</tr>
<tr>
<td><code>disable disp n</code></td>
<td>disable display for expression(s) number n</td>
</tr>
<tr>
<td><code>enable disp n</code></td>
<td>enable display for expression(s) number n</td>
</tr>
<tr>
<td><code>info display</code></td>
<td>numbered list of display expressions</td>
</tr>
</tbody>
</table>
Expressions

expr an expression in C, C++, or Modula-2 (including function calls), or:
addr@len an array of len elements beginning at addr
file[:nm] read memory at addr as specified type
$n nth displayed value
$displayed value previous to $n
$@len nth displayed value back from $n
$x value at address x
$var convenience variable; assign any value

show values show last 10 values (or surrounding $n)
show conv display all convenience variables

Symbol Table

info address show where symbol is stored
info func show names, types of defined functions
info var show names, types of global variables
info regex show regex
info whatis data type of expr without evaluating
info ptype describe type, struct, union, or enum

GDB Scripts

source script read, execute GDB commands
define cmd create new GDB command
end command-list end of command-list
document cmd create online documentation
document help-text end of help-text

Signals

handle signal act specify GDB actions for signal:
print announce signal
noprint be silent for signal
stop halt execution on signal
nostop do not halt execution
pass allow your program to handle signal
nopass do not allow your program to see signal
info signals show table of signals, GDB action for each

Debugging Targets

target type param connect to target machine, process, or file
display available targets
help target connect to another process
attach param release target from GDB control
detach

Controlling GDB

set param value set one of GDB's internal parameters
show param display current setting of parameter

Parameters understood by set and show:
complaint brak number of messages on unusual symbols
confirm on/off enable or disable cautionary queries
editing readline on/off control readline command-line editing
evaluating height tpp number of lines before pause in display
language long Language for GDB expressions (auto, c or modula-2)
listsize n number of lines shown by list
prompt str use str as GDB prompt
radix base set one of GDB's internal parameters
verbose on/off control messages when loading symbols
width cpl number of characters before line folded
write on/off Allow or forbid patching binary, core files
           (when recompiled with exec or core)
history ... groups with the following options:
   h disable/enable readline history expansion
   h file filename file for recording GDB command history
   h size size number of commands kept in history list
   h save off/on control use of external file for command history
   print ... groups with the following options:
      p ... p address off/on print memory addresses in stacks, values
      p array off/on compact or attractive format for arrays
      p demangle off/on source (demangled) or internal form for
      p asm-dem off/on demangle C++ symbols
      p objects off/on number of array elements to display
      p object on/off print C++ derived types for objects
      p pretty on/off struct display: compact or indented
      p union on/off display of union members
      p vtbl on/off display of C++ virtual functions
   show commands show last 10 commands
   show commands n show 10 commands around number n
   show commands + show next 10 commands

Working Files

file [file] use file for both symbols and executable;
           with no arg, discard both
file core [file] read file as coredump; or discard
file exec [file] use file as executable only; or discard
file symbol [file] use symbol table from file; or discard
file load [file] dynamically link file and add its symbols
file add-sym [file addr] add additional symbols from file,
file info [file addr] display working files and targets in use
info files add dirs to front of path searched for executable and symbol files
path dirs show path display executable and symbol file path
list names of shared libraries currently loaded

Source Files

dir names add directory names to front of source path
dir clear source path
show dir show current source path
list show next ten lines of source
list - show previous ten lines
list lines display source lines
[file:] num line number in named file
[file:] function beginning of function in named file
offlines off lines after last printed
nofflines off lines previous to last printed
adress line containing address
list fl show starting, ending addresses of compiled code for source line num
info source show name of current source file
info sources list all source files in use
forw regex search following source lines for regex
rew regex search preceding source lines for regex

GDB under GNU Emacs

M-x gdb run GDB under Emacs
C-h m describe GDB mode
M-s step one line (step)
M-n next line (next)
M-i step one instruction (stepi)
C-c C-f finish current stack frame (finish)
C-c cont (cont)
C-u up arg frames (up)
M-d down arg frames (down)
C-x & C-x SPC copy number from point, insert at end

GDB License

show copying Display GNU General Public License
show warranty There is NO WARRANTY for GDB.

Display full no-warranty statement.

Roland H. Pesch

The author assumes no responsibility for any errors on this card.

This card may be freely distributed under the terms of the GNU General Public License.

Please contribute to development of this card by annotating it.
Improvements can be sent to bug-gdb@gnu.org.

GDB itself is free software; you are welcome to distribute copies of it under the terms of the GNU General Public License. There is absolutely no warranty for GDB.